

# Package: quitte (via r-universe)

January 4, 2025

**Type** Package

**Title** Bits and pieces of code to use with quitte-style data frames

**Version** 0.3139.0

**Date** 2024-12-05

**Maintainer** Michaja Pehl <michaja.pehl@pik-potsdam.de>

**Description** A collection of functions for easily dealing with quitte-style data frames, doing multi-model comparisons and plots.

**License** GPL-2

**URL** <https://github.com/pik-piam/quitte>

**BugReports** <https://github.com/pik-piam/quitte/issues>

**Depends** R (>= 4.0)

**Imports** cli, countrycode, dplyr (>= 1.1.1), forcats (>= 1.0.0), ggplot2, gms (>= 0.17.0), glue, lazyeval, lifecycle, lubridate, magclass, magrittr, methods, plyr, purrr, readr, readxl, reshape2, rlang, stats, stringr, tibble, tidyr, tidyselect, writexl, zoo,

**Suggests** covr, gamstransfer, gdxrrw, knitr, mip, rmarkdown, testthat (>= 3.2.0), tidyverse, withr,

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** TRUE

**RoxygenNote** 7.3.2

**Roxygen** list(markdown = TRUE)

**Config/pak/sysreqs** libicu-dev libx11-dev zlib1g-dev

**Repository** <https://pik-piam.r-universe.dev>

**RemoteUrl** <https://github.com/pik-piam/quitte>

**RemoteRef** HEAD

**RemoteSha** db1855fa4d0a12b8697949320f08fe9d1ec14a34

## Contents

quitte-package . . . . .	3
add_countrycode . . . . .	4
add_timesteps_columns . . . . .	5
aggregate_map . . . . .	7
bin_distribute . . . . .	10
calcAddVariable . . . . .	11
calcCumulatedDiscount . . . . .	12
calcDecompEff . . . . .	13
calcMitigationCost . . . . .	14
calc_addVariable . . . . .	16
calc_growthrate . . . . .	18
calc_mode . . . . .	19
calc_quantiles . . . . .	20
cartesian . . . . .	21
character.data.frame . . . . .	22
check_quitte . . . . .	23
chooseFilter . . . . .	25
df.2.named.vector . . . . .	26
df.as.list . . . . .	26
df_populate_range . . . . .	27
df_variation . . . . .	28
duplicate . . . . .	28
factor.data.frame . . . . .	29
factorise . . . . .	30
getColValues . . . . .	31
getModels . . . . .	31
getPeriods . . . . .	32
getRegs . . . . .	33
getScenarios . . . . .	33
getVars . . . . .	34
ggplot_bar_stacked_dodged . . . . .	35
gg_colour_hue . . . . .	36
inline.data.frame . . . . .	37
interpolate_missing_periods . . . . .	37
ISOyear . . . . .	39
list_to_data_frame . . . . .	40
madrat_mule . . . . .	41
magclass_to_tibble . . . . .	42
mif2rds . . . . .	42
mutate_text . . . . .	43
order.levels . . . . .	43
overwrite . . . . .	44
prepQuitteForScatter . . . . .	45
quitte-class . . . . .	46
quitte2quantiles . . . . .	46
quitteSort . . . . .	47

quitte_example_data . . . . .	48
quitte_example_dataAR6 . . . . .	48
read.gdx . . . . .	49
read.quitte . . . . .	50
read.snapshot . . . . .	51
read_mif_header . . . . .	52
remind_timesteps . . . . .	53
removeColNa . . . . .	54
replace_column . . . . .	54
reportDuplicates . . . . .	56
revalue.levels . . . . .	57
seq_range . . . . .	58
setXor . . . . .	59
signrange . . . . .	59
strtrimcommon . . . . .	60
sum_total . . . . .	61
toolExtractSortScaleQuitte . . . . .	62
unique_or_levels . . . . .	63
write.IAMCxlsx . . . . .	64
write.mif . . . . .	64

**Index** **66**

quitte-package *The quitte R package*

**Description**

Bits and pieces for easier handling of quitte-style data frames.

**Author(s)**

Michaja Pehl

**See Also**

Useful links:

- <https://github.com/pik-piam/quitte>
- Report bugs at <https://github.com/pik-piam/quitte/issues>

---

add_countrycode	<i>Add country code</i>
-----------------	-------------------------

---

### Description

Wrapper function for `countrycode::countrycode()` enabling piped execution.

### Usage

```
add_countrycode(data, ..., warn = TRUE, na.rm = FALSE)
```

```
add_countrycode_(data, origin, destination, warn = TRUE, na.rm = FALSE)
```

### Arguments

<code>data</code>	A data frame.
<code>...</code>	Key-value pairs for NSE of origin and destination parameters.
<code>warn</code>	Prints unique elements from sourcevar for which no match was found.
<code>na.rm</code>	If TRUE, remove ambiguously matched rows.
<code>origin</code>	Named scalar linking source column to source coding scheme. See <code>countrycode::countrycode()</code> for details.
<code>destination</code>	Named scalar linking destination column name to destination coding scheme. See <code>countrycode::countrycode()</code> for details.

### Value

A data frame.

### Author(s)

Michaja Pehl

### Examples

```
library(dplyr)
data <- tibble(
  country = c('Belgium', 'Narnia', 'Russia', 'Botswana'),
  data     = 1:4)

data %>% add_countrycode(country = country.name, m49.code = un)
data %>% add_countrycode_(c('country' = 'country.name'), 'iso3c',
  warn = FALSE, na.rm = TRUE)
```

---

add\_timesteps\_columns *Bar plots with variable time steps*

---

### Description

Utility functions for plotting stacked bars with variable widths for displaying time-series data with variable time steps (like REMIND data).

### Usage

```
add_timesteps_columns(
  data,
  timesteps,
  periods = "period",
  gaps = 0,
  interval_shift = c(-0.5, 0.5),
  timesteps_period = "period",
  timesteps_interval = "year"
)

add_remind_timesteps_columns(data, periods = "period", gaps = 0)

ggplot_bar_vts(
  data,
  timesteps,
  mapping = aes(x = !!sym("period"), y = !!sym("value"), fill = !!sym("variable")),
  gaps = 0.1,
  position_fill = FALSE,
  interval_shift = c(-0.5, 0.5),
  timesteps_period = "period",
  timesteps_interval = "year"
)

ggplot_bar_remind_vts(
  data,
  mapping = aes(x = !!sym("period"), y = !!sym("value"), fill = !!sym("variable")),
  gaps = 0.1,
  position_fill = FALSE
)
```

### Arguments

data	A data frame.
timesteps	A data frame like <a href="#">remind_timesteps</a> with columns period, year, and weight, where weight determines which share of year belongs to period.
periods	The column holding the period information in data (either a string or an object). Defaults to 'period'.

gaps	Gaps between bars as a fraction of the smallest bar width. Defaults to 0.1 (e.g. $0.1 * 5 \text{ years} = 0.5 \text{ years}$ ).
interval_shift	numeric of length 2. Shifts added to the interval fix point to obtain the beginning and end of time interval. If the interval for period 1 should be $[0.5, 1.5]$ , interval_shift should be set to $c(-0.5, 0.5)$ (default). If the interval for period 1 should be $[0, 1]$ , interval_shift should be set to $c(-1, 0)$ .
timesteps_period	character string giving the column name of the period in the timesteps data frame. Defaults to 'period'.
timesteps_interval	character string giving the column name of the time interval in the timesteps data frame. Defaults to 'year'.
mapping	<code>aes()</code> mapping with aesthetics x, y, and optionally fill.
position_fill	If TRUE, stacks bars and standardises each stack to have constant height.

### Details

`add_timesteps_columns()` calculates the x-axis position and width of bars based on the information in timesteps and joins it to data. `add_remind_timesteps_columns()` uses the `remind_timesteps` data frame. `ggplot_bar_vts()` produces a bar plot with bars positioned according to timesteps. `ggplot_bar_remind_vts()` uses the `remind_timesteps` data frame.

### Value

`add_timesteps_columns()` and `add_remind_timesteps_columns()` return a data frame. `ggplot_bar_vts()` and `ggplot_bar_remind_vts()` return a `ggplot()`-like object.

### Author(s)

Michaja Pehl

### Examples

```
require(tidyverse)

# some example data
(data <- quitte_example_data %>%
  filter(first(scenario) == scenario,
         last(region) == region,
         first(variable) == variable))

# adding individual timesteps
add_timesteps_columns(data, remind_timesteps)

# adding remind timesteps with gaps
add_remind_timesteps_columns(data, gaps = 0.1)

# plotting individual timesteps without gaps
ggplot_bar_vts(data, remind_timesteps, gaps = 0)
```

```

# plotting remind timegaps, using further ggplot2 functions
ggplot_bar_remind_vts(
  data = quitte_example_data %>%
    filter(scenario %in% levels(quitte_example_data$scenario)[1:3],
           last(region) == region,
           grepl('PE\\|', variable),
           2100 >= period)) +
  scale_fill_manual(
    values = mip::plotstyle(grep('^PE\\|',
                                 levels(quitte_example_data$variable),
                                 value = TRUE))) +
  facet_wrap(~ scenario)

# another data set with a different time column
data2 <- data.frame(variable = c('Wind', 'Solar', 'Wind', 'Solar'),
                    tau = c(1,1,2,2),
                    value = 1:4)

# some timesteps dataframe with hourly data
timesteps <- data.frame(tau = c(rep(1,2),rep(2,4)),
                       hour = 1:6,
                       weight = 1)

# plotting with different timesteps than periods and years
ggplot_bar_vts(data2, timesteps,
               mapping = aes(tau, value, group = variable, fill = variable),
               timesteps_period = 'tau',
               timesteps_interval = 'hour',
               interval_shift = c(-1,0))

```

---

 aggregate\_map

*Aggregates or disaggregates a data frame*


---

## Description

Aggregates or disaggregates the values of a data frame according to a mapping

## Usage

```

aggregate_map(
  data,
  mapping,
  by,
  subset2agg = NULL,
  only.new = TRUE,
  na.rm = TRUE,
  weights = NULL,
  forceAggregation = FALSE,
  autodetect = "auto",

```

```

    scaleWeights = TRUE,
    variable = "variable",
    value = "value",
    unit = "unit",
    weight_val_col = "weight_val_col",
    weight_item_col = NULL,
    fun = sum
  )

```

## Arguments

<code>data</code>	a data frame.
<code>mapping</code>	a data frame connecting the resolution in "data" and the wished resolution
<code>by</code>	(named) vector giving the correspondence between the column name of "data" and of the "mapping"
<code>subset2agg</code>	subset of variables for which the (dis)aggregation is applied. If NULL (the default), the (dis)aggregation is applied to all variables.
<code>only.new</code>	If FALSE (the default), add the (dis)aggregated data frame to existing ones. If TRUE, return only the (dis)aggregated data frame.
<code>na.rm</code>	If TRUE (the default), remove items calculated as NA.
<code>weights</code>	a data frame, a variable name as a character vector, or NULL. See details.
<code>forceAggregation</code>	binary. If TRUE, (dis)aggregation will be applied even though the items contained in the data and in the mapping do not fully match. The data is reduced to the items covered both by the mapping and the data.
<code>autodetect</code>	this parameter takes the values 'auto', 'aggregate' or 'disaggregate'. If 'auto' (the default) the function tries to auto-detect whether this is an aggregation or a disaggregation. If 'aggregate', it will aggregate, if 'disaggregate', it will disaggregate.
<code>scaleWeights</code>	logical. If TRUE, weights are scaled so that the sum of the components equals the value of the larger category.
<code>variable</code>	Column name of variables. Defaults to "variable".
<code>value</code>	Column name of values. Defaults to "value".
<code>unit</code>	Column name of units. Defaults to "unit".
<code>weight_val_col</code>	name of the value column in the "weights" data frame, if "weights" is a data frame
<code>weight_item_col</code>	name of the item column in the "weights" data frame, if "weights" is a data frame. The item column is the column corresponding to the mapping
<code>fun</code>	aggregation function to use. Defaults to sum.



## Details

By default "weights" is set to NULL. For aggregations, this means that values will be summed as they are. For disaggregations, each component of the larger category will take the same value as the larger category, or "scaleWeights" is TRUE, each component will be given an even weight. For aggregations, "weights" can also be the name of a variable contained in "data". "weights" may also be a data frame.

## Value

A data frame.

## Author(s)

Antoine Levesque

## Examples

```
library(tidyr)
library(dplyr)
data <- inline.data.frame(c(
  "model; scenario; region; variable; unit; period; value",
  "REMIND; Baseline; USA; GDP per Capita|MER; US$2005/yr; 2010; 40000",
  "REMIND; Baseline; USA; GDP per Capita|MER; US$2005/yr; 2020; 50000",
  "REMIND; Baseline; USA; Population; million; 2010; 300",
  "REMIND; Baseline; USA; Population; million; 2020; 350",
  "REMIND; Baseline; CHN; GDP per Capita|MER; US$2005/yr; 2010; 7000",
  "REMIND; Baseline; CHN; GDP per Capita|MER; US$2005/yr; 2020; 8000",
  "REMIND; Baseline; CHN; Population; million; 2010; 1300",
  "REMIND; Baseline; CHN; Population; million; 2020; 1400"))

mapping = inline.data.frame(c(
  "region; New_region",
  "USA; GLO",
  "CHN; GLO"
))

mapping2 = inline.data.frame(c(
  "Item ; Item_new",
  "Population; Urban Population ",
  "Population; Rural Population"
))

weights = inline.data.frame(c(
  "region; itemI ; weight",
  "USA ; Urban Population; 0.5",
  "USA ; Rural Population; 0.2",
  "CHN ; Urban Population; 2",
  "CHN ; Rural Population; 1"
))

#Regional Aggregation
```

```

aggregate_map(data,mapping, by = "region", subset2agg = c("Population"))

#Regional Weighted Aggregation
aggregate_map(data,mapping, by = "region", subset2agg = "GDP per Capita|MER",
              weights = "Population")

#Variable Weighted Disaggregation
aggregate_map(data,mapping2, by = c("variable" = "Item"),
              subset2agg = c("Population"),weights = weights,
              weight_val_col = "weight", weight_item_col = "itemI")

```

---

bin_distribute	<i>Distribute into Equal Bins</i>
----------------	-----------------------------------

---

### Description

bin\_distribute(x, binsize) distributes the items in x into the minimum number of bins whose sizes differ at maximum by one and do not exceed binsize. bin\_distribute\_sizes(count, binsize) calculates the sizes of such bins for count items.

### Usage

```

bin_distribute(x, binsize)

bin_distribute_sizes(count, binsize)

```

### Arguments

x	A character vector to be distributed into bins.
binsize	The maximum bin size.
count	The number of items to be binned.

### Value

bin\_distribute() returns a list with the sub-vectors of x. bin\_distribute\_sizes() returns a vector of sizes.

### Author(s)

Michaja Pehl

### Examples

```
regions <- c('CAZ', 'CHA', 'EUR', 'IND', 'JPN', 'LAM', 'MEA', 'NEU', 'OAS',  
            'REF', 'SSA', 'USA', 'World')
```

```
bin_distribute(regions, 5)  
bin_distribute_sizes(length(regions), 5)
```

```
bin_distribute(regions, 6)  
bin_distribute_sizes(length(regions), 6)
```

```
bin_distribute(regions, 7)  
bin_distribute_sizes(length(regions), 7)
```

---

calcAddVariable	<i>Calculate new variable</i>
-----------------	-------------------------------

---

### Description

Calculate new variable from existing ones, using a generic formula.

### Usage

```
calcAddVariable(data, formula, newUnit = "None", na.act = "no")
```

### Arguments

data	A data frame with columns "variable", "unit" and "value".
formula	An object of class formula, as returned by <code>stats::formula()</code> .
newUnit	Character vector with the unit for the newly calculated variable.
na.act	Indicates how NAs in the wide data frame should be handled. Default "no" indicates no action is taken.

### Details

**Obsolete.** This function will be removed in the near future. Use `calc_addVariable()` instead.

### Value

A data frame with the original and the new variables.

### Author(s)

Anselm Schultes, Michaja Pehl

### See Also

[calc\\_addVariable\(\)](#)

## Examples

---

calcCumulatedDiscount *Calculates the cumulated discounted time series*

---

## Description

Discount and cumulated a times series - gives the time series of the net present value (NPV). Baseyear for the NPV is the first period.

## Usage

```
calcCumulatedDiscount(
  data,
  nameVar = "Consumption",
  nameDisrate = "Interest Rate t/(t-1)|Real",
  discount = 0.05,
  fixYear = "none"
)
```

## Arguments

data	a quitte object containing consumption values - consumption has to be named "Consumption"
nameVar	name of the variable to be cumulated (and discounted)
nameDisrate	Name of the variable containing the discount rate
discount	The discount rate: either a numeric value, or 'BAU' to choose the discount rate supplied in nameDisrate
fixYear	From the discounted time series, subtract the value in year fixYear, if fixYear is not 'none'

## Value

cumulated discounted values for each scenario, model, region (quitte object)

## Author(s)

Anselm Schultes

## Examples

```
## Not run:
  erg <- calcCumulatedDiscount(data, disRate=0.03)

## End(Not run)
```

---

calcDecompEff	<i>Decomposes a change in a variable based on the changes of its factors (Kaya-like)</i>
---------------	--

---

### Description

Computes decomposition for a change in time or policy of a variable. The decomposition follows the methodology of the paper: "Some properties of an exact energy decomposition model", Sun and Ang, 2000, Energy

### Usage

```
calcDecompEff(df, x, bau = NULL, pol = NULL, gap = "policy")
```

```
calcDecompEff_scen(df, x, bau)
```

### Arguments

df	a quitte object (with coluns model, scenario, region, variable, unit, period, value)
x	a character vector detailing the explained variable from the decomposition as well as the factors in the decomposition. The explained variable should be named "explained" in the character vector.
bau	the name of the reference scenario, as a character string. It is NULL as a default
pol	the name of the policy scenario, as a character string. It is NULL as a default
gap	either "policy" or "time". If "policy", bau and pol should be detailed. If "time", bau and pol should stay NULL.

### Value

A data frame with the effects of each component of the decomposition. The data frame contains new columns:

- explained gives the name of the explained variable
- factors gives the name of the factor considered (from the decomposition chain)
- type gives the parameters:
  - eff is the result of the decomposition: how much of the change is to be attributed to the factor
  - value is the value of the factor
  - lag is the value of the factor in the reference scenario or in the previous period
  - delta is the difference in the factor's value between the policy and the reference, or between one period and another

### Author(s)

Antoine Levesque

**Examples**

```

# In this example, emissions = ue * fe_ue * emi_fe
testdf = inline.data.frame(
  c("scenario ;period; variable; value",

    "reference; 2015 ; emissions; 10",
    "reference; 2015 ; ue      ; 4",
    "reference; 2015 ; fe_ue   ; 2",
    "reference; 2015 ; emi_fe  ; 1.25",

    "reference; 2050 ; emissions; 50",
    "reference; 2050 ; ue      ; 25",
    "reference; 2050 ; fe_ue   ; 1.25",
    "reference; 2050 ; emi_fe  ; 1.6",

    "policy1; 2015 ; emissions; 10",
    "policy1; 2015 ; ue      ; 4",
    "policy1; 2015 ; fe_ue   ; 2",
    "policy1; 2015 ; emi_fe  ; 1.25",

    "policy1; 2050 ; emissions; 20",
    "policy1; 2050 ; ue      ; 25",
    "policy1; 2050 ; fe_ue   ; 1.25",
    "policy1; 2050 ; emi_fe  ; 0.64",

    "policy2; 2015 ; emissions; 10",
    "policy2; 2015 ; ue      ; 4",
    "policy2; 2015 ; fe_ue   ; 2",
    "policy2; 2015 ; emi_fe  ; 1.25",

    "policy2; 2050 ; emissions; 10",
    "policy2; 2050 ; ue      ; 25",
    "policy2; 2050 ; fe_ue   ; 1.25",
    "policy2; 2050 ; emi_fe  ; 0.32")
)
testdf = as.quitte(testdf)
decomposition_chain = c(explained = "emissions", "ue","fe_ue","emi_fe")

result = calcDecompEff(testdf,x = decomposition_chain,gap = "time")
result2 = calcDecompEff(testdf,x = decomposition_chain,bau = "reference",pol = "policy1")
result3 = calcDecompEff_scen(testdf, x = decomposition_chain, bau = "reference")

```

---

calcMitigationCost      *Calculate mitigation costs*

---

**Description**

Calculate mitigation costs

**Usage**

```
calcMitigationCost(  
  data,  
  scenBau,  
  scenPol,  
  yearFrom = 2010,  
  yearTo = 2100,  
  nameVar = "Consumption",  
  nameDisrate = "Interest Rate  $t/(t-1)|Real$ ",  
  discount = 0.05  
)
```

**Arguments**

data	quite object
scenBau	baseline scenario name
scenPol	policy scenario name
yearFrom	the startyear
yearTo	the endyear
nameVar	Name of the variable containing consumption. Defaults to "Consumption"
nameDisrate	Name of the variable for the discount rate, only needed if discount=endo. Defaults to "Interest Rate $t/(t-1) Real$ "
discount	discount rate - exogenous only for now

**Value**

regional mitigation costs (quite object)

**Author(s)**

Anselm Schultes

**Examples**

```
## Not run:  
calcMitigationCost(qd,"BAU","POL")  
  
## End(Not run)
```

---

calc\_addVariable      *Calculate new variables*

---

### Description

Calculate new variables from existing ones, using generic formulas.

### Usage

```
calc_addVariable(
  data,
  ...,
  units = NA,
  na.rm = TRUE,
  completeMissing = FALSE,
  only.new = FALSE,
  variable = variable,
  unit = unit,
  value = value,
  overwrite = TRUE,
  skip.missing.rhs = FALSE
)
```

```
calc_addVariable_(
  data,
  .dots,
  na.rm = TRUE,
  completeMissing = FALSE,
  only.new = FALSE,
  variable = "variable",
  unit = "unit",
  value = "value",
  overwrite = TRUE,
  skip.missing.rhs = FALSE
)
```

### Arguments

data	A data frame.
...	Formulas to calculate new variables. Either name-value pairs, the path to a .csv file, the content of a .csv file as a string, or a data frame. See details.
units	Character vector of units corresponding to new variables. Must be of length equal to ... or of length one (in which case all new variables receive the same unit).
na.rm	If TRUE (the default), remove items calculated as NA. This is generally the case for all calculations involving NA values, and all calculations involving missing variables. See completeMissing parameter.



completeMissing If TRUE, implicitly missing data, i.e. missing combinations of input data, are filled up with 0 before the calculation, and they are therefore not computed as NA (and potentially removed from the output). Make sure 0 is a sensible value for your calculations, else complete missing values manually. Defaults to FALSE.

only.new If FALSE (the default), add new variables to existing ones. If TRUE, return only new variables.

variable Column name of variables. Defaults to "variable".

unit Column name of units. Defaults to "unit". Ignored if no column with the same name is in data (e.g. data frames without unit column).

value Column name of values. Defaults to "value".

overwrite If TRUE (the default), values are overwritten if they already exist. If FALSE values are discarded and not overwritten if they already exist

skip.missing.rhs If FALSE (the default), fail if any right-hand-side variable is missing. If TRUE, warn, and skip that calculation. If "silent", skip without warning.

.dots Used to work around non-standard evaluation. See details.  
If ... is a list of name-value pairs, it has the general format

```
"lhs" = "rhs + calculations - formula", "`lhs 2`" = "lhs / `rhs 2`"
```

where lhs are the names of new variables to be calculated and rhs are the variables to calculate from. If lhs and rhs are no proper *identifiers*, they need to be quoted (see [Quotes](#) for details). When in doubt, just quote.  
If the new variables should have units, set units appropriately.  
.dots is a named list of strings denoting formulas and optionally units. The general format is

```
list("`lhs 1`" = "`rhs` / `calculation`",
      "`lhs 2`" = "sin(`rhs 2`)")
```

Units are optionally included with the formulas in a vector like

```
list("`lhs w/ unit`" = c("`rhs 1` + `rhs 2`", "rhs unit"))
```

Units do not require quoting.  
As an alternative, the variable, unit and formula can be specified as a .csv file in this format:

```
variable;          unit;          formula
Carbon Intensity|Cement; Mt CO2/Mt; `Emi|CO2|Cement` / `Production|Cement`
```

or as a single string containing the .csv content (joined by \n line breaks)

```
paste(c(
  "variable;          unit;          formula",
  "Consumption|pCap; US$2005/cap; 0.001 * `Consumption` / `Population`"),
  collapse = '\n')
```

or as a data frame with the same columns:

```
data.frame(
  variable = 'Consumption|pCap',
  unit     = 'US$2005/cap',
  formula  = '0.001 * `Consumption` / `Population`')
```

... and .dots are processed in order, and variables already calculated in the same call can be used for further calculations. Other existing columns, including period, can be referenced, but this is not supported and the results are considered *undefined*.

## Value

A data frame.

## Author(s)

Michaja Pehl

## Examples

```
data <- inline.data.frame(c(
  "model; scenario; region; variable; unit; period; value",
  "REMIND; Baseline; USA; GDP|MER; billion US$2005/yr; 2010; 12990",
  "REMIND; Baseline; USA; Population; million; 2010; 310.4",
  "REMIND; Baseline; USA; PE; EJ/yr; 2010; 91.62",
  "REMIND; Baseline; CHN; GDP|MER; billion US$2005/yr; 2020; 8882",
  "REMIND; Baseline; CHN; GDP|MER; billion US$2005/yr; 2010; 4119",
  "REMIND; Baseline; CHN; Population; million; 2020; 1387",
  "REMIND; Baseline; CHN; Population; million; 2010; 1349"))

calc_addVariable(data, "GDPpC" = "`GDP|MER` / Population * 1e3",
  "`ln GDPpC`" = "log(GDPpC)",
  units = c("US$2005/cap", NA))

calc_addVariable_(
  data,
  list("`GDPpC`" = c("`GDP|MER` / `Population` * 1e3", "US$/cap"),
    "`ln GDPpC`" = "log(`GDPpC`)")
)
```

---

calc\_growthrate

*Calculates the growth rate in '%/yr' for variables*

---

## Description

Calculates the growth rate in '%/yr' for variables

## Usage

```
calc_growthrate(x, only.new = FALSE, filter.function = identity)
```

**Arguments**

x	anything with an as.quitte method (data.frame, quitte or magclass object, mif file, ...)
only.new	If FALSE (the default), add new variables to existing ones. If variable names already exist in the data.frame, they are replaced. If TRUE, return only new variables.
filter.function	A function used to filter data before calculating growth rates. If instead a character vector is passed, only variables matching this vector are used.

**Details**

If, for example, your data contains the data in 2070 and 2060, the growth rate returned for 2070 is calculated as  $100 * ((d2070/d2060)^{(1/10)} - 1)$ . No growth rate can be calculated for the first year of the data. Infinite or undefined values (for example if  $d2060 = 0$ ) are dropped.

**Value**

data as a quitte object

**Author(s)**

Oliver Richters

**Examples**

```
## Not run:
GDPgrowth <- calc_growthrate(quitte_example_data, only.new = TRUE, filter.function = "GDP|PPP")
alldata <- calc_growthrate(quitte_example_data, only.new = FALSE,
  filter.function = function(x) filter(x, grepl("GDP", .data$variable)))

## End(Not run)
```

---

calc\_mode

*Calculate the mode of a sample*

---

**Description**

Calculate the mode of a sample

**Usage**

```
calc_mode(v)
```

**Arguments**

v                    A vector.

**Value**

The mode, or a vector of modes if the sample is multi-modal.

**Author(s)**

Michaja Pehl

**Examples**

```
calc_mode(c(1, 1, 100))
calc_mode(c('a', 'a', 'b', 'c', 'c'))
```

---

calc_quantiles	<i>Sample Quantiles</i>
----------------	-------------------------

---

**Description**

This is a wrapper function for [quantile](#) for easy use with data frames.

**Usage**

```
calc_quantiles(
  .data,
  value = NA,
  probs = c(q0 = 0, q25 = 0.25, q50 = 0.5, q75 = 0.75, q100 = 1),
  na.rm = TRUE,
  type = 7
)

calc_quantiles_(
  .data,
  value = "value",
  probs = c(q0 = 0, q25 = 0.25, q50 = 0.5, q75 = 0.75, q100 = 1),
  na.rm = TRUE,
  type = 7
)
```

**Arguments**

.data	a data frame, possibly grouped
value	column name for which sample quantiles should be calculated
probs	named numeric vector of probabilities with values in [0, 1].
na.rm	logical; if TRUE, any <b>NA</b> and <b>NaNs</b> are removed from data before the quantiles are computed.
type	an integer between 1 and 9 select one of the nine quantile algorithms detailed in <a href="#">quantile</a> to be used.

**Value**

A data frame.

**Author(s)**

Michaja Pehl

**Examples**

```
require(dplyr)
require(tidyr)

tibble(group = rep(c("A", "B"), 10),
        value = 1:20) %>%
  group_by(group) %>%
  calc_quantiles() %>%
  pivot_wider(names_from = 'quantile')
```

---

cartesian

*Generate Cartesian Product from Vectors*

---

**Description**

Generate Cartesian Product from Vectors

**Usage**

```
cartesian(..., sep = ".")
```

**Arguments**

...	Vectors to be combined
sep	If a character string, that string will separate the elements of ... (if necessary cast to character) in the output. If NULL, elements of ... will not be cast. Defaults to '.'.

**Value**

If sep is a character string, then a character vector of concatenated elements. If sep is NULL, then a list of concatenated elements.

**Examples**

```

cartesian(c('a', 'b'), 1:3, c('X', 'Y', 'Z'))
# [1] "a.1.X" "a.1.Y" "a.1.Z" "a.2.X" "a.2.Y" "a.2.Z" "a.3.X" "a.3.Y"
# [9] "a.3.Z" "b.1.X" "b.1.Y" "b.1.Z" "b.2.X" "b.2.Y" "b.2.Z" "b.3.X"
# [17] "b.3.Y" "b.3.Z"

str(cartesian(c('a', 'b'), 17:19, sep = NULL))
# List of 6
# $ :List of 2
# ..$ : chr "a"
# ..$ : int 17
# $ :List of 2
# ..$ : chr "a"
# ..$ : int 18
# $ :List of 2
# ..$ : chr "a"
# ..$ : int 19
# $ :List of 2
# ..$ : chr "b"
# ..$ : int 17
# $ :List of 2
# ..$ : chr "b"
# ..$ : int 18
# $ :List of 2
# ..$ : chr "b"
# ..$ : int 19

```

---

character.data.frame *Changes factor columns of a data frame into character columns*

---

**Description**

character.data.frame() turns factor columns of a data frame into character columns.

**Usage**

```
character.data.frame(df, ...)
```

```
character.data.frame_(df, .dots)
```

**Arguments**

df	a data frame
...	Column names to convert to characters.
.dots	Character vector of column names to turn into characters.

**Value**

A data frame.

**Author(s)**

Antoine Levesque

**See Also**[factor.data.frame\(\)](#)**Examples**

```
require(dplyr)
(df <- tibble(
  char = letters[1:3],
  fact = factor(LETTERS[24:26], levels = LETTERS[c(1:3, 24:26)]),
  num = (1:3) ^ 2))

character.data.frame(df)
character.data.frame_(df, 'num')
```

---

`check_quitte`*Check IAMC-style data frame for inconsistencies.*

---

**Description**

Check an IAMC-style data frame to see if values across variables and regions sum up to the totals specified within the data frame.

**Usage**

```
check_quitte(quitte, check_variables, check_regions = NULL)
```

**Arguments**

`quitte` IAMC-style data frame.  
`check_variables` List, string or file of variables to check.  
`check_regions` List, string or file of regions to check.

**Details**

Checking is performed for all variables and regions in `check_variables` and `check_regions`, which can be passed as a list of format

```
list("sum" = c("summand1", "summand2", ...))
```

a character string of format

```

sum1
summand1a
summand1b

sum2
summand2a
...

```

or as the path to a text file with this format.

If checking should be performed for variables or regions that are neither sum nor summand (e.g., the variable 'GDP' should be summed across regions, but is itself not a sum of other variables), include them as sum and their only summand in the respective list (i.e., `list("GDP" = "GDP")`) or as a double line in the character string or file.

If `check_regions` is `NULL`, variables are checked across all regions in `quitte`.

### Value

A data frame of all entries that did not match.

### Author(s)

Michaja Pehl

### Examples

```

require(dplyr, quietly = TRUE, warn.conflicts = FALSE)
quitte <- rbind(
  data.frame(
    model    = "REMIND",
    scenario = "Baseline",
    region   = c("World", "USA", "EUR"),
    variable = "GDP",
    unit     = "US$2005",
    period   = 2005,
    value    = c(3, 1, 1)
  ),
  data.frame(
    model    = "REMIND",
    scenario = "Baseline",
    region   = "ROW",
    variable = c("FE|Total", "FE|Solids", "FE|Electricity"),
    unit     = "EJ/a",
    period   = 2005,
    value    = c(3, 1, 1)
  )
)

check_variables <- list(
  "FE|Total" = c("FE|Solids", "FE|Electricity"),
  "GDP"      = "GDP"
)

```



```
check_regions <- paste0("World\nUSA\nEUR\n\nROW\nROW")

print(quitte)
print(check_variables)
cat(check_regions)

check_quitte(quitte, check_variables, check_regions)
```

---

chooseFilter                      *Allows to interactively filter data from quitte object*

---

## Description

Allows to interactively filter data from quitte object

## Usage

```
chooseFilter(
  data,
  types = c("model", "scenario", "region", "variable", "period"),
  keep = list()
)
```

## Arguments

data	A quitte object or something that can be transformed into one by as.quitte
types	vector of quitte columns for user to select data if more than one option available
keep	list with quitte columns as names and data points that should always be kept. If the column is not also in types, only the elements in that list are kept

## Author(s)

Oliver Richters

## Examples

```
## Not run:
  qe <- chooseFilter(quitte_example_dataAR6, types = c("model"),
                    keep = list(region = "World"))

## End(Not run)
```

---

df.2.named.vector      *Data frame to named vector*

---

### Description

Turns the two first columns of a data frame into a named vector, taking the values from the second and the names from the first column.

### Usage

```
df.2.named.vector(.data)
```

### Arguments

.data      A data frame with at least two columns.

### Value

A named vector.

### Author(s)

Michaja Pehl

### Examples

```
data <- data.frame(names = c("one", "two", "three"), values = 1:3)
data
df.2.named.vector(data)
```

---

df.as.list      *Data Frame as List*

---

### Description

Data Frame as List

### Usage

```
df.as.list(df, names = 1, x = 2)
```

### Arguments

df      A data frame.

names      Index used for naming list items. Integer or character, must work with df[[names]]. Defaults to the first data frame column.

x      Index used for list items. Integer or character, must work with df[[x]]. Defaults to the second data frame column.

**Value**

A list.

**Examples**

```
(df <- data.frame(
  modules = c('power', 'macro', 'welfare', 'PE_FE_parameters',
             'initialCap', 'aerosols'),
  `*` = c('IntC', 'singleSectorGr', 'utilitarian', 'iea2014', 'on',
         'exoGAINS'),
  check.names = FALSE
))

df.as.list(df, 'modules', '*')
```

---

df_populate_range	<i>Populate Data With Sequence Along Range</i>
-------------------	--

---

**Description**

Generate sequences of n equidistant data points for a column in a data frame.

**Usage**

```
df_populate_range(df, column, n = 100)
```

**Arguments**

df	A data frame.
column	<a href="#">&lt;tidy-select&gt;</a> The column with data to populate.
n	Length of the sequence to return. Defaults to 100.

**Value**

A data frame.

**Examples**

```
require(dplyr, warn.conflicts = FALSE, quietly = TRUE)
require(tidyr, warn.conflicts = FALSE, quietly = TRUE)

tibble(A = (1:3) ^ 2,
       B = exp(0:2)) %>%
  pivot_longer(everything()) %>%
  arrange(name, value) %>%
  print() %>%
  df_populate_range(value, n = 6)
```

---

df_variation	<i>Data Frame Variation</i>
--------------	-----------------------------

---

**Description**

Removes all columns from a data frame which have only identical data, to facilitate a quick overview.

**Usage**

```
df_variation(x)
```

**Arguments**

x                    A data frame.

**Value**

A data frame.

**Examples**

```
(quitte_example_data['Consumption' == quitte_example_data$variable,] -> x)
```

```
df_variation(x)
```

---

duplicate	<i>Duplicate rows</i>
-----------	-----------------------

---

**Description**

Duplicate rows in a data frame, modifying a specified column.

**Usage**

```
duplicate(data, ...)
```

```
duplicate_(data, column)
```

**Arguments**

data                A data frame or quitte object.

..., column        A key-value pair of the column to modify.

**Value**

A data frame or quitte object, same as input.

## Examples

```
require(dplyr)
(data <- tibble(region = rep(c('AFR', 'CHN'), 2),
                variable = paste('Var', c(1, 1, 2, 2)),
                value = 1:4))

data %>% duplicate(region = 'World')
```

---

factor.data.frame      *(Re-) Factorise factor and character columns in data frame*

---

## Description

factor.data.frame() turns character columns in a data frame into factor columns and refactorises factor columns, silently dropping unused levels.

## Usage

```
factor.data.frame(df, ...)
```

```
factor.data.frame_(df, .dots)
```

## Arguments

df	A data frame.
...	Column names to factorise.
.dots	Character vector of column names to factorise.

## Value

A data frame.

## Author(s)

Michaja Pehl

## See Also

[character.data.frame\(\)](#)

**Examples**

```
require(dplyr)
(df <- tibble(
  char = letters[1:3],
  fact = factor(LETTERS[24:26], levels = LETTERS[c(1:3, 24:26)]),
  num = (1:3) ^ 2))

str(factor.data.frame(df))
str(factor.data.frame_(df, 'num'))
```

---

factorise

*Factorise*

---

**Description**

Generate a factor with levels in prescribed order.

**Usage**

```
factorise(x)
```

**Arguments**

x                    A character vector.

**Value**

A factor from x, with levels in the same order as they appear in within x.

**Author(s)**

Michaja Pehl

**Examples**

```
factor(c('a', 'c', 'b'))
factorise(c('a', 'c', 'b'))
```

---

getColValues	<i>Retrieves values from one column of a quitte object</i>
--------------	--

---

**Description**

Retrieves values from one column of a quitte object

**Usage**

```
getColValues(df, colVar)
```

**Arguments**

df	quitte object
colVar	name of the column of interest

**Value**

a vector containing the values of the column without duplicates. The class of the returned vector is either numeric or character

**Author(s)**

Antoine Levesque

**Examples**

```
data(mtcars)  
getColValues(mtcars, "mpg")
```

---

getModel	<i>Retrieves models from a quitte object</i>
----------	--

---

**Description**

Retrieves models from a quitte object

**Usage**

```
getModel(df)
```

**Arguments**

df	quitte object
----	---------------

**Value**

a character vector containing the values of the column without duplicates

**Author(s)**

Antoine Levesque

**Examples**

```
## Not run:  
getPeriods(dataframe)  
  
## End(Not run)
```

---

getPeriods

*Retrieves periods from a quitte object*

---

**Description**

Retrieves periods from a quitte object

**Usage**

```
getPeriods(df)
```

**Arguments**

df                    quitte object

**Value**

a numeric vector containing the values of the column without duplicates

**Author(s)**

Antoine Levesque

**Examples**

```
## Not run:  
getPeriods(dataframe)  
  
## End(Not run)
```



---

getRegs	<i>Retrieves regions from a quitte object</i>
---------	---

---

**Description**

Retrieves regions from a quitte object

**Usage**

```
getRegs(df)
```

**Arguments**

df                    quitte object

**Value**

a character vector containing the values of the column without duplicates

**Author(s)**

Antoine Levesque

**Examples**

```
## Not run:  
getRegs(dataframe)  
  
## End(Not run)
```

---

getScenarios	<i>Retrieves scenarios from a quitte object</i>
--------------	---

---

**Description**

Retrieves scenarios from a quitte object

**Usage**

```
getScenarios(df)
```

**Arguments**

df                    quitte object

**Value**

a character vector containing the values of the column without duplicates

**Author(s)**

Antoine Levesque

**Examples**

```
## Not run:  
  getPeriods(dataframe)  
  
## End(Not run)
```

---

getVar

*Retrieves variable names from a quitte object*

---

**Description**

Retrieves variable names from a quitte object

**Usage**

```
getVar(df)
```

**Arguments**

df                    quitte object

**Value**

a character vector containing the values of the column without duplicates

**Author(s)**

Antoine Levesque

**Examples**

```
## Not run:  
  getVars(dataframe)  
  
## End(Not run)
```

---

`ggplot_bar_stacked_dodged`*Stacked and dodged bar plots*

---

## Description

Utility functions for plotting stacked (on top of each other) *and* dodged (next to each other) bars in the same figure.

## Usage

```
ggplot_bar_stacked_dodged(data, mapping, gap = 1)
```

```
add_stacked_dodged_xpos(data, ..., gap = 1)
```

```
calc_stacked_dodged_xlabels(data, ..., gap = 1)
```

## Arguments

<code>data</code>	A data frame
<code>mapping</code>	An aesthetic mapping generated by <a href="#">ggplot2:aes</a> , containing the aesthetics <code>x</code> , <code>y</code> , <code>fill</code> , and <code>dodge</code> . The aesthetic <code>x</code> will form groups on the x-axis, while <code>dodge</code> will form individual bars within the groups.
<code>gap</code>	The width of the gap between bars, relative to the width of the bars themselves (default: 1).
<code>...</code>	A selection of two columns. Both will be combined to form x-axis coordinates. The first will form the outer iteration (groups), the second the inner iteration (bars within a group). If unnamed, the column with calculated positions will be called <code>xpos</code> .

## Details

`add_stacked_dodged_xpos()` adds x-axis positions to a data frame for plotting two categorical variables within a bar plot. `calc_stacked_dodged_xlabels()` calculates matching label positions on the x-axis. `ggplot_bar_stacked_dodged()` uses both functions to generate a plot.

## Value

`add_stacked_dodged_xpos()` returns the input data frame with an additional column. Row and column order are preserved. `calc_stacked_dodged_xlabels()` returns a named character vector for use with [ggplot2:scale\\_x\\_continuous](#). `ggplot_bar_stacked_dodged()` returns a [ggplot2:ggplot](#) object.

## Author(s)

Michaja Pehl

**Examples**

```
require(tidyverse)

set.seed(0)
(data <- crossing(a = factorise(c('left', 'center', 'right')),
  b = factorise(c('top', 'middle', 'bottom')),
  c = letters[1:4],
  d = LETTERS[25:26]) %>%
  mutate(value = abs(rnorm(n())) + 0.2))

(plot.data <- add_stacked_dodged_xpos(data, c('c', 'a')))

(xlabels <- calc_stacked_dodged_xlabels(data, c('c', 'a')))

ggplot(data = plot.data) +
  scale_x_continuous(breaks = xlabels) +
  facet_wrap(~ d, ncol = 1, scales = 'free_x')

ggplot_bar_stacked_dodged(data, aes(x = a, y = value, fill = b, dodge = c),
  gap = 1/3) +
  facet_wrap(~ d, ncol = 1, scales = 'free_x')
```

---

`gg_colour_hue`*Get n Colours*

---

**Description**

Get n colours, evenly spaced along the colour wheel. Just like the ones `ggplot2::scale_colour_hue()` is using.

**Usage**

```
gg_colour_hue(n)
```

**Arguments**

`n` Either the number of colours to generate, or a character vector which will be used as names for the returned colours.

**Value**

A vector of character strings which can be used as color specifications by R graphics functions. The vector is named if `n` is a character vector.

**Examples**

```
gg_colour_hue(5)
gg_colour_hue(letters[1:3])
```

---

inline.data.frame	<i>Convert a vector of stings to a data frame</i>
-------------------	---

---

**Description**

inline.data.frame() converts a vector of strings that contain separated items into a data frame.

**Usage**

```
inline.data.frame(..., sep = ";", quote = "")
```

**Arguments**

...	string, or a vector of strings
sep	Item separator within strings, defaults to ";"
quote	Quote character for masking separators, empty by default

**Value**

a data frame

**Author(s)**

Michaja Pehl

**Examples**

```
inline.data.frame(
  "letters; numbers",
  "A; 1",
  "B; 2",
  NULL) # this last line allows for easy switching of line order
inline.data.frame(c("letters; numbers", "A; 1", "B; 2"))
```

---

interpolate_missing_periods
-----------------------------

---

*Interpolate missing periods*

---

**Description**

Adds missing periods to data frame and interpolates missing values linearly or using splines from adjacent existing ones. Values for periods smaller/bigger than the existing ones can be filled with the values for the first/last available period in the case of linear interpolation.

**Usage**

```
interpolate_missing_periods(
  data,
  ...,
  value = "value",
  expand.values = FALSE,
  method = "linear",
  combinations = "nesting"
)

interpolate_missing_periods_(
  data,
  periods,
  value = "value",
  expand.values = FALSE,
  method = "linear",
  combinations = "nesting"
)
```

**Arguments**

<code>data</code>	A data frame or a quitty object.
<code>...</code>	A name-value pair of periods to fill. If unnamed, defaults to 'period'. If empty (but possibly named) uses only periods present in data.
<code>value</code>	Name of the column to fill, defaults to 'value'.
<code>expand.values</code>	If FALSE (the default), values are not expanded beyond the range of available data. If TRUE values at the closest extreme is used for linear interpolation. Results for spline interpolation are possibly nonsensical.
<code>method</code>	Specifies the interpolation method. Either 'linear' for linear interpolation or 'spline', 'spline_fmm', or 'spline_natural' for spline interpolation. 'spline' is an alias for 'spline_fmm'. See <a href="#">spline()</a> for details.
<code>combinations</code>	Specifies the method with which other columns are treated. They are either preserved as-is ('nesting', the default), or are expanded to all unique combinations ('crossing'). See <a href="#">tidyr::expand()</a> for details.
<code>periods</code>	A named list of periods to fill.

**Value**

A data frame or a quitty object, the same as `data`.

**Author(s)**

Michaja Pehl

**Examples**

```

require(dplyr)

# generate some test data with explicit (A-y-2025) and implicit (B-x-2030)
# missing values
(data <- tibble(
  group = rep(c('A', 'B'), c(8, 4)),
  item  = c(rep('x', 4), rep('y', 4), rep('x', 4)),
  period = rep(c(2015, 2025, 2030, 2035), 3),
  value  = c(2, 4, 5, 6, 20, NA, 50, 60, NA, 400, 500, NA)))

# fill values for already existing periods
interpolate_missing_periods(data)

# fill values for existing periods, with full combinations of other columns
interpolate_missing_periods(data, combinations = 'crossing')

# add additional periods and fill values
interpolate_missing_periods(data, period = seq(2010, 2035, 5))

# also fill values outside the original data range
interpolate_missing_periods(data, seq(2010, 2035, 5), expand.values = TRUE)

# works on data frames with different column names
(data <- data %>%
  rename(year = period, coeff = value))

interpolate_missing_periods(data, year, value = 'coeff')

# works on quitte objects too
(quitte <- data %>%
  rename(model = group, scenario = item, period = year, value = coeff) %>%
  mutate(variable = 'Var 1', unit = 'u1') %>%
  as.quitte())

interpolate_missing_periods(quitte, expand.values = TRUE)

# and works with POSIXct periods
(quitte <- quitte %>%
  mutate(period = ISOyear(period)))

interpolate_missing_periods(quitte, period = ISOyear(seq(2010, 2035, 5)))

# standard evaluation example
interpolate_missing_periods_(data, periods = list(year = seq(2010, 2035, 5)),
  value = 'coeff', expand.values = TRUE)

```

**Description**

Converts integer years (e.g. 2023) to [POSIXct](#) date/time values (e.g. 2023-07-02 12:00:00 GMT) corresponding to July 2, noon, which is the middle of the (non-leap) year. The function keeps a cache of already converted values, as the underlying function [ISOdate\(\)](#) is rather slow.

**Usage**

```
ISOyear(year)
```

**Arguments**

year                    Vector of years to convert to [POSIXct](#).

**Value**

A vector of [POSIXct](#) values.

**Author(s)**

Michaja Pehl

**Examples**

```
ISOyear(c(2005, 2010, 2100, 1900))
```

---

list\_to\_data\_frame      *Convert mapping list to data frame*

---

**Description**

Convert mapping list to data frame

**Usage**

```
list_to_data_frame(l, ...)

list_to_data_frame_(l, category = "category", item = "item")
```

**Arguments**

l                        A named list of character vectors.

...                     Unquoted names of category and item columns. Defaults to 'category' and 'item'.

category                Name of category column. Defaults to 'category'.

item                     Name of item column. Defaults to 'item'.



**Value**

A data frame.

**Author(s)**

Michaja Pehl

**Examples**

```
l <- list(Africa = c('Egypt', 'Tanzania'),
          Europe = c('Portugal', 'Ukraine', 'Denmark'))
list_to_data_frame(l, region, country)
```

---

madrat\_mule

*Madrat Mule*

---

**Description**

Convert *anything* to a [magpie](#) object and back to traffic data across madrat borders.

**Usage**

```
madrat_mule(x)
```

**Arguments**

x                    Anything.

**Value**

A [magpie](#) object containing x (in unusable form), or the original x if a [magpie](#) object was passed.

**Author(s)**

Michaja Pehl

**Examples**

```
str(x <- madrat_mule(quitte_example_data))
madrat_mule(x)
```

---

magclass\_to\_tibble      *Make a Tibble of a Magclass*

---

**Description**

Sensible magclass to tibble conversion.

**Usage**

```
magclass_to_tibble(m, colnames = NULL)
```

**Arguments**

`m`                      A [magpie](#) object.  
`colnames`              Column names for the returned tibble. Must match the number of columns.

**Value**

A [tibble](#).

**Examples**

```
magclass_to_tibble(magclass::maxample('pop'))
```

---

mif2rds                      *MIF2RDS converter*

---

**Description**

Converts mif file into more memory efficient rds file. Additionally, removes NA and duplicate entries.

**Usage**

```
mif2rds(input, output = "output.rds")
```

**Arguments**

`input`                      Path to the MIF file to be converted  
`output`                      File name of the output file

**Author(s)**

Jan Philipp Dietrich

---

mutate_text	<i>Uses mutate based on a character vector#'</i>
-------------	--

---

**Description**

Uses mutate based on a character vector#'

**Usage**

```
mutate_text(df, s)
```

**Arguments**

df                    a data frame  
s                     a character string containing the formula to be applied in mutate

**Value**

A data frame transformed with the mutate function

**Author(s)**

Antoine Levesque

**Examples**

```
df = data.frame(x = c(1,2),  
               y = c(3,4))  
form = "z = x + y"  
mutate_text(df, form)
```

---

order.levels	<i>Order data frame factor levels.</i>
--------------	--

---

**Description**

Arranges the levels of data frame columns in a given order. Non-factor columns are silently converted.

**Usage**

```
order.levels(df, ..., drop.extra.levels = TRUE)  
order.levels_(df, dots, drop.extra.levels = TRUE)
```

**Arguments**

df                    A data frame (or quite object).  
 ...                   Name-value pairs assigning level order to factor columns.  
 drop.extra.levels    If TRUE (default) levels not present in the factor are silently dropped.  
 dots                   A named list of factor columns and corresponding levels.

**Value**

A data frame (or quite object, same as data).

**Author(s)**

Michaja Pehl

**Examples**

```
require(dplyr)
str(df <- tibble(UPPER = LETTERS[3:1], lower = factor(letters[24:26]),
                value = 1:3))
str(order.levels(df, UPPER = LETTERS[1:3], lower = letters[26:20]))
str(order.levels_(df, list(UPPER = LETTERS[1:3], lower = letters[26:23]),
                      drop.extra.levels = FALSE))
```

---

overwrite

*Replace rows in a data frame with new values*

---

**Description**

overwrite() rbind()s the data frames lhs and rhs, removing any duplicate lines, which are determined without regard to the columns in except.

**Usage**

```
overwrite(lhs, rhs, except = "value")
```

**Arguments**

lhs                    data frame with values that will replace others  
 rhs                    data frame with values that will be replaced  
 except                names of columns that will not be considered in determining which columns to replace; defaults to "value"

**Value**

data frame in which rows from rhs have been replaced with rows from lhs

**Author(s)**

Michaja Pehl

**Examples**

```
require(dplyr)
data <- data.frame(expand.grid(UPPER = LETTERS[1:2],
                             lower = letters[24:26]),
                  value = 1:6)
data
data %>%
  filter(lower == "y") %>%
  mutate(value = value * 10) %>%
  overwrite(data)
```

---

prepQuitteForScatter *selects two variables from a long format Quitte and puts them into a wide format Quitte*

---

**Description**

QuitteIn contains two variables contained in varNames that should be plotted in a scatter plot. The functions forms the new QuitteOut with the variables x and y. QuitteOut can be used in ggplot with plotting x and y. The unit needs to be replaced by None.

**Usage**

```
prepQuitteForScatter(quitteIn, varNames)
```

**Arguments**

quitteIn	Quitte with original data
varNames	Vector with two variable names that must be contained in QuitteIn\$variable

**Value**

quitte object

**Author(s)**

Nico Bauer, Anselm Schultes, Jerome Hilaire

**Examples**

```
## Not run:
  quitteOut <- prepQuitteForScatter(quitteIn, c('Emissions|CO2', 'Price|Carbon'))

## End(Not run)
```

---

quitte-class	<i>Class "quitte" ~~~</i>
--------------	---------------------------

---

**Description**

The quitte class is a more standardized data.frame format. `is.quitte` tests if `x` is an quitte-object, `as.quitte` transforms `x` to an quitte-object (if possible).

**Usage**

```
as.quitte(x, periodClass = "integer", addNA = FALSE, na.rm = FALSE)

is.quitte(x, warn = TRUE)
```

**Arguments**

<code>x</code>	An object that should be either tested or transformed as/to an quitte-object.
<code>periodClass</code>	integer or POSIXct
<code>addNA</code>	modifies a factor by turning NA into an extra level (so that NA values are counted in tables, for instance).
<code>na.rm</code>	if set to TRUE entries with value NA will be removed
<code>warn</code>	display warnings or not

**Functions**

- `is.quitte()`: quitte check

**Author(s)**

Jan Philipp Dietrich

---

quitte2quantiles	<i>calculate quantiles</i>
------------------	----------------------------

---

**Description**

calculate quantiles

**Usage**

```
quitte2quantiles(
  x,
  probs = c(q0 = 0, q25 = 0.25, q50 = 0.5, q75 = 0.75, q100 = 1),
  grouping = c("region", "variable", "period", "scenario")
)
```

**Arguments**

x                    dataframe to add quantiles  
 probs                default=c(q0=0,q25=0.25,q50=0.5,q75=0.75,q100=1)  
 grouping            default=c("region", "variable", "period", "scenario")

**Author(s)**

Gunnar Luderer, Lavinia Baumstark

**Examples**

```
## Not run:
p <- x.minmax = quitte2quantiles(x,probs=c("min"=0,"max"=1))

## End(Not run)
```

---

quitteSort                    *Sorts a quitte object in a standardized way: model -> scenario -> variable -> unit -> region -> period*

---

**Description**

Sorts a quitte object in a standardized way: model -> scenario -> variable -> unit -> region -> period

**Usage**

quitteSort(x)

**Arguments**

x                    anything with an as.quitte method (data.frame, quitte or magclass object, mif file, ...)

**Value**

the sorted quitte object

**Author(s)**

Oliver Richters

---

quite\_example\_data    *quite example data*

---

**Description**

A quite object with some example data from the REMIND model.

**Usage**

quite\_example\_data

**Author(s)**

Michaja Pehl

---

quite\_example\_dataAR6  
*quite example data with three models (REMIND, GCAM, MES-SAGEix) and two scenarios (Current Policies, Delayed transition)*

---

**Description**

A quite object with some example data from NGFS Phase 3.

**Usage**

quite\_example\_dataAR6

**Author(s)**

Oliver Richters



---

read.gdx	<i>Read item from .gdx file as quitte data frame</i>
----------	--

---

### Description

`read.gdx()` is a wrapper function for either `gdxrrw::rgdx()` or `gamstransfer::readGDX()` that returns a quitte data frame.

### Usage

```
read.gdx(  
 .gdxName,  
  requestList.name,  
  fields = "1",  
  colNames = NULL,  
  factors = deprecated(),  
  squeeze = TRUE  
)
```

### Arguments

<code>.gdxName</code>	Path to a .gdx file.
<code>requestList.name</code>	Name of the item to read.
<code>fields</code>	Fields to read from variables and equations. When using <code>gdxrrw</code> , any of l, m, lo, up, s. When using <code>gamstransfer</code> , level, marginal, lower, upper, and scale are understood as well. all will return all fields. Ignored when reading sets or parameters.
<code>colNames</code>	String vector of column names to override dimension and field names.
<code>factors</code>	Deprecated. Do not use any more.
<code>squeeze</code>	If TRUE, squeeze out any zero or EPS stored in the GDX container. Ignored when using <code>gamstransfer</code> .

### Details

`read.gdx()` will use `gdxrrw::rgdx()` if `gdxrrw` is installed and the option `quitte_force_gamstransfer` is not TRUE, otherwise it will use `gamstransfer::readGDX()`.

### Value

A quitte data frame.

### Author(s)

Michaja Pehl

---

read.quitte	<i>Read IAMC-style .csv or .xlsx files, or object from rds file.</i>
-------------	--

---

### Description

Reads IAMC-style .csv or .xlsx files or object from rds file into a quitte data frame.

### Usage

```
read.quitte(
  file,
  sep = NULL,
  quote = "",
  na.strings = c("UNDF", "NA", "N/A", "n_a"),
  convert.periods = FALSE,
  check.duplicates = TRUE,
  factors = TRUE,
  drop.na = FALSE,
  comment = "#",
  filter.function = identity,
  chunk_size = 200000L
)
```

### Arguments

file	Path of IAMC-style .csv, .xlsx, or rds file or vector of paths to read.
sep	Column separator, defaults to ";" in read_mif_header().
quote	Quote characters, empty by default.
na.strings	Entries to interpret as NA; defaults to c("UNDF", "NA", "N/A", "n_a")
convert.periods	If TRUE, periods are converted to <a href="#">POSIXct</a> . If FALSE (the default), periods are numerical.
check.duplicates	If TRUE a duplicates check will be performed on the data. For time- and memory-critical applications this can be switched off.
factors	Return columns as factors (TRUE, the default) or not.
drop.na	Should NA values be dropped from the quitte?
comment	A character which at line start signifies the optional comment header with metadata at the head of file. The comment header, if present, is returned as a comment_header attribute. If multiple files are read, the comment_header attribute is a list of comment headers with file paths as names.
filter.function	A function used to filter data during read. See Details.
chunk_size	Number of lines to read at a time. Defaults to 200000. (REMINDED .mif files have between 55000 and 105000 lines for H12 and EU21 regional settings, respectively.)

## Details

In order to process large data sets, like IIASA data base snapshots, `read.quitte()` reads provided files in chunks of `chunk_size` lines (not for Excel files), and applies `filter.function()` to the chunks. This allows for filtering data piece-by-piece, without exceeding available memory. `filter.function` is a function taking one argument, a `quitte` data frame of the read chunk, and is expected to return a data frame. Usually it should simply contain all the filters usually applied after all the data is read in. Suppose there is a file `big_IIASA_snapshot.csv`, from which only data for the REMIND and MESSAGE models between the years 2020 to 2050 is of interest. Normally, this data would be processed as

```
read.quitte(file = 'big_IIASA_snapshot.csv') %>%
  filter(grepl('^(REMIND|MESSAGE)', .data$model),
         between(.data$period, 2020, 2060))
```

If however `big_IIASA_snapshot.csv` is too large to be read in completely, it can be read using

```
read.quitte(file = 'big_IIASA_snapshot.csv',
            filter.function = function(x) {
  x %>%
    filter(grepl('^(REMIND|MESSAGE)', .data$model),
           between(.data$period, 2020, 2060))
})
```

## Value

A `quitte` data frame.

## Author(s)

Michaja Pehl

## Examples

```
## Not run:
read.quitte(c("some/data/file.mif", "some/other/data/file.mif"))
read.quitte("some/data/file.csv", sep = ",", quote = "'")

## End(Not run)
```

---

read.snapshot

*Reads IAMC-style .csv or .xlsx files obtained as a IIASA snapshot into a quitte data frame, or data from rds file, allowing to filter the loaded data. If head, tail and grep are on your system, a pre-filtering improves performance for csv files.*

---

**Description**

Reads IAMC-style .csv or .xlsx files obtained as a IIASA snapshot into a `quitte` data frame, or data from rds file, allowing to filter the loaded data. If `head`, `tail` and `grep` are on your system, a pre-filtering improves performance for csv files.

**Usage**

```
read.snapshot(file, keep = list(), filter.function = identity)
```

**Arguments**

<code>file</code>	Path of single IAMC-style .csv/.mif/.xlsx/.rds file
<code>keep</code>	list with <code>quitte</code> columns as names and data points that should be kept. If <code>head</code> , <code>tail</code> and <code>grep</code> are available and a csv/mif file is read, this list is used to extract the data before reading it into R. The more you restrict the data here, the faster the data is read.
<code>filter.function</code>	A function used to filter data during read, see <code>read.quitte</code> description. This allows for more complex filtering, but no performance-enhancing pre-filtering using <code>grep</code> is used. The 'keep' list and the 'filter.function' can be combined.

**Value**

A `quitte` data frame.

**Author(s)**

Oliver Richters

**Examples**

```
## Not run:
read.filter.snapshot("snapshot.csv", list(scenario = c("CurPol", "NDC"), region = "World"))
## End(Not run)
```

---

read_mif_header	<i>Read .mif Header</i>
-----------------	-------------------------

---

**Description**

Read .mif Header

**Usage**

```
read_mif_header(file, sep = ";", comment = "#")
```

**Arguments**

file	A path to a .mif file.
sep	Column separator, defaults to ";".
comment	A character which at line start signifies the optional comment header with meta-data at the head of file, defaults to "#".

**Value**

A list with elements header, comment\_header, and useless.last.column.

**Author(s)**

Michaja Pehl

---

remind_timesteps	<i>REMINd time steps</i>
------------------	--------------------------

---

**Description**

A data frame containing the weights with which years contribute to specific periods.

**Usage**

```
remind_timesteps
```

**Author(s)**

Michaja Pehl

**Examples**

```
require(tidyverse)

remind_timesteps %>%
  filter(period %in% c(2055, 2060, 2070)) %>%
  spread(period, weight, fill = 0) %>%
  print(n = Inf)
```

---

removeColNa	<i>Removes all NA columns of a data frame</i>
-------------	---

---

**Description**

removeColNa() Removes all columns of a data frame for which all entries are NA, or the default of fct\_explicit\_na

**Usage**

```
removeColNa(df)
```

**Arguments**

df                    a data frame

**Value**

a data frame

**Author(s)**

Antoine Levesque

**Examples**

```
df <- data.frame(  
  character = letters[1:5],  
  factor = as.factor(LETTERS[1:5]),  
  value = 1:5,  
  unit = NA,  
  unit2 = forcats::fct_na_value_to_level(factor(NA), level = '(Missing)'),  
  stringsAsFactors = FALSE)  
str(df)  
str(removeColNa(df))
```

---

replace_column	<i>Replace data frame column</i>
----------------	----------------------------------

---

**Description**

Replaces the column of a data frame with that from a mask data frame.

**Usage**

```
replace_column(
  data,
  mask,
  ...,
  drop.extra = FALSE,
  ignore.ambiguous.match = FALSE
)
```

```
replace_column_(
  data,
  mask,
  old_column,
  match_column,
  new_column,
  drop.extra = FALSE,
  ignore.ambiguous.match = FALSE
)
```

**Arguments**

<code>data</code>	A data frame or a quitted object.
<code>mask</code>	A data frame containing the <code>match_column</code> and the <code>new_column</code> .
<code>...</code>	Definition of <i>old</i> , <i>match</i> , and <i>new</i> columns, see details.
<code>drop.extra</code>	Drop rows not present in <i>match</i> column?
<code>ignore.ambiguous.match</code>	<code>replace_column()</code> will issue a warning if the <i>match</i> column in <i>mask</i> does not map unambiguously to <i>data</i> , unless it is suppressed (TRUE). Using ambiguous matches can be desired for duplicating specific rows.
<code>old_column</code>	<i>old</i> column name, see details.
<code>match_column</code>	<i>match</i> column name, see details.
<code>new_column</code>	<i>new</i> column name, see details.

**Details**

Replaces the *old* column in data frame *data* by the *new* column from data frame *mask* based on the matching between *old* (*data*) and *match* (*mask*) columns.

This can be used to replace columns based on a mapping to, e.g., rename scenarios, regions, etc. in model data.

**Value**

A data frame or a quitted object, same as *data*.

**Author(s)**

Michaja Pehl

**Examples**

```

# simple example with matching old and match column names
(model_data <- data.frame(
  model = c('Model1', '2ndModel', 'Model Three'),
  region = c('Region 1', 'Region 2', 'Region 1'),
  value = 1:3))

(mask <- data.frame(
  model = c('Model1', '2ndModel', 'Model Three', 'fourth Model'),
  clear_name = paste('Model', 1:4)))

replace_column(model_data, mask, model, clear_name)

# mismatched column names
(model_data <- data.frame(
  model = c('Model1', '2ndModel', 'Model Three', 'fourth Model'),
  region = c('Region 1', 'Region 2', 'Region 1', 'Region 2'),
  value = 1:4))

(mask <- data.frame(
  ugly_name = c('Model1', '2ndModel', 'Model Three'),
  clear_name = paste('Model', 1:3)))

replace_column(model_data, mask, model = ugly_name, clear_name)

# SE example
replace_column_(model_data, mask, 'model', 'ugly_name', 'clear_name')

# dropping the extra entries in model
replace_column(model_data, mask, model = ugly_name, clear_name,
  drop.extra = TRUE)

# also works on quitte objects
require(dplyr)
(quitte <- tibble(
  model = c('Model1', '2ndModel'),
  scenario = 'Scenario',
  region = 'Region',
  variable = 'Variable',
  unit = 'Unit',
  period = 2010,
  value = 1:2) %>%
  as.quitte())
replace_column(quitte, mask, model = ugly_name, clear_name)
str(.Last.value)

```



**Description**

Finds duplicates in a quitte object and warns about them per model and scenario. Also warn if variables are identical but value or units differ.

**Usage**

```
reportDuplicates(mifdata, action = "warn")
```

**Arguments**

mifdata	object that can be converted with as.quitte
action	if set to 'warn', a warning with duplicate variables is raised

**Value**

only the data that is duplicated. Has 0 rows if everything is fine

---

revalue.levels	<i>Revalue data frame factor levels.</i>
----------------	--

---

**Description**

Revalue the names of a level or character column in a dataframe, according to a named vector given as an input

**Usage**

```
revalue.levels(df, ...)
revalue.levels_(df, dots)
```

**Arguments**

df	A data frame (or quitte object).
...	Name-value pairs assigning a named vector with new names to a column from the dataframe.
dots	A named list of columns containing the named vector with the old and new names for each column

**Value**

A data frame (or quitte object, same as data).

**Author(s)**

Antoine Levesque

**Examples**

```

data <- inline.data.frame(c(
  "model; scenario; region; variable; unit; period; value",
  "REMIND; Baseline; USA; GDP per Capita|MER; US$2005/yr; 2010; 40000",
  "REMIND; Baseline; USA; Population; million; 2010; 300",
  "REMIND; Baseline; CHN; GDP per Capita|MER; US$2005/yr; 2010; 7000"))

reg_vec = c(USA = "United States")
var_vec = c("GDP per Capita|MER" = "gdp",
           Population = "pop")

revalue.levels(data, region = reg_vec)
revalue.levels_(data, list(region = reg_vec, variable = var_vec))

```

---

`seq_range`*Sequence from a Range*

---

**Description**

Generate regular sequence from a range. Wrapper function for `seq()`.

**Usage**

```
seq_range(range, by = NA, length.out = NULL)
```

**Arguments**

<code>range</code>	Vector with starting and end values of the sequence. Only first two elements are considered.
<code>by</code>	Number; increment of the sequence.
<code>length.out</code>	Desired length of the sequence. A non-negative number, which will be rounded up if fractional.

**Value**

Returns a vector of type "integer" or "double": programmers should not rely on which.

**Author(s)**

Michaja Pehl

**See Also**

[seq\(\)](#), [range\(\)](#)

**Examples**

```
seq_range(range(1:13), by = 3)
```

---

setXor	<i>Set Xor</i>
--------	----------------

---

**Description**

Performs  $(x \cup y) \setminus (x \cap y)$  on parameters, returning all elements that are in either x or y, but not both.

**Usage**

```
setXor(x, y)
```

**Arguments**

x, y                    Objects to perform set function on.

**Author(s)**

Michaja Pehl

**Examples**

```
x <- c('a', 'b', 'c')
y <- c('b', 'c', 'd')
setXor(x, y)
```

---

signrange	<i>Sign Range</i>
-----------	-------------------

---

**Description**

Returns the range of signs in a numerical vector as a character string.

**Usage**

```
signrange(x, na.rm = TRUE)
```

**Arguments**

x                        A numerical vector.  
na.rm                    Should NAs be ignored?

**Value**

A character string of signs found in x.

**Author(s)**

Michaja Pehl

**Examples**

```
signrange(-1)
signrange(0)
signrange(1)
signrange(c(-1, 0))
signrange(c(0, 1))
signrange(c(-1, 1))
```

---

strtrimcommon

*Trim common portions from both sides of a vector of strings*


---

**Description**

Trim common portions from both sides of a vector of strings

**Usage**

```
strtrimcommon(x, split = "", USE.NAMES = FALSE, return.all = FALSE)
```

**Arguments**

x	A vector of strings
split	A <a href="#">character</a> to use for splitting. If split is empty (i.e. split = ''), x is split into single characters. Otherwise, x is split on split boundaries.
USE.NAMES	logical; if TRUE use x as <a href="#">names</a> for the result.
return.all	logical; if FALSE (the default), returns only the striped strings. If TRUE, returns a list with elements left, strings, right, containing left and right common portions, and the trimmed strings, respectively.

**Value**

A (named) vector of strings, or a list of string vectors (see parameter return.all for details).

**Author(s)**

Michaja Pehl

**Examples**

```
x <- c('/tmp/remind2_test-convGDX2MIF_fulldata.gdx',
      '/tmp/remind2_test-Ariadne_fulldata.gdx',
      '/tmp/remind2_test-NAVIGATE_fulldata.gdx',
      '/tmp/remind2_test-NGFS_fulldata_oneRegi.gdx',
      '/tmp/remind2_test-SHAPE_fulldata.gdx')

strtrimcommon(x, USE.NAMES = TRUE)

x <- c('Some|name|with|common|text|elements',
      'Some|name|without|extra|text|elements')

strtrimcommon(x, split = '|', return.all = TRUE)
```

---

sum\_total

*Sum over one dimension of a data frame*


---

**Description**

sum\_total() is a short-hand function to calculate and insert the (weighted) sum of a extensive (intensive) category in a data frame.

**Usage**

```
sum_total(data, group, value = NA, name = "Total", na.rm = TRUE, weight = NA)

sum_total_(data, group, value = NA, name = "Total", na.rm = TRUE, weight = NA)
```

**Arguments**

data	a data frame
group	column for which the sum is to be calculated
value	column of the numbers to be summed
name	entry in column group for the sum; defaults to "Total"
na.rm	logical. Should missing values (including NaN) be removed (default)?
weight	column of the weights to be applied, if any

**Value**

a data frame

**Author(s)**

Michaja Pehl

**Examples**

```

require(dplyr)

(d <- expand.grid(
  UPPER = LETTERS[1:2],
  lower = letters[24:26],
  number = 1:2
) %>%
  arrange(UPPER, lower, number) %>%
  mutate(value = c(1:6, NA, 8:12)))

sum_total(d, UPPER)

sum_total(d, lower, name = 'sum over lower', na.rm = FALSE)

(e <- tibble(
  item = c('large', 'medium', 'small'),
  specific.value = c(1, 10, 100),
  size = c(1000, 100, 1)))

sum_total(e, item, value = specific.value, name = 'Average', weight = size)

```

---

```

toolExtractSortScaleQuitte
  prepare data for plots

```

---

**Description**

prepare data for plots

**Usage**

```

toolExtractSortScaleQuitte(
  x,
  scen,
  vars,
  var.scaling = 1,
  regi = c("World"),
  prd = getPeriods(x)
)

```

**Arguments**

x	dataframe to prepare
scen	scenario to select
vars	variables to select

var.scaling      scaling of the variables, default=1  
regi              region to select, default="World"  
prd               period to select, default=getPeriods(x)

**Author(s)**

Gunnar Luderer, Lavinia Baumstark

**Examples**

```
## Not run:  
p <- toolExtractSortScaleQuitte(x, scen=c("BAU"),  
                                vars=c("Emi|CO2", "FE|Industry"),  
                                regi=c("EUR", "LAM"), prd=c(2005, 2030, 2050))  
  
## End(Not run)
```

---

unique\_or\_levels      *Unique or levels*

---

**Description**

Abstract the differences between character vectors and factors.

**Usage**

```
unique_or_levels(x)
```

**Arguments**

x                      A character vector or a factor.

**Value**

A character vector with the unique elements of x if it is a character vector, or the levels of x if it is a factor.

**Author(s)**

Michaja Pehl

---

write.IAMCxlsx	<i>Write .xlsx file</i>
----------------	-------------------------

---

### Description

Write a .xlsx file in line with IAMC standard.

### Usage

```
write.IAMCxlsx(x, path, append = FALSE)
```

### Arguments

x	A <a href="#">quite</a> data frame.
path	Path or connection to write to.
append	Overwrite existing files (FALSE, default), or append to them (TRUE).

### Author(s)

Michaja Pehl, Oliver Richters

### Examples

```
write.IAMCxlsx(quite_example_data, tempfile())
```

---

write.mif	<i>Write .mif file</i>
-----------	------------------------

---

### Description

A wrapper around [readr::write\\_lines](#) for writing files conforming to the [.mif standard](#).

### Usage

```
write.mif(  
  x,  
  path,  
  comment_header = NULL,  
  comment = "#",  
  append = FALSE,  
  sep = ";",  
  na = "NA"  
)
```



```
write.IAMCcsv(  
  x,  
  path,  
  comment_header = NULL,  
  comment = "#",  
  append = FALSE,  
  sep = ",",  
  na = ""  
)
```

### Arguments

x	A <a href="#">quitte</a> data frame.
path	Path or connection to write to.
comment_header	Comment header to be written to the .mif file. Ignored if append is TRUE.
comment	A character to prefix comment header lines with. Must match existing comment characters in file path if append is TRUE.
append	Overwrite existing files (FALSE, default), or append to them (TRUE).
sep	Single character used to separate fields within a record. Defaults to ;.
na	String used for NA elements. Defaults to 'NA' for write.mif() and to '' for write.IAMCcsv().

### Details

write.IAMCcsv() uses commas as field separators instead of semi-colons.

### Author(s)

Michaja Pehl

### Examples

```
write.mif(quitte_example_data, tempfile())
```

# Index

## \* classes

- quitte-class, 46
  
- add\_countrycode, 4
- add\_countrycode\_ (add\_countrycode), 4
- add\_remind\_timesteps\_columns  
  (add\_timesteps\_columns), 5
- add\_stacked\_dodged\_xpos  
  (ggplot\_bar\_stacked\_dodged), 35
- add\_timesteps\_columns, 5
- aes(), 6
- aggregate\_map, 7
- as.quitte (quitte-class), 46
- as.quitte, character-method  
  (quitte-class), 46
- as.quitte-methods (quitte-class), 46
  
- bin\_distribute, 10
- bin\_distribute\_sizes (bin\_distribute),  
  10
  
- calc\_addVariable, 16
- calc\_addVariable(), 11
- calc\_addVariable\_ (calc\_addVariable), 16
- calc\_growthrate, 18
- calc\_mode, 19
- calc\_quantiles, 20
- calc\_quantiles\_ (calc\_quantiles), 20
- calc\_stacked\_dodged\_xlabels  
  (ggplot\_bar\_stacked\_dodged), 35
- calcAddVariable, 11
- calcCumulatedDiscount, 12
- calcDecompEff, 13
- calcDecompEff\_scen (calcDecompEff), 13
- calcMitigationCost, 14
- cartesian, 21
- character, 60
- character.data.frame, 22
- character.data.frame(), 29
  
- character.data.frame\_  
  (character.data.frame), 22
- check\_quitte, 23
- chooseFilter, 25
- countrycode::countrycode(), 4
  
- df.2.named.vector, 26
- df.as.list, 26
- df\_populate\_range, 27
- df\_variation, 28
- duplicate, 28
- duplicate\_ (duplicate), 28
  
- factor.data.frame, 29
- factor.data.frame(), 23
- factor.data.frame\_ (factor.data.frame),  
  29
- factorise, 30
  
- gamstransfer::readGDX(), 49
- gdxrrw::rgdx(), 49
- getColValues, 31
- getModels, 31
- getPeriods, 32
- getRegs, 33
- getScenarios, 33
- getVars, 34
- gg\_colour\_hue, 36
- ggplot(), 6
- ggplot2::scale\_colour\_hue(), 36
- ggplot2:aes, 35
- ggplot2:ggplot, 35
- ggplot2:scale\_x\_continuous, 35
- ggplot\_bar\_remind\_vts  
  (add\_timesteps\_columns), 5
- ggplot\_bar\_stacked\_dodged, 35
- ggplot\_bar\_vts (add\_timesteps\_columns),  
  5
  
- inline.data.frame, 37

interpolate\_missing\_periods, 37  
interpolate\_missing\_periods\_  
    (interpolate\_missing\_periods),  
    37  
is.quitte (quitte-class), 46  
ISOdate(), 40  
ISOyear, 39  
  
list\_to\_data\_frame, 40  
list\_to\_data\_frame\_  
    (list\_to\_data\_frame), 40  
  
madrat\_mule, 41  
magclass\_to\_tibble, 42  
magpie, 41, 42  
mif2rds, 42  
mutate\_text, 43  
  
NA, 20  
names, 60  
NaN, 20  
  
order.levels, 43  
order.levels\_ (order.levels), 43  
overwrite, 44  
  
POSIXct, 40, 50  
prepQuitteForScatter, 45  
  
quantile, 20  
quitte, 64, 65  
quitte (quitte-package), 3  
quitte-class, 46  
quitte-package, 3  
quitte2quantiles, 46  
quitte\_example\_data, 48  
quitte\_example\_dataAR6, 48  
quitteSort, 47  
Quotes, 17  
  
range(), 58  
read.gdx, 49  
read.quitte, 50  
read.snapshot, 51  
read\_mif\_header, 52  
readr::write\_lines, 64  
remind\_timesteps, 5, 6, 53  
removeColNa, 54  
replace\_column, 54  
replace\_column\_ (replace\_column), 54  
  
reportDuplicates, 56  
revalue.levels, 57  
revalue.levels\_ (revalue.levels), 57  
  
seq(), 58  
seq\_range, 58  
setXor, 59  
signrange, 59  
spline(), 38  
stats::formula(), 11  
strtrimcommon, 60  
sum\_total, 61  
sum\_total\_ (sum\_total), 61  
  
tibble, 42  
tidy-select, 27  
tidyr::expand(), 38  
toolExtractSortScaleQuitte, 62  
  
unique\_or\_levels, 63  
  
write.IAMCcsv (write.mif), 64  
write.IAMCxlsx, 64  
write.mif, 64