Package ‘mclm’

October 13, 2022

Type Package

Title Mastering Corpus Linguistics Methods

Version 0.2.7

Description Read, inspect and process corpus files for quantitative corpus linguistics.
Obtain concordances via regular expressions, tokenize texts,
and compute frequencies and association measures. Useful for collocation analysis,
keywords analysis and variationist studies (comparison of linguistic variants
and of linguistic varieties).

Encoding UTF-8

Depends ca, tibble

Imports crayon, dplyr, Rcpp, readr, stringi, stringr, tm, xml2, yaml

LinkingTo Rcpp

Suggests MASS, tidyr, purrr, testthat (>= 3.0.0), covr

License GPL-2

RoxygenNote 7.2.1

URL `https://github.com/masterclm/mclm`,
`https://masterclm.github.io/mclm/`

Config/testthat/edition 3

NeedsCompilation yes

Author Dirk Speelman [aut],
Mariana Montes [aut, cre]

Maintainer Mariana Montes `<mariana.montes@kuleuven.be>`

Repository CRAN

Date/Publication 2022-10-03 07:50:02 UTC

R topics documented:

`assoc_scores` *Association scores used in collocation analysis and keyword analysis*

Description

`assoc_scores` and `assoc_abcd` take as their arguments co-occurrence frequencies of a number of items and return a range of association scores used in collocation analysis, collostruction analysis and keyword analysis.

Usage

```
assoc_scores(
  x,
  y = NULL,
  min_freq = 3,
  measures = NULL,
  with_variants = FALSE,
  show_dots = FALSE,
  p_fisher_2 = FALSE,
  haldane = TRUE,
  small_pos = 1e-05
)

assoc_abcd(
  a,
  b,
  c,
  d,
  types = NULL,
  measures = NULL,
  with_variants = FALSE,
  show_dots = FALSE,
  p_fisher_2 = FALSE,
  haldane = TRUE,
  small_pos = 1e-05
)
```

Arguments

`x`	Either an object of class `freqlist` or an object of class `cooc_info`. If `x` is a `freqlist`, it is interpreted as the target frequency list (i.e. the list with the frequency of items in the target context) and `y` must be a `freqlist` with the frequency of items in the reference context. If `x` is an object of class `cooc_info` instead, it is interpreted as containing target frequency information, reference frequency information and corpus size information.
`y`	An object of class `freqlist` with the frequencies of the reference context if `x` is also a `freqlist`. If `x` is an object of class `cooc_info`, this argument is ignored.
`min_freq`	Minimum value for `a[[i]]` (or for the frequency of an item in the target frequency list) needed for its corresponding item to be included in the output.
`measures`	Character vector containing the association measures (or related quantities) for which scores are requested. Supported measure names (and related quantities) are described in `Value` below. If `measures` is `NULL`, it is interpreted as short for the default selection, i.e. `c("exp_a", "DP_rows", "RR_rows", "OR", "MS", "Dice", "PMI", "chi2_signed", "G_signed", "t", "fisher")`. If `measures` is `"ALL"`, all supported measures are calculated (but not necessarily all the variants; see `with_variants`).
`with_variants`	Logical. Whether, for the requested `measures`, all variants should be included in the output (`TRUE`) or only the main version (`FALSE`). See also `p_fisher_2`.
`show_dots`	Logical. Whether a dot should be shown in console each time calculations for a measure are finished.
`p_fisher_2`	Logical. only relevant if `"fisher"` is included in `measures`. If `TRUE`, the p-value for a two-sided test (testing for either attraction or repulsion) is also calculated. By default, only the (computationally less demanding) p-value for a one-sided test is calculated. See `Value` for more details.

`haldane`	Logical. Should the Haldane-Anscombe correction be used? (See the Details section.) If `haldane` is `TRUE`, and there is at least one zero frequency in a contingency table, the correction is used for all measures calculated for that table, not just for measures that need this to be done.
`small_pos`	Alternative (but sometimes inferior) approach to dealing with zero frequencies, compared to `haldane`. The argument `small_pos` only applies when `haldane` is set to `FALSE`. (See the Details section.) If `haldane` is `FALSE`, and there is at least one zero frequency in a contingency table, adding small positive values to the zero frequency cells is done systematically for all measures calculated for that table, not just for measures that need this to be done.
`a`	Numeric vector expressing how many times some tested item occurs in the target context. More specifically, `a[[i]]`, with `i` an integer, expresses how many times the `i`-th tested item occurs in the target context.
`b`	Numeric vector expressing how many times other items than the tested item occur in the target context. More specifically, `b[[i]]`, with `i` an integer, expresses how many times *other* items than the `i`-th tested item occur in the target context.
`c`	Numeric vector expressing how many times some tested item occurs in the reference context. More specifically, `c[[i]]`, with `i` an integer, expresses how many times the `i`-th tested item occurs in the reference context.
`d`	Numeric vector expressing how many times items other than the tested item occur in the reference context. More specifically, `d[[i]]`, with `i` an integer, expresses how many times *other* items than the `i`-th tested item occur in the reference context.
`types`	A character vector containing the names of the linguistic items of which the association scores are to be calculated, or `NULL`. If `NULL`, `assoc_abcd()` creates dummy types such as `"t001"`, `"t002"`, etc.

Details

Input and output:

`assoc_scores()` takes as its arguments a target frequency list and a reference frequency lists (either as two `freqlist` objects or as a `cooc_info` object) and returns a number of popular measures expressing, for (almost) every item in either one of these lists, the extent to which the item is attracted to the target context, when compared to the reference context. The "almost" is added between parentheses because, with the default settings, some items are automatically excluded from the output (see `min_freq`).

`assoc_abcd()` takes as its arguments four vectors `a`, `b`, `c`, and `d`, of equal length. Each tuple of values `(a[i], b[i], c[i], d[i])`, with `i` some integer number between 1 and the length of the vectors, is assumed to represent the four numbers *a*, *b*, *c*, *d* in a contingency table of the type:

	tested item	**any other item**	**total**
target context	*a*	*b*	*m*
reference context	*c*	*d*	*n*
total	*k*	*l*	*N*

In the above table m, n, k, l and N are marginal frequencies. More specifically, $m = a + b$, $n = c + d$, $k = a + c$, $l = b + d$ and $N = m + n$.

Dealing with zeros:

Several of the association measures break down when one or more of the values `a`, `b`, `c`, and `d` are zero (for instance, because this would lead to division by zero or taking the log of zero). This can be dealt with in different ways, such as the Haldane-Anscombe correction.

Strictly speaking, Haldane-Anscombe correction specifically applies to the context of (log) odds ratios for two-by-two tables and boils down to adding `0.5` to each of the four values `a`, `b`, `c`, and `d` in every two-by-two contingency table for which the original values `a`, `b`, `c`, and `d` would not allow us to calculate the (log) odds ratio, which happens when one (or more than one) of the four cells is zero. Using the Haldane-Anscombe correction, the (log) odds ratio is then calculated on the bases of these 'corrected' values for `a`, `b`, `c`, and `d`.

However, because other measures that do not compute (log) odds ratios might also break down when some value is zero, all measures will be computed on the 'corrected' contingency matrix.

If the `haldane` argument is set to `FALSE`, division by zero or taking the log of zero is avoided by systematically adding a small positive value to all zero values for `a`, `b`, `c`, and `d`. The argument `small_pos` determines which small positive value is added in such cases. Its default value is `0.00001`.

Value

An object of class `assoc_scores`. This is a kind of data frame with as its rows all items from either the target frequency list or the reference frequency list with a frequency larger than `min_freq` in the target list, and as its columns a range of measures that express the extent to which the items are attracted to the target context (when compared to the reference context). Some columns don't contain actual measures but rather additional information that is useful for interpreting other measures.

Possible columns:

The following sections describe the (possible) columns in the output. All of these measures are reported if `measures` is set to `"ALL"`. Alternatively, each measure can be requested by specifying its name in a character vector given to the `measures` argument. Exceptions are described in the sections below.

Observed and expected frequencies:

- `a`, `b`, `c`, `d`: The frequencies in cells a, b, c and d, respectively. If one of them is `0`, they will be augmented by 0.5 or `small_pos` (see `Details`). These output columns are always present.
- `dir`: The direction of the association: `1` in case of relative attraction between the tested item and the target context (if $\frac{a}{m} \geq \frac{c}{n}$) and `-1` in case of relative repulsion between the target item and the target context (if $\frac{a}{m} < cn$).
- `exp_a`, `exp_b`, `exp_c`, `exp_d`: The expected values for cells a, b, c and d, respectively. All these columns will be included if `"expected"` is in `measures`. `exp_a` is also one of the default measures and is therefore included if `measures` is `NULL`. The values of these columns are computed as follows:
  - `exp_a` = $\frac{m \times k}{N}$
  - `exp_b` = $\frac{m \times l}{N}$
  - `exp_c` = $\frac{n \times k}{N}$
  - `exp_d` = $\frac{n \times l}{N}$

Effect size measures:
Some of these measures are based on proportions and can therefore be computed either on the rows or on the columns of the contingency table. Each measure can be requested on its own, but pairs of measures can also be requested with the first part of their name, as indicated in their corresponding descriptions.

- `DP_rows` and `DP_cols`: The difference of proportions, sometimes also called Delta-p (Δp), between rows and columns respectively. Both columns are present if `"DP"` is included in `measures`. `DP_rows` is also included if `measures` is `NULL`. They are calculated as follows:
  - `DP_rows` = $\frac{a}{m} - \frac{c}{n}$
  - `DP_cols` = $\frac{a}{k} - \frac{b}{l}$
- `perc_DIFF_rows` and `perc_DIFF_cols`: These measures can be seen as normalized versions of Delta-p, i.e. essentially the same measures divided by the denominator and multiplied by `100`. They therefore express how large the difference of proportions is, relative to the reference proportion. The multiplication by `100` turns the resulting 'relative difference of proportion' into a percentage. Both columns are present if `"perc_DIFF"` is included in `measures`. They are calculated as follows:
  - `perc_DIFF_rows` = $100 * \frac{(a/m)-(c/n)}{c/n}$
  - `perc_DIFF_cols` = $100 * \frac{(a/k)-(b/l)}{c/n}$
- `DC_rows` and `DC_cols`: The difference coefficient can be seen as a normalized version of Delta-p, i.e. essentially dividing the difference of proportions by the sum of proportions. Both columns are present if `"DC"` is included in `measures`. They are calculated as follows:
  - `DC_rows` = $\frac{(a/m)-(c/n)}{(a/m)+(c/n)}$
  - `DC_cols` = $\frac{(a/k)-(b/l)}{(a/k)+(b/l)}$
- `RR_rows` and `RR_cols`: Relative risk for the rows and columns respectively. `RR_rows` represents then how large the proportion in the target context is, relative to the proportion in the reference context. Both columns are present if `"RR"` is included in `measures`. `RR_rows` is also included if `measures` is `NULL`. They are calculated as follows:
  - `RR_rows` = $\frac{a/m}{c/n}$
  - `RR_cols` = $\frac{a/k}{b/l}$
- `LR_rows` and `LR_cols`: The so-called 'log ratio' of the rows and columns, respectively. It can be seen as a transformed version of the relative risk, viz. its binary log. Both columns are present if `"LR"` is included in `measures`. They are calculated as follows:
  - `LR_rows` = $\log_2\left(\frac{a/m}{c/n}\right)$
  - `LR_cols` = $\log_2\left(\frac{a/k}{b/l}\right)$

Other measures use the contingency table in a different way and therefore don't have a complementary row/column pair. In order to retrieve these columns, if `measures` is not `"ALL"`, their name must be in the `measures` vector. Some of them are included by default, i.e. if `measures` is `NULL`.

- `OR`: The odds ratio, which can be calculated either as $\frac{a/b}{c/d}$ or as $\frac{a/c}{b/d}$. This column is present `measures` is `NULL`.
- `log_OR`: The log odds ratio, which can be calculated either as $\log\left(\frac{a/b}{c/d}\right)$ or as $\log\left(\frac{a/c}{b/d}\right)$. In other words, it is the natural log of the odds ratio.
- `MS`: The minimum sensitivity, which is calculated as $\min(\frac{a}{m}, \frac{a}{k})$. In other words, it is either $\frac{a}{m}$ or $\frac{a}{k}$, whichever is lowest. This column is present `measures` is `NULL`.

- `Jaccard`: The Jaccard index, which is calculated as $\frac{a}{a+b+c}$. It expresses a, which is the frequency of the test item in the target context, relative to $b + c + d$, i.e. the frequency of all other contexts.
- `Dice`: The Dice coefficient, which is calculated as $\frac{2a}{m+k}$. It expresses the harmonic mean of $\frac{a}{m}$ and $\frac{a}{k}$ This column is present `measures` is `NULL`.
- `logDice`: An adapted version of the Dice coefficient. It is calculated as $14 + \log_2\left(\frac{2a}{m+k}\right)$. In other words, it is `14` plus the binary log of the Dice coefficient.
- `phi`: The phi coefficient (ϕ), which is calculated as $\frac{(a\times d)-(b\times c)}{\sqrt{m\times n\times k\times l}}$.
- `Q`: Yule's Q, which is calculated as $\frac{(a\times d)-(b\times c)}{(a\times d)(b\times c)}$.
- `mu`: The measure mu (μ), which is calculated as $\frac{a}{\mathrm{exp\_a}}$ (see `exp_a`).
- `PMI` and `pos_PMI`: (Positive) pointwise mutual information, which can be seen as a modification of the mu measure and is calculated as $\log_2\left(\frac{a}{\mathrm{exp\_a}}\right)$. In `pos_PMI`, negative values are set to `0`. The `PMI` column is present `measures` is `NULL`.
- `PMI2` and `PMI3`: Modified versions of `PMI` that aim to give relatively more weight to cases with relatively higher a. However, because of this modification, they are not pure effect size measures any more.
  - `PMI2` = $\log_2\left(\frac{a^2}{\mathrm{exp\_a}}\right)$
  - `PMI3` = $\log_2\left(\frac{a^3}{\mathrm{exp\_a}}\right)$

Strength of evidence measures:

The first measures in this section tend to come in triples: a test statistic, its p-value (preceded by `p_`) and its signed version (followed by `_signed`). The test statistics indicate evidence of either attraction or repulsion. Thus, in order to indicate the direction of the relationship, a negative sign is added in the "signed" version when $\frac{a}{k} < \frac{c}{l}$.

In each of these cases, the name of the main measure (e.g. `"chi2"`) and/or its signed counterpart (e.g. `"chi2_signed"`) must be in the `measures` argument, or `measures` must be `"ALL"`, for the columns to be included in the output. If the main function is requested, the signed counterpart will also be included, but if only the signed counterpart is requested, the non-signed version will be excluded. For the p-value to be retrieved, either the main measure or its signed version must be requested and, *additionally*, the `with_variants` argument must be set to `TRUE`.

- `chi2`, `p_chi2` and `chi2_signed`: The chi-squared test statistic (χ^2) as used in a chi-squared test of independence or in a chi-squared test of homogeneity for a two-by-two contingency table. Scores of this measure are high when there is strong evidence for attraction, but also when there is strong evidence for repulsion. The `chi2_signed` column is present if `measures` is `NULL`. `chi2` is calculated as follows:

$$\frac{(a-\mathrm{exp\_a})^2}{\mathrm{exp\_a}} + \frac{(b-\mathrm{exp\_b})^2}{\mathrm{exp\_b}} + \frac{(c-\mathrm{exp\_c})^2}{\mathrm{exp\_c}} + \frac{(d-\mathrm{exp\_d})^2}{\mathrm{exp\_d}}$$

.

- `chi2_Y`, `p_chi2_Y` and `chi2_Y_signed`: The chi-squared test statistic (χ^2) as used in a chi-squared test with Yates correction for a two-by-two contingency table. `chi2_Y` is calculated as follows:

$$\frac{(|a-\mathrm{exp\_a}|-0.5)^2}{\mathrm{exp\_a}} + \frac{(|b-\mathrm{exp\_b}|-0.5)^2}{\mathrm{exp\_b}} + \frac{(|c-\mathrm{exp\_c}|-0.5)^2}{\mathrm{exp\_c}} + \frac{(|d-\mathrm{exp\_d}|-0.5)^2}{\mathrm{exp\_d}}$$

.

- chi2_2T, p_chi2_2T and chi2_2T_signed: The chi-squared test statistic (χ^2) as used in a chi-squared goodness-of-fit test applied to the first column of the contingency table. The "2T" in the name stands for 'two terms' (as opposed to chi2, which is sometimes the 'four terms' version). chi2_2T is calculated as follows:

$$\frac{(a - \text{exp\_a})^2}{\text{exp\_a}} + \frac{(c - \text{exp\_c})^2}{\text{exp\_c}}$$

.

- chi2_2T_Y, p_chi2_2T_Y and chi2_2T_Y_signed: The chi-squared test statistic (χ^2) as used in a chi-squared goodness-of-fit test with Yates correction, applied to the first column of the contingency table. chi2_2T_Y is calculated as follows:

$$\frac{(|a - \text{exp\_a}| - 0.5)^2}{\text{exp\_a}} + \frac{(|c - \text{exp\_c}| - 0.5)^2}{\text{exp\_c}}$$

.

- G, p_G and G_signed: G test statistic, which is also sometimes called log-likelihood ratio (LLR) and, somewhat confusingly, G-squared. This is the test statistic as used in a log-likelihood ratio test for independence or homogeneity in a two-by-two contingency table. Scores are high in case of strong evidence for attraction, but also in case of strong evidence of repulsion. The G_signed column is present if measures is NULL. G is calculated as follows:

$$2\left(a \times \log(\frac{a}{\text{exp\_a}}) + b \times \log(\frac{b}{\text{exp\_b}}) + c \times \log(\frac{c}{\text{exp\_c}}) + d \times \log(\frac{d}{\text{exp\_d}})\right)$$

- G_2T, p_G_2T and G_2T_signed: The test statistic used in a log-likelihood ratio test for goodness-of-fit applied to the first column of the contingency table. The "2T" stands for 'two terms'. G_2T is calculated as follows:

$$2\left(a \times \log(\frac{a}{\text{exp\_a}}) + c \times \log(\frac{c}{\text{exp\_c}})\right)$$

The final two groups of measures take a different shape. The _as_chisq1 columns compute qchisq(1 - p, 1), with p being the p-values they are transforming, i.e. the p right quantile in a χ^2 distribution with one degree of freedom (see p_to_chisq1()).

- t, p_t_1, t_1_as_chisq1, p_t_2 and t_2_as_chisq1: The t-test statistic, used for a t-test for the proportion $\frac{a}{N}$ in which the null hypothesis is based on $\frac{k}{N} \times \frac{m}{N}$. Column t is present if "t" is included in measures or if measures is "ALL" or NULL. The other four columns are present if t is requested and if, additionally, with_variants is TRUE.
  - t = $\frac{a/N + k/N + m/N}{\sqrt{((a/N) \times (1 - a/N))/N}}$
  - p_t_1 is the p-value that corresponds to t when assuming a one-tailed test that only looks at attraction; t_1_as_chisq1 is its transformation.
  - p_t_2 is the p-value that corresponds to t when assuming a two-tailed test, viz. that looks at both attraction and repulsion; t_2_as_chisq1 is its transformation.
- p_fisher_1, fisher_1_as_chisq1, p_fisher_1r, fisher_1r_as_chisq1: The p-value of a one-sided Fisher exact test. The column p_fisher_1 is present if either "fisher" or "p_fisher" are in measures or if measures is "ALL" or NULL. The other columns are present if p_fisher_1 as been requested and if, additionally, with_variants is TRUE.

  - p_fisher_1 and p_fisher_1r are the p-values of the Fisher exact test that look at attraction and repulsion respectively.
  - fisher_1_as_chisq1 and fisher_1r_as_chisq1 are their respective transformations..
- p_fisher_2 and fisher_2_as_chisq1: p-value for a two-sided Fisher exact test, viz. looking at both attraction and repulsion. p_fisher_2 returns the p-value and fisher_2_as_chisq1 is its transformation. The p_fisher_2 column is present if either "fisher" or "p_fisher_1" are in measures or if measures is "ALL" or NULL and if, additionally, p_fisher_2 is TRUE. fisher_2_as_chisq1 is present if p_fisher_2 was requested and, additionally, with_variants is TRUE.

Properties of the class:

An object of class assoc_scores has:

- associated as.data.frame(), print(), sort() and tibble::as_tibble() methods,
- an interactive explore() method and useful getters, viz. n_types() and type_names().

An object of this class can be saved to file with write_assoc() and read with read_assoc().

Examples

```
assoc_abcd(10 , 200, 100,  300, types = "four")
assoc_abcd(30, 1000,  14, 5000, types = "fictitious")
assoc_abcd(15, 5000,  16, 1000, types = "toy")
assoc_abcd( 1,  300,   4, 6000, types = "examples")

a <- c(10,    30,    15,    1)
b <- c(200, 1000,  5000,  300)
c <- c(100,   14,    16,    4)
d <- c(300, 5000, 10000, 6000)
types <- c("four", "fictitious", "toy", "examples")
(scores <- assoc_abcd(a, b, c, d, types = types))

as_data_frame(scores)
as_tibble(scores)

print(scores, sort_order = "PMI")
print(scores, sort_order = "alpha")
print(scores, sort_order = "none")
print(scores, sort_order = "nonsense")

print(scores, sort_order = "PMI",
      keep_cols = c("a", "exp_a", "PMI", "G_signed"))
print(scores, sort_order = "PMI",
      keep_cols = c("a", "b", "c", "d", "exp_a", "G_signed"))
print(scores, sort_order = "PMI",
     drop_cols = c("a", "b", "c", "d", "exp_a", "G_signed",
                   "RR_rows", "chi2_signed", "t"))
```

as_character *Coerce object to character*

Description

This method turns its argument x, or at least part of the information in it, into a character vector.

Usage

```
as_character(x, ...)

## Default S3 method:
as_character(x, ...)

## S3 method for class 're'
as_character(x, ...)

## S3 method for class 'tokens'
as_character(x, ...)
```

Arguments

x	Object to coerce to character
...	Additional arguments

Value

Object of class character

Examples

```
(tks <- tokenize("The old man and the sea."))
as_character(tks) # turn 'tokens' object into character vector
as.character(tks) # alternative approach

as_character(1:10)
as.character(1:10)

regex <- re("(?xi) ^ .*")
as_character(regex) # turn 're' object into character vector
as.character(regex) # alternative approach
```

as_conc — *Coerce data frame to a concordance object*

Description

This function coerces a data frame to an object of the class `conc`.

Usage

```
as_conc(x, left = NA, match = NA, right = NA, keep_original = FALSE, ...)
```

Arguments

`x`	A data frame.
`left`	The name of the column in `x` that contains the left co-text of the concordance. Is `is.na(left)`, then this column is assumed to have the name `"left"`.
`match`	The name of the column in `x` that contains the match of the concordance. Is `is.na(match)`, then this column is assumed to have the name `"match"`.
`right`	The name of the column in `x` that contains the right co-text of the concordance. Is `is.na(right)`, then this column is assumed to have the name `"right"`.
`keep_original`	Logical. If the values of `left`, `match` or `right` are not `NA`, should the original names of those columns be kept in the `conc` object.
`...`	Additional arguments.

Value

Object of class `conc`, a kind of data frame with as its rows the matches and with the following columns:

- `glob_id`: Number indicating the position of the match in the overall list of matches.
- `id`: Number indicating the position of the match in the list of matches for one specific query.
- `source`: Either the filename of the file in which the match was found (in case of the setting `as_text = FALSE`), or the string '-' (in case of the setting `as_text = TRUE`).
- `left`: The left-hand side co-text of each match.
- `match`: The actual match.
- `right`: The right-hand side co-text of each match.

It also has additional attributes and methods such as:

- base `as_data_frame()` and `print()` methods, as well as a `print_kwic()` function,
- an `explore()` method.

An object of class `conc` can be merged with another by means of `merge_conc()`. It can be written to file with `write_conc()` and then read with `read_conc()`. It is also possible to import concordances created by means other than `write_conc()` with `import_conc()`.

Examples

```
(conc_data <- conc('A very small corpus.', '\\w+', as_text = TRUE))
df <- as.data.frame(conc_data)
as_conc(df)
```

`as_data_frame` *Coerce object to a data frame*

Description

`as_data_frame()` is an alternative to `as.data.frame()`. A number of objects in mclm can be turned into dataframes with one of these functions.

Usage

```
as_data_frame(x, row.names = NULL, optional = FALSE, ...)

## Default S3 method:
as_data_frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'assoc_scores'
as.data.frame(x, ...)

## S3 method for class 'conc'
as.data.frame(x, ...)

## S3 method for class 'fnames'
as.data.frame(x, ...)

## S3 method for class 'freqlist'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'details.slma'
as.data.frame(x, ...)

## S3 method for class 'slma'
as.data.frame(x, ...)

## S3 method for class 'tokens'
as.data.frame(x, ...)

## S3 method for class 'types'
as.data.frame(x, ...)
```

Arguments

`x`	Object to coerce to data.frame.
`row.names`	`NULL` or a character vector giving the rownames for the dataframe.
`optional`	Logical. If `TRUE`, setting rownames and converting column names is optional (see `as.data.frame()`).
`...`	Additional arguments

Value

Object of class `data.frame`

Examples

```
# for an assoc_scores object ---------------------
a <- c(10,    30,    15,    1)
b <- c(200, 1000,  5000,  300)
c <- c(100,   14,    16,    4)
d <- c(300, 5000, 10000, 6000)
types <- c("four", "fictitious", "toy", "examples")
(scores <- assoc_abcd(a, b, c, d, types = types))

as.data.frame(scores)
as_data_frame(scores)

# for a conc object ------------------------------
(conc_data <- conc('A very small corpus.', '\\w+', as_text = TRUE))
as.data.frame(conc_data)

# for an fnames object ---------------------------
cwd_fnames <- as_fnames(c('file1', 'file2'))
as.data.frame(cwd_fnames)

# for a freqlist, types or tokens object ---------
toy_corpus <- "Once upon a time there was a tiny toy corpus.
  It consisted of three sentences. And it lived happily ever after."
(flist <- freqlist(toy_corpus, as_text = TRUE))
as.data.frame(flist)

(flist2 <- keep_re(flist, "^..?$"))
as.data.frame

(toks <- tokenize(toy_corpus))
as.data.frame(toks)

(toks <- tokenize(toy_corpus))
as.data.frame(toks)
```

`as_fnames` *Coerce object to 'fnames'*

Description

This function coerces a character vector into an object of class `fnames`.

Usage

```
as_fnames(x, remove_duplicates = TRUE, sort = TRUE, ...)
```

Arguments

`x`	A character vector (or a `freqlist` object!)
`remove_duplicates`	Boolean. Whether duplicates should be removed.
`sort`	Boolean. Whether the output should be sorted.
`...`	Additional arguments.

Value

An object of class `fnames`.

Examples

```
as_fnames("path/to/my/corpus_file")
```

`as_freqlist` *Coerce table to a frequency list*

Description

This function coerces an object of class `table` to an object of class `freqlist`.

Usage

```
as_freqlist(x, tot_n_tokens = NULL, sort_by_ranks = TRUE)
```

Arguments

`x`	Object of class `table` or named numeric vector that will be interpreted as such.
`tot_n_tokens`	Number representing the total number of tokens in the corpus from which the frequency list is derived. When `tot_n_tokens` is `NULL`, this total number of tokens will be taken to be the sum of the frequencies in `x`.
`sort_by_ranks`	Logical. If `TRUE`, the items in the frequency list are sorted by frequency rank. If `FALSE`, the items in the frequency list, depending on the input type, either are sorted alphabetically or are not sorted at all.

Value

An object of class `freqlist`, which is based on the class `table`. It has additional attributes and methods such as:

- base `print()`, `as_data_frame()`, `summary()` and `sort`,
- `tibble::as_tibble()`,
- an interactive `explore()` method,
- various getters, including `tot_n_tokens()`, `n_types()`, `n_tokens()`, values that are also returned by `summary()`, and more,
- subsetting methods such as `keep_types()`, `keep_pos()`, etc. including `[]` subsetting (see brackets).

Additional manipulation functions include `type_freqs()` to extract the frequencies of different items, `freqlist_merge()` to combine frequency lists, and `freqlist_diff()` to subtract a frequency list from another.

Objects of class `freqlist` can be saved to file with `write_freqlist()`; these files can be read with `read_freqlist()`.

See Also

`freqlist()`

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."

## make frequency list in a roundabout way
tokens <- tokenize(toy_corpus)
flist <- as_freqlist(table(tokens))
flist

## more direct procedure
freqlist(toy_corpus, as_text = TRUE)

## build frequency list from scratch: example 1
flist <- as_freqlist(c("a" = 12, "toy" = 53, "example" = 20))
flist

## build frequency list from scratch: example 2
flist <- as_freqlist(c("a" = 12, "toy" = 53, "example" = 20),
                     tot_n_tokens = 1300)
flist
```

as_numeric — *Coerce object to a numeric vector*

Description

This generic method turns its first argument x or at least part of the information in it into a numeric object. It is an alternative notation for base::as.numeric().

Usage

```
as_numeric(x, ...)

## Default S3 method:
as_numeric(x, ...)
```

Arguments

x	An object to coerce.
...	Additional arguments.

Value

A numeric vector.

Examples

```
(flist <- freqlist(tokenize("The old story of the old man and the sea.")))

# extract frequency counts from a frequency list
as_numeric(flist)
as.numeric(flist)

# preferable alternative
type_freqs(flist)
```

as_tokens — *Coerce object to class* tokens

Description

This function coerces a character object or another object that can be coerced to a character into an object of class tokens.

Usage

```
as_tokens(x, ...)
```

Arguments

`x`	Object to coerce.
`...`	Additional arguments (not implemented).

Value

An object of class `tokens`.

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."

tks <- tokenize(toy_corpus)
print(tks, n = 1000)

tks[3:12]
print(as_tokens(tks[3:12]), n = 1000)
as_tokens(tail(tks))
```

---

as_types *Coerce object to a vector of types*

Description

This function coerces an object, such as a character vector, to an object of class `types`.

Usage

```
as_types(x, remove_duplicates = TRUE, sort = TRUE, ...)
```

Arguments

`x`	Object to coerce
`remove_duplicates`	Logical. Should duplicates be removed from `x` prior to coercing to a vector of types.
`sort`	Logical. Should `x` be alphabetically sorted prior to coercing to a vector of types; this argument is ignored if `remove_duplicates` is `TRUE`, because the result of removing duplicates is always sorted.
`...`	Additional arguments (not implemented)

Value

An object of the class `types`, which is based on a character vector. It has additional attributes and methods such as:

- base `print()`, `as_data_frame()`, `sort()` and `base::summary()` (which returns the number of items and of unique items),
- `tibble::as_tibble()`,
- the `n_types()` getter and the `explore()` method,
- subsetting methods such as `keep_types()`, `keep_pos()`, etc. including `[]` subsetting (see brackets).

An object of class `types` can be merged with another by means of `types_merge()`, written to file with `write_types()` and read from file with `write_types()`.

See Also

`types()`

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."

flist <- freqlist(toy_corpus, re_token_splitter = "\\W+", as_text = TRUE)
print(flist, n = 1000)
(sel_types <- as_types(c("happily", "lived", "once")))
keep_types(flist, sel_types)
tks <- tokenize(toy_corpus, re_token_splitter = "\\W+")
print(tks, n = 1000)
tks[3:12] # idx is relative to selection
head(tks) # idx is relative to selection
tail(tks) # idx is relative to selection
```

`brackets` *Subset an object by different criteria*

Description

This method can be used to subset objects based on different criteria.

Usage

```
## S3 method for class 'fnames'
x[i, invert = FALSE, ...]

## S3 replacement method for class 'fnames'
x[i, invert = FALSE] <- value
```

```
## S3 method for class 'freqlist'
x[i, invert = FALSE, ...]

## S3 method for class 'tokens'
x[i, invert = FALSE, ...]

## S3 replacement method for class 'tokens'
x[i, invert = FALSE, ...] <- value

## S3 method for class 'types'
x[i, invert = FALSE, ...]

## S3 replacement method for class 'types'
x[i, invert = FALSE] <- value
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`i`	Selection criterion; depending on its class, it behaves differently.
`invert`	Logical. Whether the matches should be selected rather than the non-matches.
`...`	Additional arguments.
`value`	Value to assign.

Details

The subsetting method with the notation `[]`, applied to mclm objects, is part of a family of subsetting methods: see `keep_pos()`, `keep_re()`, `keep_types()` and `keep_bool()`. In this case, the argument `i` is the selection criterion and, depending on its class, the method behaves different:

- providing a `re` object is equivalent to calling `keep_re()`,
- providing a numeric vector is equivalent to calling `keep_pos()`,
- providing a logical vector is equivalent to calling `keep_bool()`,
- providing a `types` object or a character vector is equivalent to calling `keep_types()`.

When the notation `x[i, ...]` is used, it is also possible to set the `invert` argument to `TRUE` (which then is one of the additional arguments in `...`). This `invert` argument then serves the same purpose as the `invert` argument in the `keep_` methods, turning it into a `drop_` method.

Value

Object of the same class as `x` with the selected elements only.

See Also

Other subsetters: `keep_bool()`, `keep_pos()`, `keep_re()`, `keep_types()`

Examples

```
# For a 'freqlist' object --------------------
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

## like keep_re()
flist[re("[ao]")]
flist[re("[ao]"), invert = TRUE]

## like keep_pos()
flist[type_freqs(flist) < 2]
flist[ranks(flist) <= 3]
flist[ranks(flist) <= 3, invert = TRUE]
flist[2:3]

## like keep_bool()
(flist2 <- keep_bool(flist, type_freqs(flist) < 2))
flist2[orig_ranks(flist2) > 2]

## like keep_types()
flist[c("man", "and")]
flist[as_types(c("man", "and"))]

# For a 'types' object -----------------------
(tps <- as_types(letters[1:10]))

tps[c(1, 3, 5, 7, 9)]
tps[c(TRUE, FALSE)]
tps[c("a", "c", "e", "g", "i")]

tps[c(1, 3, 5, 7, 9), invert = TRUE]
tps[c(TRUE, FALSE), invert = TRUE]
tps[c("a", "c", "e", "g", "i"), invert = TRUE]

# For a 'tokens' object ----------------------
(tks <- as_tokens(letters[1:10]))

tks[re("[acegi]"), invert = TRUE]
tks[c(1, 3, 5, 7, 9), invert = TRUE]
tks[c(TRUE, FALSE), invert = TRUE]
tks[c("a", "c", "e", "g", "i"), invert = TRUE]
```

`cat_re` *Print a regular expression to the console*

Description

The function `cat_re()` prints a regular expression to the console. By default, the regular expression is not printed as an R string, but as a ‘plain regular expression’. More specifically, the regular expression is printed without surrounding quotation marks, and characters that are special characters in R strings (such as quotation marks and backslashes) are not escaped with a backslash. Also, by default, multi-line regular expressions are printed as single-line regular expressions with all regular expression comments removed.

Usage

```
cat_re(x, format = c("plain", "R"), as_single_line = TRUE)
```

Arguments

`x`	An object of class `re` or a character vector containing a regular expression. If `x` is a character vector of length higher than 1, only its first element will be used.
`format`	Character vector describing the requested format (as a `"plain"` regular expression or as an `"R"` string). If its length is higher than 1, only its first element will be used.
`as_single_line`	Logical. Whether `x` should be converted to a single line regular expression, therefore also removing all comments, prior to printing. If the length of this vector is larger than 1, only its first item will be used.

Details

WARNING: In the current implementation, the way the character # is handled is not guaranteed to be correct. More specifically, the code is not guaranteed to correctly distinguish between a # symbol that introduces a regular expression comment and a # symbol that doesn't do so. Firstly, there is no testing whether at the point of encountering # we're in free-spacing mode. Second, there is no thorough testing whether or not the # symbol is part of a character class. However, # is processed correctly as long as any 'literal #' is immediately preceded by either a backslash or an opening square bracket, and any 'comment-introducing #' is not immediately preceded by a backslash or an opening square bracket.

Value

Invisibly, `x`.

See Also

`scan_re()`

Examples

```
# single-line regular expression
x <- "(?xi)  \\b \\w* willing \\w* \\b"
cat_re(x)
y <- "(?xi)
       \\b        # word boundary
       \\w*       # optional prefix
       willing   # stem
       \\w*       # optional suffix
       \\b        # word boundary"
cat_re(y)
cat_re(y, as_single_line = FALSE)
cat_re(y, format = "R")
cat_re(y, format = "R", as_single_line = FALSE)

regex <- re("(?xi)
                \\b        # word boundary
                \\w*       # optional prefix
                willing   # stem
                \\w*       # optional suffix
                \\b        # word boundary")
cat_re(regex)
cat_re(regex, as_single_line = FALSE)
```

---

ca_help *Helpers for plotting* ca *objects*

Description

The functions `row_pcoord()` and `col_pcoord()` retrieve the coordinates of the rows and columns of a `ca` object across all dimensions. The functions `xlim4ca()` and `ylim4ca()` return the range of values for the first and second dimensions.

Usage

```
row_pcoord(x, ...)

col_pcoord(x, ...)

xlim4ca(x, ...)

ylim4ca(x, ...)
```

Arguments

`x`	An object of class `ca`.
`...`	Additional arguments (not implemented).

Details

In the output of `row_pcoord()`, each row corresponds to a row from the dataframe that `ca::ca()` was applied to, and each column corresponds to a principal component. In the output of `col_pcoord()`, each row corresponds to a column from the dataframe that `ca::ca()` was applied to, and each column corresponds to a principal component.

Value

A matrix (for `row_pcoord()` and `col_pcoord()`) or a numeric vector (for `xlim4ca()` and `ylim4ca()`).

Functions

- `row_pcoord()`: Retrieve row principal coordinates for all dimensions
- `col_pcoord()`: Retrieve column principal coordinates for all dimensions
- `xlim4ca()`: Return range of first dimension for plotting
- `ylim4ca()`: Return range of second dimension for plotting

Examples

```
# traditional biplot from {ca}

library(ca)
data("author")
author_ca <- ca(author)
plot(author_ca)

# alternative plot with {mclm} tools
r_pc <- row_pcoord(author_ca)
c_pc <- col_pcoord(author_ca)
xlim <- xlim4ca(author_ca)
ylim <- ylim4ca(author_ca)
author_names <- as.factor(gsub(
                              "^.*?\\((.*?)\\)$", "\\1",
                             rownames(author), perl = TRUE))
plot(r_pc[,1], r_pc[,2], pch = 18,
    xlim = xlim, ylim = ylim, xlab = "", ylab = "",
    main = "authors and their alphabet",
    col = as.numeric(author_names))
abline(h = 0, col = "gray", lty = 3)
abline(v = 0, col = "gray", lty = 3)
text(c_pc[,1], c_pc[,2], colnames(author), col = "gray")
legend("topright",
       legend = levels(author_names),
       pch = rep(18, length(levels(author_names))),
       col = 1:length(levels(author_names)),
       title = "authors")
```

`chisq1_to_p` *Proportion of chi-squared distribution with one degree of freedom that sits to the right of x*

Description

Helper function that takes as its argument a numerical value `x` and that returns the proportion *p* of the chi-squared distribution with one degree of freedom that sits to the right of the value ‘x.

Usage

```
chisq1_to_p(x)
```

Arguments

`x`	A number.

Value

The proportion *p* of the chi-squared distribution with one degree of freedom that sits to the right of the value `x`.

See Also

`p_to_chisq1()`

`cleanup_spaces` *Clean up the use of whitespace in a character vector*

Description

The function `cleanup_spaces()` takes a character vector and input and turns any uninterrupted stretch of whitespace characters into one single space character. Moreover, it can also *remove* leading whitespace and trailing whitespace.

Usage

```
cleanup_spaces(x, remove_leading = TRUE, remove_trailing = TRUE)
```

Arguments

`x`	Character vector.
`remove_leading`	Logical. If `TRUE`, leading whitespace will be removed.
`remove_trailing`	Logical. If `TRUE`, trailing whitespace will be removed.

Value

A character vector.

Examples

```
txt <- "  A \\t  small      example \\n with redundant whitespace    "
cleanup_spaces(txt)
cleanup_spaces(txt, remove_leading = FALSE, remove_trailing = FALSE)
```

conc *Build a concordance for the matches of a regex*

Description

This function builds a concordance for the matches of a regular expression. The result is a dataset that can be written to a file with the function `write_conc()`. It mimics the behavior of the concordance tool in the program AntConc.

Usage

```
conc(
  x,
  pattern,
  c_left = 200,
  c_right = 200,
  perl = TRUE,
  re_drop_line = NULL,
  line_glue = "\n",
  re_cut_area = NULL,
  file_encoding = "UTF-8",
  as_text = FALSE
)
```

Arguments

`x`	A character vector determining which text is to be used as corpus. If `as_text = TRUE`, `x` is treated as the actual text to be used as corpus. If `as_text = FALSE` (the default), `x` is treated as a vector of filenames, interpreted as the names of the corpus files that contain the actual corpus data.
`pattern`	Character string containing the regular expression that serves as search term for the concordancer.
`c_left`	Number. How many characters to the left of each match must be included in the result as left co-text of the match.
`c_right`	Number. How many characters to the right of each match must be included in the result as right co-text of the match.
`perl`	If `TRUE`, `pattern` is treated as a PCRE flavor regular expression. Otherwise, `pattern` is treated as a regular expression in R's default flavor of regular expression.
`re_drop_line`	Character vector or `NULL`. If `NULL`, the argument is ignored. Otherwise, lines in `x` containing a match for `re_drop_line` are treated as not belonging to the corpus and are excluded from the results.
`line_glue`	Character vector or `NULL`. If `NULL`, the argument is ignored. Otherwise, all lines in the corpus are glued together in one character vector of length 1, with the string `line_glue` pasted in between consecutive lines. The value of `line_glue`

	can also be equal to the empty string (`""`). The ’line_glue’ operation is conducted immediately after the ’drop line’ operation.
`re_cut_area`	Character vector or `NULL`. If `NULL`, the argument is ignored. Otherwise, all matches in the corpus are ’cut out’ of the text prior to the identification of the tokens in the text (and are therefore not taken into account when identifying tokens). The ’cut area’ operation is conducted immediately after the ’line glue’ operation.
`file_encoding`	File encoding for reading each corpus file. Ignored if `as_text = TRUE`. Otherwise, it must be a character vector of length one (in which case the same encoding is used for all files) or with the same length as `x` (in which case each file can have a different encoding).
`as_text`	Logical. If `TRUE`, the content of `x` is treated as the actual text of the corpus (with each item within `x` treated as a separate ’document in RAM’). If `FALSE`, `x` is treated as a vector of filenames, interpreted as the names of the corpus files with the actual corpus data.

Details

In order to make sure that the columns `left`, `match`, and `right` in the output of `conc` do not contain any TAB or NEWLINE characters, whitespace in these items is being ’normalized’. More particularly, each stretch of whitespace, i.e. each uninterrupted sequences of whitespace characters, is replaced by a single SPACE character.

The values in the items the `glob_id` and `id` in the output of `conc` are always identical in a dataset that is the output of the function `conc`. The item `glob_id` only becomes useful when later, for instance, one wants to merge two datasets.#’

Value

Object of class `conc`, a kind of data frame with as its rows the matches and with the following columns:

- `glob_id`: Number indicating the position of the match in the overall list of matches.
- `id`: Number indicating the position of the match in the list of matches for one specific query.
- `source`: Either the filename of the file in which the match was found (in case of the setting `as_text = FALSE`), or the string ’-’ (in case of the setting `as_text = TRUE`).
- `left`: The left-hand side co-text of each match.
- `match`: The actual match.
- `right`: The right-hand side co-text of each match.

It also has additional attributes and methods such as:

- base `as_data_frame()` and `print()` methods, as well as a `print_kwic()` function,
- an `explore()` method.

An object of class `conc` can be merged with another by means of `merge_conc()`. It can be written to file with `write_conc()` and then read with `read_conc()`. It is also possible to import concordances created by means other than `write_conc()` with `import_conc()`.

Examples

```
(conc_data <- conc('A very small corpus.', '\\w+', as_text = TRUE))
print(conc_data)
print_kwic(conc_data)
```

---

create_cooc *Build collocation frequencies.*

Description

These functions builds a surface or textual collocation frequency for a specific node.

Usage

```
surf_cooc(
  x,
  re_node,
  w_left = 3,
  w_right = 3,
  re_boundary = NULL,
  re_drop_line = NULL,
  line_glue = NULL,
  re_cut_area = NULL,
  re_token_splitter = re("[^_\\p{L}\\p{N}\\p{M}'-]+"),
  re_token_extractor = re("[_\\p{L}\\p{N}\\p{M}'-]+"),
  re_drop_token = NULL,
  re_token_transf_in = NULL,
  token_transf_out = NULL,
  token_to_lower = TRUE,
  perl = TRUE,
  blocksize = 300,
  verbose = FALSE,
  dot_blocksize = 10,
  file_encoding = "UTF-8"
)

text_cooc(
  x,
  re_node,
  re_boundary = NULL,
  re_drop_line = NULL,
  line_glue = NULL,
  re_cut_area = NULL,
  re_token_splitter = re("[^_\\p{L}\\p{N}\\p{M}'-]+"),
  re_token_extractor = re("[_\\p{L}\\p{N}\\p{M}'-]+"),
  re_drop_token = NULL,
  re_token_transf_in = NULL,
  token_transf_out = NULL,
  token_to_lower = TRUE,
  perl = TRUE,
  blocksize = 300,
  verbose = FALSE,
  dot_blocksize = 10,
  file_encoding = "UTF-8"
)
```

Arguments

`x`	List of filenames of the corpus files.
`re_node`	Regular expression used for identifying instances of the 'node', i.e. the target item for which collocation information is collected.
`w_left`	Number of tokens to the left of the 'node' that are treated as belonging to the co-text of the 'node'. (But also see `re_boundary`.)
`w_right`	Number of tokens to the right of the 'node' that are treated as belonging to the co-text of the 'node'. (But also see `re_boundary`.)
`re_boundary`	Regular expression. For `text_cooc()`, it identifies boundaries between 'textual units'. For `surf_cooc()`, it identifies 'cut-off' points for the co-text of the 'node'. If it is not `NULL`, the maximum length of the left and right co-texts are still given by `w_left` and `w_right`, but if a match for `re_boundary` is found within the co-text, both the 'boundary token' and all tokens beyond it are excluded.
`re_drop_line`	Regular expression or `NULL`. if `NULL`, the argument is ignored. Otherwise, lines in the corpus that match it are treated as not belonging to the corpus and excluded from the results.
`line_glue`	Character vector or `NULL`. if `NULL`, the argument is ignored. Otherwise, all the lines in the corpus are glued together in one character vector of length 1, with the string `line_glue` pasted in between consecutive lines. This value can also be equal to an empty string `""`. The 'line glue' operation is conducted immediately after the 'drop line' operation.
`re_cut_area`	Regular expression or `NULL`. if `NULL`, the argument is ignored. Otherwise, all matches in the corpus are 'cut out' from the text prior to the identification of the tokens and are therefore not taken into account when identifying the tokens. The 'cut area' operation is conducted immediately after the 'line glue' operation.
`re_token_splitter`	Regular expression or `NULL`. if `NULL`, the argument is ignored and `re_token_extractor` is used instead. Otherwise, it identifies the areas between the tokens within a line of the corpus. The 'token identification' operation is conducted immediately after the 'cut area' operation.

`re_token_extractor`	Regular expression that identifies the locations of the actual tokens. It is only used if `re_token_splitter` is `NULL`. Currently the implementation of this argument is a lot less time-efficient than that of `re_token_splitter`. The 'token identification' operation is conducted immediately after the 'cut area' operation.
`re_drop_token`	Regular expression or `NULL`. if `NULL`, the argument is ignored. Otherwise, it identifies tokens to be excluded from the results. The 'drop token' operation is conducted immediately after the 'token identification' operation.
`re_token_transf_in`	A regular expression that identifies areas in the tokens that are to be transformed. This argument works together with `token_transf_out`. If either of them is `NULL`, they are both ignored. Otherwise, all matches in the tokens for `re_token_transf_in` are replaced with the replacement string `token_transf_out`. The 'token transformation' operation is conducted immediately after the 'drop token' transformation.
`token_transf_out`	A 'replacement string'. This argument works together with `re_token_transf_in` and is ignored if either argument is `NULL`.
`token_to_lower`	Logical. Whether tokens should be converted to lowercase before returning the results. The 'token to lower' operation is conducted immediately after the 'token transformation' operation.
`perl`	Logical. Whether the PCRE flavor of regular expressions should be used in the arguments that contain regular expressions.
`blocksize`	Number indicating how many corpus files are read to memory 'at each individual step' during the steps in the procedure. Normally the default value of `300` should not be changed, but when one works with exceptionally small corpus files, it may be worthwhile to use a higher number, and when one works with exceptionally large corpus files, it may be worthwhile to use a lower number.
`verbose`	Logical. If `TRUE`, messages are printed to the console to indicate progress.
`dot_blocksize`	Logical. If `TRUE`, dots are printed to the console to indicate progress.
`file_encoding`	Encoding of the input files. Either a character vector of length 1, in which case all files are assumed to be in the same encoding, or a character vector with the same length as `x`, which allows for different encodings for different files.

Details

Two major steps can be distinguished in the procedure conducted by these functions. The first major step is the *identification of the (sequence of) tokens* that, for the purpose of this analysis, will be considered to be the content of the corpus.

The function arguments that jointly determine the details of this step are `re_drop_line`, `line_glue`, `re_cut_area`, `re_token_splitter`, `re_token_extractor`, `re_drop_token`, `re_token_transf_in`, token_transf_out, and token_to_lower. The sequence of tokens that is the ultimate outcome of this step is then handed over to the second major step of the procedure.

The second major step is the *establishment of the co-occurrence frequencies*. The function arguments that jointly determine the details of this step are re_node and re_boundary for both functions, and w_left and w_right for surf_cooc() only. It is important to know that this second step is conducted after the tokens of the corpus have been identified, and that it is applied to a sequence of tokens, not to the original text. More specifically the regular expressions re_node and re_boundary are tested against individual tokens, as they are identified by the token identification procedure. Moreover, in surf_cooc(), the numbers w_left and w_right also apply to tokens a they are identified by the token identification procedure.

Value

An object of class cooc_info, containing information on co-occurrence frequencies.

Functions

- surf_cooc(): Build surface collocation frequencies
- text_cooc(): Build textual collocation frequencies

---

details *Details on a specific item*

Description

This method zooms in on details of an object x based on an item y. When x is of class slma (currently the only supported class), y must be one of the lexical markers described in it.

Usage

```
details(x, y, ...)

## S3 method for class 'slma'
details(x, y, shorten_names = TRUE, ...)
```

Arguments

x	An object containing global statistics for a collection of linguistic units, such as an object of class slma.
y	A character vector of length one representing one linguistic item.
...	Additional arguments.
shorten_names	Logical. If TRUE, filenames in the rownames are shortened with short_names().

Value

An object with details. When `x` is of class `slma`, the class of the output is `details.slma`, namely a list with the following items:

- `summary`: The row of `x$scores` corresponding to `y`.
- `scores` (what is printed by default), a dataframe with one row per pair of documents in the `slma` and the frequencies and association scores of the chosen item as columns.
- `item`: the value of `y`.
- `sig_cutoff` and `small_pos`, as defined in `slma`.

Examples

```
a_corp <- get_fnames(system.file("extdata", "cleveland", package = "mclm"))
b_corp <- get_fnames(system.file("extdata", "roosevelt", package = "mclm"))
slma_ex <- slma(a_corp, b_corp, keep_intermediate = TRUE)

gov <- details(slma_ex, "government")
gov$summary

# A bit of tidy manipulation to shorten filenames
if (require("dplyr") && require("tidyr")) {
  as_tibble(gov, rownames = "files") %>%
     tidyr::separate(files, into = c("file_A", "file_B"), sep = "--") %>%
     dplyr::mutate(dplyr::across(dplyr::starts_with("file"), short_names))
}
```

`drop_empty_rc` *Drop empty rows and columns from a matrix*

Description

With `x` a matrix containing frequency counts, `drop_empty_rc` makes a copy of `x` from which the all-zero rows and all-zero columns are removed. No checks are performed by this function.

Usage

```
drop_empty_rc(x)
```

Arguments

`x`	A matrix, assumed to contain frequency counts.

Details

This is just a convenience function. It is identical to, and implemented as, `x[rowSums(x) > 0, colSums(x) > 0, drop = FALSE]`.

Value

Matrix, with all-zero rows and columns removed.

Examples

```
# first example
m <- matrix(nrow = 3, byrow = TRUE,
            dimnames = list(c('r1','r2','r3'),
                            c('c1','c2','c3')),
           c(10, 0, 4,
             0, 0, 0,
             5, 0, 7))

m
m2 <- drop_empty_rc(m)
m2

## second example
m <- matrix(nrow = 3, byrow = TRUE,
           dimnames = list(c('r1','r2','r3'),
                           c('c1','c2','c3')),
           c(0, 0, 4,
             0, 0, 0,
             0, 0, 7))
m
m2 <- drop_empty_rc(m)
m2

## third example
m <- matrix(nrow = 3, byrow = TRUE,
            dimnames = list(c('r1','r2','r3'),
                            c('c1','c2','c3')),
           c(0, 0, 0,
             0, 0, 0,
             0, 0, 0))
m
m2 <- drop_empty_rc(m)
m2
```

`drop_tags` *Drop XML tags from character string*

Description

This function takes a character vector and returns a copy from which all XML-like tags have been removed. Moreover, if `half_tags_too = TRUE` any half tag at the beginning or end of `x` is also removed.

Usage

```
drop_tags(x, half_tags_too = TRUE)
```

Arguments

`x`	String with XML tag
`half_tags_too`	Logical. Whether tags with only opening/closing bracket should also be removed.

Details

This function is not XML-aware. It uses a very simple definition of what counts as a tag. More specifically, any character sequence starting with < and ending with > is considered a 'tag'; inside such a tag, between < and >, `drop_tags()` accepts any sequence of zero or more characters.

Value

Character string

Examples

```
xml_snippet <- "id='3'/><w pos='Det'>An</w> <w pos='N'>example</w> <w"
drop_tags(xml_snippet)
drop_tags(xml_snippet, half_tags_too = FALSE)
```

`explore` *Interactively navigate through an object*

Description

This method only works in an interactive R session to open 'exploration mode', in which the user can navigate through the object x by means of brief commands.

Usage

```
explore(x, ...)

## S3 method for class 'assoc_scores'
explore(
  x,
  n = 20,
  from = 1,
  from_col = 1,
  perl = TRUE,
  sort_order = c("none", "G_signed", "PMI", "alpha"),
  use_clear = TRUE,
  ...
)

## S3 method for class 'conc'
explore(x, n = 20, from = 1, use_clear = TRUE, ...)

## S3 method for class 'fnames'
explore(x, n = 20, from = 1, perl = TRUE, use_clear = TRUE, ...)

## S3 method for class 'freqlist'
explore(x, n = 20, from = 1, perl = TRUE, use_clear = TRUE, ...)

## S3 method for class 'tokens'
explore(x, n = 20, from = 1, perl = TRUE, use_clear = TRUE, ...)

## S3 method for class 'types'
explore(x, n = 20, from = 1, perl = TRUE, use_clear = TRUE, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`...`	Additional arguments.
`n`	Maximum number of items in the object to be printed at once.
`from`	Index of the first item to be printed.
`from_col`	Index of the first column to be displayed in the regular area (among all selected columns, including frozen columns). If `from_col` points
`perl`	Logical. Whether or not the regular expressions used in the exploration session use the PERL flavor of regular expression.
`sort_order`	Order in which the items are to be printed. In general, possible values are `"alpha"` (meaning that the items are to be sorted alphabetically), and `"none"` (meaning that the items are not to be sorted). If `x` is an object of class `assoc_scores`, a column name or vector of column names may be provided instead.
`use_clear`	Logical. If `TRUE`, and if the feature is supported by the R environment, the console will be cleared in between all interactive steps in the exploration session.

Details

`explore()` is different from other R instructions because it does not automatically stop executing and show a new regular prompt (>) in the console. Instead it shows a special prompt (>>) at which you can use `explore()`-specific commands. Note that at the special prompt >> none of the regular R instructions will work. The instructions that do work at this prompt, for `explore()`, are listed below. After each instruction the user must press `ENTER`.

- `b` (begin): The first items in `x` are shown.
- `e` (end): The last items in `x` are shown.
- `d` (down *n* items): The 'next page' of items is shown.
- `u` (up *n* items): The 'previous page' of items is shown.

- `n` (next item): The list/table shifts one item down the list.
- `p` (previous item): The list/table shifts one item up the list.
- `g {linenumber}` (go to...): Jump to line `{linenumber}`.
  E.g. `g 1000` will jump to the 1000th line.
- `f {regex}` (find...): Jump to the next item matching the regular expression `{regex}`.
  E.g. `f (?xi) astic $` will jump to the next item ending in `"astic"`. The software starts searching from the *second item* presently visible onward.
  `f` will jump to the next item matching the last regular expression used with `f {regex}`.
  This command is **not** available when `x` is a `conc` object.
- `l` (left): In `assoc_scores` objects, move one column to the left.
- `r` (right): In `assoc_scores` objects, move one column to the right.
- ?: A help page is displayed, showing all possible commands.
- `q` (quit): Terminate interactive session.

Value

Invisibly, `x`.

`find_xpath` *Run XPath query*

Description

This function finds matches for an XPath query in a corpus.

Usage

```
find_xpath(x, pattern, fun = NULL, final_fun = NULL, namespaces = NULL, ...)
```

Arguments

`x`	A corpus: an `fnames` object, a character vector of an XML source, or a document parsed with `xml2::read_xml()`.
`pattern`	An XPath query.
`fun`	Function to be applied to the individual nodes prior to returning the result.
`final_fun`	Function to be applied to the complete list of matches prior to returning the result.
`namespaces`	A namespace as generated by `xml2::xml_ns()`.
`...`	Additional arguments.

Value

A nodeset or the output of applying `fun` to a nodeset.

Examples

```
test_xml <- '
<p>
  <w pos="at">The</w>
  <w pos="nn">example</w>
  <punct>.</punct>
</p>'

find_xpath(test_xml, "//w")
find_xpath(test_xml, "//@pos")
find_xpath(test_xml, "//w[@pos='nn']")

find_xpath(test_xml, "//w", fun = xml2::xml_text)
find_xpath(test_xml, "//w", fun = xml2::xml_attr, attr = "pos")
```

`fnames` *Retrieve the names of files in a given path*

Description

Build an object of class `fnames`.

Usage

```
get_fnames(
  path = ".",
  re_pattern = NULL,
  recursive = TRUE,
  perl = TRUE,
  invert = FALSE
)
```

Arguments

`path`	The location of the files to be listed.
`re_pattern`	Optional regular expression. If present, then only the filenames that match it are retrieved (unless `invert = TRUE`, in which case those filenames are excluded). The match is done over the absolute path of the files.
`recursive`	Boolean value. Should the subdirectories of `path` also be searched?
`perl`	Boolean value. Whether `re_pattern` should be interpreted as a PERL flavor of regular expression.
`invert`	Boolean value. If `TRUE`, filenames matching `re_pattern` are the only ones retrieved. If `FALSE`, filenames matching `re_pattern` are excluded.

Value

An object of class `fnames`, which is a special kind of character vector storing the absolute paths of the corpus files. It has additional attributes and methods such as:

- base `print()`, `as_data_frame()`, `sort()` and `summary()` (which returns the number of items and of unique items),
- `tibble::as_tibble()`,
- an interactive `explore()` method,
- a function to get the number of items `n_fnames()`,
- subsetting methods such as `keep_types()`, `keep_pos()`, etc. including `[]` subsetting (see brackets), as well as the specific functions `keep_fnames()` and `drop_fnames()`.

Additional manipulation functions includes `fnames_merge()` to combine filenames collections and the `short_names()` family of functions to shorten the names.

Objects of class `fnames` can be saved to file with `write_fnames()`; these files can be read with `read_fnames()`.

It is possible to coerce a character vector into an `fnames` object with `as_fnames()`.

Examples

```
cwd_fnames <- get_fnames(recursive = FALSE)

cwd_fnames <- as_fnames(c("file1", "file2", "file3"))
cwd_fnames
print(cwd_fnames)
as_data_frame(cwd_fnames)
as_tibble(cwd_fnames)

sort(cwd_fnames)

summary(cwd_fnames)
```

`freqlist` *Build the frequency list of a corpus*

Description

This function builds the word frequency list from a corpus.

Usage

```
freqlist(
  x,
  re_drop_line = NULL,
  line_glue = NULL,
  re_cut_area = NULL,
  re_token_splitter = re("[^_\\p{L}\\p{N}\\p{M}'-]+"),
  re_token_extractor = re("[_\\p{L}\\p{N}\\p{M}'-]+"),
  re_drop_token = NULL,
  re_token_transf_in = NULL,
  token_transf_out = NULL,
  token_to_lower = TRUE,
  perl = TRUE,
  blocksize = 300,
  verbose = FALSE,
  show_dots = FALSE,
  dot_blocksize = 10,
  file_encoding = "UTF-8",
  ngram_size = NULL,
  max_skip = 0,
  ngram_sep = "_",
  ngram_n_open = 0,
  ngram_open = "[]",
  as_text = FALSE
)
```

Arguments

`x`	Either a list of filenames of the corpus files (if `as_text` is `TRUE`) or the actual text of the corpus (if `as_text` is `FALSE`). If `as_text` is `TRUE` and the length of the vector `x` is higher than one, then each item in `x` is treated as a separate line (or a separate series of lines) in the corpus text. Within each item of `x`, the character `"\\n"` is also treated as a line separator.
`re_drop_line`	`NULL` or character vector. If `NULL`, it is ignored. Otherwise, a character vector (assumed to be of length 1) containing a regular expression. Lines in `x` that contain a match for `re_drop_line` are treated as not belonging to the corpus and are excluded from the results.
`line_glue`	`NULL` or character vector. If `NULL`, it is ignored. Otherwise, all lines in a corpus file (or in `x`, if `as_text` is `TRUE`), are glued together in one character vector of length 1, with the string `line_glue` pasted in between consecutive lines. The value of `line_glue` can also be equal to the empty string `""`. The 'line glue' operation is conducted immediately after the 'drop line' operation.
`re_cut_area`	`NULL` or character vector. If `NULL`, it is ignored. Otherwise, all matches in a corpus file (or in `x`, if `as_text` is `TRUE`), are 'cut out' of the text prior to the identification of the tokens in the text (and are therefore not taken into account when identifying the tokens). The 'cut area' operation is conducted immediately after the 'line glue' operation.
`re_token_splitter`	Regular expression or `NULL`. Regular expression that identifies the locations where lines in the corpus files are split into tokens. (See Details.) The 'token identification' operation is conducted immediately after the 'cut area' operation.

`re_token_extractor`	Regular expression that identifies the locations of the actual tokens. This argument is only used if `re_token_splitter` is `NULL`. (See Details.) The 'token identification' operation is conducted immediately after the 'cut area' operation.
`re_drop_token`	Regular expression or `NULL`. If `NULL`, it is ignored. Otherwise, it identifies tokens that are to be excluded from the results. Any token that contains a match for `re_drop_token` is removed from the results. The 'drop token' operation is conducted immediately after the 'token identification' operation.
`re_token_transf_in`	Regular expression that identifies areas in the tokens that are to be transformed. This argument works together with the argument `token_transf_out`. If both `re_token_transf_in` and `token_transf_out` differ from `NA`, then all matches, in the tokens, for the regular expression `re_token_transf_in` are replaced with the replacement string `token_transf_out`. The 'token transformation' operation is conducted immediately after the 'drop token' operation.
`token_transf_out`	Replacement string. This argument works together with `re_token_transf_in` and is ignored if `re_token_transf_in` is `NULL` or `NA`.
`token_to_lower`	Logical. Whether tokens must be converted to lowercase before returning the result. The 'token to lower' operation is conducted immediately after the 'token transformation' operation.
`perl`	Logical. Whether the PCRE regular expression flavor is being used in the arguments that contain regular expressions.
`blocksize`	Number that indicates how many corpus files are read to memory `at each individual step' during th` should not be changed, but when one works with exceptionally small corpus files, it may be worthwhile to use a higher number, and when one works with exceptionally large corpus files, it may be worthwhile to use a lower number.
`verbose`	If`TRUE`, messages are printed to the console to indicate progress.
`show_dots, dot_blocksize`	If `TRUE`, dots are printed to the console to indicate progress.
`file_encoding`	File encoding that is assumed in the corpus files.
`ngram_size`	Argument in support of ngrams/skipgrams (see also `max_skip`). If one wants to identify individual tokens, the value of `ngram_size` should be `NULL` or `1`. If one wants to retrieve token ngrams/skipgrams, `ngram_size` should be an integer indicating the size of the ngrams/skipgrams. E.g. `2` for bigrams, or `3` for trigrams, etc.
`max_skip`	Argument in support of skipgrams. This argument is ignored if `ngram_size` is `NULL` or is `1`. If `ngram_size` is `2` or higher, and `max_skip` is `0`, then regular ngrams are being retrieved (albeit that they may contain open slots; see `ngram_n_open`). If `ngram_size` is `2` or higher, and `max_skip` is `1` or higher, then skipgrams are being retrieved (which in the current implementation cannot contain open slots; see `ngram_n_open`).

	For instance, if `ngram_size` is `3` and `max_skip` is `2`, then 2-skip trigrams are being retrieved. Or if `ngram_size` is `5` and `max_skip` is `3`, then 3-skip 5-grams are being retrieved.
`ngram_sep`	Character vector of length 1 containing the string that is used to separate/link tokens in the representation of ngrams/skipgrams in the output of this function.
`ngram_n_open`	If `ngram_size` is `2` or higher, and moreover `ngram_n_open` is a number higher than `0`, then ngrams with 'open slots' in them are retrieved. These ngrams with 'open slots' are generalizations of fully lexically specific ngrams (with the generalization being that one or more of the items in the ngram are replaced by a notation that stands for 'any arbitrary token'). For instance, if `ngram_size` is `4` and `ngram_n_open` is `1`, and if moreover the input contains a 4-gram `"it_is_widely_accepted"`, then the output will contain all modifications of `"it_is_widely_accepted"` in which one (since `ngram_n_open` is `1`) of the items in this n-gram is replaced by an open slot. The first and the last item inside an ngram are never turned into an open slot; only the items in between are candidates for being turned into open slots. Therefore, in the example, the output will contain `"it_[]_widely_accepted"` and `"it_is_[]_accepted"`. As a second example, if `ngram_size` is `5` and `ngram_n_open` is `2`, and if moreover the input contains a 5-gram `"it_is_widely_accepted_that"`, then the output will contain `"it_[]_[]_accepted_that"`, `"it_[]_widely_[]_that"`, and `"it_is_[]_[]_that"`.
`ngram_open`	Character string used to represent open slots in ngrams in the output of this function.
`as_text`	Logical. Whether `x` is to be interpreted as a character vector containing the actual contents of the corpus (if `as_text` is `TRUE`) or as a character vector containing the names of the corpus files (if `as_text` is `FALSE`). If if `as_text` is `TRUE`, then the arguments `blocksize`, `verbose`, `show_dots`, `dot_blocksize`, and `file_encoding` are ignored.

Details

The actual token identification is either based on the `re_token_splitter` argument, a regular expression that identifies the areas between the tokens, or on `re_token_extractor`, a regular expression that identifies the area that are the tokens. The first mechanism is the default mechanism: the argument `re_token_extractor` is only used if `re_token_splitter` is `NULL`. Currently the implementation of `re_token_extractor` is a lot less time-efficient than that of `re_token_splitter`.

Value

An object of class `freqlist`, which is based on the class `table`. It has additional attributes and methods such as:

- base `print()`, `as_data_frame()`, `summary()` and `sort`,
- `tibble::as_tibble()`,
- an interactive `explore()` method,
- various getters, including `tot_n_tokens()`, `n_types()`, `n_tokens()`, values that are also returned by `summary()`, and more,

- subsetting methods such as `keep_types()`, `keep_pos()`, etc. including `[]` subsetting (see brackets).

Additional manipulation functions include `type_freqs()` to extract the frequencies of different items, `freqlist_merge()` to combine frequency lists, and `freqlist_diff()` to subtract a frequency list from another.

Objects of class `freqlist` can be saved to file with `write_freqlist()`; these files can be read with `read_freqlist()`.

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."

(flist <- freqlist(toy_corpus, as_text = TRUE))
print(flist, n = 20)
as.data.frame(flist)
as_tibble(flist)
summary(flist)
print(summary(flist))

t_splitter <- "(?xi) [:\\s.;,?!\"]+"
freqlist(toy_corpus,
         re_token_splitter = t_splitter,
         as_text = TRUE)

freqlist(toy_corpus,
         re_token_splitter = t_splitter,
         token_to_lower = FALSE,
         as_text = TRUE)

t_extractor <- "(?xi) ( [:;?!] | [.]+ | [\\w'-]+ )"
freqlist(toy_corpus,
        re_token_splitter = NA,
        re_token_extractor = t_extractor,
        as_text = TRUE)

freqlist(letters, ngram_size = 3, as_text = TRUE)

freqlist(letters, ngram_size = 2, ngram_sep = " ", as_text = TRUE)
```

`freqlist_diff` *Subtract frequency lists*

Description

This function merges information from two frequency lists, subtracting the frequencies found in the second frequency lists from the frequencies found in the first list.

Usage

```
freqlist_diff(x, y)
```

Arguments

`x, y`	Objects of class `freqlist`.

Value

An object of class `freqlist`.

Examples

```
(flist1 <- freqlist("A first toy corpus.", as_text = TRUE))
(flist2 <- freqlist("A second toy corpus.", as_text = TRUE))

freqlist_diff(flist1, flist2)
```

`import_conc` *Import a concordance*

Description

This function imports a concordance from files generated by other means than `write_conc()`.

Usage

```
import_conc(x, file_encoding = "UTF-8", source_type = c("corpuseye"), ...)
```

Arguments

`x`	A vector of input filenames.
`file_encoding`	Encoding of the file(s).
`source_type`	Character string. How the file is read. Currently only `"corpuseye"` is supported.
`...`	Additional arguments (not implemented).

Value

An object of class `conc`.

See Also

`read_conc()` for files written with `write_conc()`.

keep_bool *Subset an object based on logical criteria*

Description

These methods can be used to subset objects based on a logical vector.

Usage

```
keep_bool(x, bool, invert = FALSE, ...)

drop_bool(x, bool, ...)

## S3 method for class 'fnames'
drop_bool(x, bool, ...)

## S3 method for class 'fnames'
keep_bool(x, bool, invert = FALSE, ...)

## S3 method for class 'freqlist'
drop_bool(x, bool, ...)

## S3 method for class 'freqlist'
keep_bool(x, bool, invert = FALSE, ...)

## S3 method for class 'tokens'
drop_bool(x, bool, ...)

## S3 method for class 'tokens'
keep_bool(x, bool, invert = FALSE, ...)

## S3 method for class 'types'
drop_bool(x, bool, ...)

## S3 method for class 'types'
keep_bool(x, bool, invert = FALSE, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`bool`	A logical vector of the same length as `x`. If `bool` is not of the correct length, it is *recycled*. Assuming `invert` is `FALSE`, those items are selected for which `bool` is `TRUE`.
`invert`	Logical. Whether the matches should be selected rather than the non-matches.
`...`	Additional arguments.

Details

The methods keep_pos() and drop_pos() are part of a family of methods of the mclm package used to subset different objects. The methods starting with keep_ extract the items in x based on the criterion specified by the second argument. In contrast, the methods starting with drop_ *exclude* the items that match the criterion in the same argument.

Calling a drop_ method is equivalent to calling its keep_ counterpart when the invert argument is TRUE.

Value

Object of the same class as x with the selected elements only.

See Also

Other subsetters: brackets, keep_pos(), keep_re(), keep_types()

Examples

```
# For a 'freqlist' object---------------------
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

keep_bool(flist, type_freqs(flist) < 2)
drop_bool(flist, type_freqs(flist) >= 2)
keep_bool(flist, ranks(flist) <= 3)

keep_bool(flist, c(FALSE, TRUE, TRUE, FALSE))

(flist2 <- keep_bool(flist, type_freqs(flist) < 2))
keep_bool(flist2, orig_ranks(flist2) > 2)

# For a 'types' object ----------------------
(tps <- as_types(letters[1:10]))

keep_bool(tps, c(TRUE, FALSE))
drop_bool(tps, c(TRUE, FALSE))

# For a 'tokens' object ----------------------
(tks <- as_tokens(letters[1:10]))

keep_bool(tks, c(TRUE, FALSE))
drop_bool(tks, c(TRUE, FALSE))
```

---

keep_fnames *Filter collection of filenames by name*

Description

The functions build a subset of an object of class fnames based on a vector of characters, either including them (with keep_fnames(invert = FALSE)) or excluding them (with keep_fnames(invert = FALSE) or drop_fnames()).

Usage

```
keep_fnames(x, y, invert = FALSE, ...)

drop_fnames(x, y, ...)
```

Arguments

`x`	An object of class `fnames`, to be filtered.
`y`	An object of class `fnames` or class `types` or a character vector. This is the filtering criterion.
`invert`	Boolean value. If `TRUE`, the elements in `y` are excluded rather than kept (and `keep_fnames()` behaves like `drop_fnames()`)
`...`	Additional arguments.

Value

An object of class `fnames`.

Examples

```
all_fnames <- as_fnames(c("file1", "file2", "file3",
                          "file4", "file5", "file6"))

unwanted_fnames <- as_fnames(c("file1", "file4"))
keep_fnames(all_fnames, unwanted_fnames, invert = TRUE)
drop_fnames(all_fnames, unwanted_fnames)

wanted_fnames <- as_fnames(c("file3", "file5"))
keep_fnames(all_fnames, wanted_fnames)
```

`keep_pos` *Subset an object by index*

Description

These methods can be used to subset objects based on a numeric vector of indices.

Usage

```
keep_pos(x, pos, invert = FALSE, ...)

## S3 method for class 'fnames'
drop_pos(x, pos, ...)

## S3 method for class 'fnames'
keep_pos(x, pos, invert = FALSE, ...)
```

```
## S3 method for class 'freqlist'
drop_pos(x, pos, ...)

## S3 method for class 'freqlist'
keep_pos(x, pos, invert = FALSE, ...)

drop_pos(x, pos, ...)

## S3 method for class 'tokens'
drop_pos(x, pos, ...)

## S3 method for class 'tokens'
keep_pos(x, pos, invert = FALSE, ...)

## S3 method for class 'types'
drop_pos(x, pos, ...)

## S3 method for class 'types'
keep_pos(x, pos, invert = FALSE, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`pos`	A numeric vector, the numbers in which identify positions (= indices) of items in `x`. If the numbers are positive, then their values point to the items that are to be selected. If the numbers are negative, then their absolute values point to the items that are not to be selected. Positive and negative numbers must not be mixed.
`invert`	Logical. Whether the matches should be selected rather than the non-matches.
`...`	Additional arguments.

Details

The methods `keep_pos()` and `drop_pos()` are part of a family of methods of the mclm package used to subset different objects. The methods starting with `keep_` extract the items in `x` based on the criterion specified by the second argument. In contrast, the methods starting with `drop_` *exclude* the items that match the criterion in the same argument.

Calling a `drop_` method is equivalent to calling its `keep_` counterpart when the `invert` argument is `TRUE`.

Value

Object of the same class as `x` with the selected elements only.

See Also

Other subsetters: `brackets`, `keep_bool()`, `keep_re()`, `keep_types()`

Examples

```
# For a 'freqlist' object --------------------
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

keep_pos(flist, c(2, 3))

# For a 'types' object -----------------------
(tps <- as_types(letters[1:10]))

keep_pos(tps, c(1, 3, 5, 7, 9))
drop_pos(tps, c(1, 3, 5, 7, 9))

# For a 'tokens' object ----------------------
(tks <- as_tokens(letters[1:10]))

keep_pos(tks, c(1, 3, 5, 7, 9))
drop_pos(tks, c(1, 3, 5, 7, 9))
```

`keep_re` *Subset an object based on regular expressions*

Description

These methods can be used to subset objects based on a regular expression.

Usage

```
keep_re(x, pattern, perl = TRUE, invert = FALSE, ...)

drop_re(x, pattern, perl = TRUE, ...)

## S3 method for class 'fnames'
drop_re(x, pattern, perl = TRUE, ...)

## S3 method for class 'fnames'
keep_re(x, pattern, perl = TRUE, invert = FALSE, ...)

## S3 method for class 'freqlist'
drop_re(x, pattern, perl = TRUE, ...)

## S3 method for class 'freqlist'
keep_re(x, pattern, perl = TRUE, invert = FALSE, ...)

## S3 method for class 'tokens'
drop_re(x, pattern, perl = TRUE, ...)

## S3 method for class 'tokens'
keep_re(x, pattern, perl = TRUE, invert = FALSE, ...)
```

```
## S3 method for class 'types'
drop_re(x, pattern, perl = TRUE, ...)

## S3 method for class 'types'
keep_re(x, pattern, perl = TRUE, invert = FALSE, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`pattern`	Either an object of the class `re` or a character vector of length one containing a regular expression.
`perl`	Logical. Whether `pattern` is treated as a PCRE flavor regular expression. The `perl` argument is only used if `pattern` is a regular character vector. If `pattern` is an object of the class `re`, then the `perl` argument is ignored, and the relevant information in the `re` object `pattern`, viz. the value of `pattern$perl`, is used instead.
`invert`	Logical. Whether the matches should be selected rather than the non-matches.
`...`	Additional arguments.

Details

The methods `keep_pos()` and `drop_pos()` are part of a family of methods of the mclm package used to subset different objects. The methods starting with `keep_` extract the items in `x` based on the criterion specified by the second argument. In contrast, the methods starting with `drop_` *exclude* the items that match the criterion in the same argument.

Calling a `drop_` method is equivalent to calling its `keep_` counterpart when the `invert` argument is `TRUE`.

Value

Object of the same class as `x` with the selected elements only.

See Also

Other subsetters: `brackets`, `keep_bool()`, `keep_pos()`, `keep_types()`

Examples

```
# For a 'freqlist' object --------------------
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

keep_re(flist, "[ao]")
drop_re(flist, "[ao]")
keep_re(flist, "[ao]", invert = TRUE) # same as drop_re()

# For a 'types' object -----------------------
(tps <- as_types(letters[1:10]))
```

```
keep_re(tps, "[acegi]")
drop_re(tps, "[acegi]")

# For a 'tokens' object ----------------------
(tks <- as_tokens(letters[1:10]))

keep_re(tks, "[acegi]")
drop_re(tks, "[acegi]")
```

keep_types *Subset an object based on a selection of types*

Description

These methods can be used to subset objects based on a list of types.

Usage

```
keep_types(x, types, invert = FALSE, ...)

drop_types(x, types, ...)

## S3 method for class 'fnames'
drop_types(x, types, ...)

## S3 method for class 'fnames'
keep_types(x, types, invert = FALSE, ...)

## S3 method for class 'freqlist'
drop_types(x, types, ...)

## S3 method for class 'freqlist'
keep_types(x, types, invert = FALSE, ...)

## S3 method for class 'tokens'
drop_types(x, types, ...)

## S3 method for class 'tokens'
keep_types(x, types, invert = FALSE, ...)

## S3 method for class 'types'
drop_types(x, types, ...)

## S3 method for class 'types'
keep_types(x, types, invert = FALSE, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`types`	Either an object of the class `types` or a character vector.
`invert`	Logical. Whether the matches should be selected rather than the non-matches.
`...`	Additional arguments.

Details

The methods `keep_pos()` and `drop_pos()` are part of a family of methods of the mclm package used to subset different objects. The methods starting with `keep_` extract the items in `x` based on the criterion specified by the second argument. In contrast, the methods starting with `drop_` *exclude* the items that match the criterion in the same argument.

Calling a `drop_` method is equivalent to calling its `keep_` counterpart when the `invert` argument is `TRUE`.

Value

Object of the same class as `x` with the selected elements only.

See Also

Other subsetters: `brackets`, `keep_bool()`, `keep_pos()`, `keep_re()`

Examples

```
# For a 'freqlist' object ------------------------
(flist <- freqlist("The man and the mouse.", as_text = TRUE))
keep_types(flist, c("man", "and"))
drop_types(flist, c("man", "and"))
keep_types(flist, c("man", "and"), invert = TRUE) # same as drop_types()

# For a 'types' object ---------------------------
(tps <- as_types(letters[1:10]))

keep_types(tps, c("a", "c", "e", "g", "i"))
drop_types(tps,  c("a", "c", "e", "g", "i"))

# For a 'tokens' object --------------------------
(tks <- as_tokens(letters[1:10]))

keep_types(tks, c("a", "c", "e", "g", "i"))
drop_types(tks,  c("a", "c", "e", "g", "i"))
```

mclm_xml_text *Get text from xml node*

Description

Get text from xml node

Usage

```
mclm_xml_text(node, trim = FALSE)
```

Arguments

node	XML node as read with `xml2::read_xml()`.
trim	If `TRUE` will trim leading and trailing spaces.

Value

Character vector: The text value of the (elements of the) node, concatenated with spaces in between.

Examples

```
test_xml <- '
<p>
  <w pos="at">The</w>
  <w pos="nn">example</w>
  <punct>.</punct>
</p>'

test_xml_parsed <- xml2::read_xml(test_xml)

# xml2 output
xml2::xml_text(test_xml_parsed)

# mclm version
mclm_xml_text(test_xml_parsed)
```

merge_conc *Merge concordances*

Description

This function merges multiple objects of class `conc` into one `conc` object.

Usage

```
merge_conc(..., show_warnings = TRUE)
```

Arguments

`...`	Two or more objects of class `conc`.
`show_warnings`	Logical. If `FALSE`, warnings are suppressed.

Value

An object of class `conc`.

Examples

```
(cd_1 <- conc('A first very small corpus.', '\\w+', as_text = TRUE))
as.data.frame(cd_1)

(cd_2 <- conc('A second very small corpus.', '\\w+', as_text = TRUE))
(cd_3 <- conc('A third very small corpus.', '\\w+', as_text = TRUE))
(cd <- merge_conc(cd_1, cd_2, cd_3))
as.data.frame(cd)
```

`merge_fnames` *Merge filenames collections*

Description

These functions merge two or more `fnames` objects into one larger `fnames` object, removing duplicates (keeping only the first appearance) and only resorting the items if `sort = TRUE`.

Usage

```
fnames_merge(x, y, sort = FALSE)

fnames_merge_all(..., sort = FALSE)
```

Arguments

`x, y`	An object of class `fnames`.
`sort`	Boolean value. Should the items in the output be sorted?
`...`	Various objects of class `fnames` or a list of objects of class `fnames`.

Value

An object of class `fnames`.

Examples

```
cwd_fnames <- as_fnames(c("file1.txt", "file2.txt"))
cwd_fnames2 <- as_fnames(c("dir1/file3.txt", "dir1/file4.txt"))
cwd_fnames3 <- as_fnames(c("dir2/file5.txt", "dir2/file6.txt"))
fnames_merge(cwd_fnames, cwd_fnames2)
fnames_merge_all(cwd_fnames, cwd_fnames2, cwd_fnames3)
```

merge_freqlist *Merge frequency lists*

Description

These functions merge two or more frequency lists, adding up the frequencies. In the current implementation, original ranks are lost when merging.

Usage

```
freqlist_merge(x, y)

freqlist_merge_all(...)
```

Arguments

x, y	An object of class freqlist.
...	Various objects of class freqlist or a list of objects of class freqlist.

Value

An object of class freqlist.

Examples

```
(flist1 <- freqlist("A first toy corpus.", as_text = TRUE))
(flist2 <- freqlist("A second toy corpus.", as_text = TRUE))
(flist3 <- freqlist("A third toy corpus.", as_text = TRUE))

freqlist_merge(flist1, flist2)

freqlist_merge_all(flist1, flist2, flist3)
freqlist_merge_all(list(flist1, flist2, flist3)) # same result
```

merge_tokens *Merge* tokens *objects*

Description

tokens_merge() merges two tokens objects x and y into a larger tokens object. tokens_merge_all() merge all the arguments into one tokens object. The result is a concatenation of the tokens, in which the order of the items in the input is preserved.

Usage

```
tokens_merge(x, y)

tokens_merge_all(...)
```

Arguments

`x, y`	An object of class `tokens`
`...`	Objects of class `tokens` or a list with objects of class `tokens`.

Value

An object of class `tokens`.

Examples

```
(tks1 <- tokenize(c("This is a first sentence.")))
(tks2 <- tokenize(c("It is followed by a second one.")))
(tks3 <- tokenize(c("Then a third one follows.")))

tokens_merge(tks1, tks2)
tokens_merge_all(tks1, tks2, tks3)
tokens_merge_all(list(tks1, tks2, tks3))
```

`merge_types` *Merge 'types' objects*

Description

These methods merge two or more objects of class `types`.

Usage

```
types_merge(x, y, sort = FALSE)

types_merge_all(..., sort = FALSE)
```

Arguments

`x, y`	An object of class `types`.
`sort`	Logical. Should the results be sorted.
`...`	Either objects of the class `types` or lists containing such objects.

Value

An object of the class `types`.

Functions

- `types_merge()`: Merge two types
- `types_merge_all()`: Merge multiple types

Examples

```
(tps1 <- as_types(c("a", "simple", "simple", "example")))
(tps2 <- as_types(c("with", "a", "few", "words")))
(tps3 <- as_types(c("just", "for", "testing")))
types_merge(tps1, tps2)       # always removes duplicates, but doesn't sort
sort(types_merge(tps1, tps2)) # same, but with sorting
types_merge_all(tps1, tps2, tps3)
types_merge_all(list(tps1, tps2, tps3))
```

`n_fnames` *Count number of items in an 'fnames' object*

Description

This function counts the number of items, duplicated or not, in an `fnames` object. If there are duplicated items, it will return a warning.

Usage

```
n_fnames(x, ...)
```

Arguments

`x`	Object of class `fnames`.
`...`	Additional arguments.

Value

A number.

Examples

```
cwd_fnames <- as_fnames(c("folder/file1.txt", "folder/file2.txt", "folder/file3.txt"))
n_fnames(cwd_fnames)
```

n_tokens *Count tokens*

Description

This method returns the number of tokens in an object.

Usage

```
n_tokens(x, ...)

## S3 method for class 'freqlist'
n_tokens(x, ...)

## S3 method for class 'tokens'
n_tokens(x, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`...`	Additional arguments.

Value

A number.

See Also

Other getters and setters: `n_types()`, `orig_ranks()`, `ranks()`, `tot_n_tokens()`, `type_names()`

Examples

```
(tks <- tokenize("The old man and the sea."))
n_tokens(tks)

(flist <- freqlist(tks))
n_tokens(flist)
n_types(flist)
```

n_types *Count types*

Description

This method returns the number of types in an object.

Usage

```
n_types(x, ...)

## S3 method for class 'assoc_scores'
n_types(x, ...)

## S3 method for class 'freqlist'
n_types(x, ...)

## S3 method for class 'tokens'
n_types(x, ...)

## S3 method for class 'types'
n_types(x, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`...`	Additional arguments.

Value

A number.

See Also

Other getters and setters: `n_tokens()`, `orig_ranks()`, `ranks()`, `tot_n_tokens()`, `type_names()`

Examples

```
(tks <- tokenize("The old man and the sea."))

# for a types object ----------
(tps <- types(tks))
n_types(tps)

# for a freqlist object -------
(flist <- freqlist(tks))
n_tokens(flist)
n_types(flist)
```

```
# for an assoc_scores object --
a <- c(10,    30,    15,    1)
b <- c(200, 1000,  5000,  300)
c <- c(100,   14,    16,    4)
d <- c(300, 5000, 10000, 6000)
types <- c("four", "fictitious", "toy", "examples")

(scores <- assoc_abcd(a, b, c, d, types = types))
n_types(scores)
```

`orig_ranks` *Retrieve or set original ranks*

Description

These methods retrieve or set, for a the original ranks for the frequency counts of an object. These original ranks are only defined if `x` is the result of a selection procedure (i.e. if `x` contains frequency counts for a selection of items only, and not for all tokens in the corpus).

Usage

```
orig_ranks(x, ...)

orig_ranks(x) <- value

## S3 replacement method for class 'freqlist'
orig_ranks(x) <- value

## S3 method for class 'freqlist'
orig_ranks(x, with_names = FALSE, ...)

## Default S3 replacement method:
orig_ranks(x) <- value
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`...`	Additional arguments.
`value`	Currently it can only be `NULL`.
`with_names`	Logical. Whether or not the items in the output should be given names. If `TRUE`, then the names of the types in the frequency list are used as names.

Value

Either `NULL` or a numeric vector, representing the original ranks, with as its names the types to which these ranks apply.

See Also

Other getters and setters: `n_tokens()`, `n_types()`, `ranks()`, `tot_n_tokens()`, `type_names()`

Examples

```
x <- freqlist("The man and the mouse.",
              as_text = TRUE)
x
orig_ranks(x)
orig_ranks(x, with_names = TRUE)

y <- keep_types(x, c("man", "and"))
orig_ranks(y)
y

orig_ranks(y) <- NULL
y
orig_ranks(y)

tot_n_tokens(y) <- sum(y)
y
```

---

perl_flavor *Retrieve or set the flavor of a regular expression*

Description

These functions retrieve or set the `perl` property of an object of class `re`.

Usage

```
perl_flavor(x)

perl_flavor(x) <- value
```

Arguments

`x`	Object of class `re`.
`value`	Logical.

Details

The assignment function merely sets the `perl` property so that the `x` attribute is read as an expression using the PCRE flavor of regular expression (when `perl = TRUE`) or not (when `perl = FALSE`). The regular expression itself is not modified: if `perl` is set to an inappropriate value, the regular expression will no longer function properly in any of the functions that support `re` objects.

Value

A logical vector of length 1.

Examples

```
(regex <- re("^.{3,}"))
perl_flavor(regex)

perl_flavor(regex) <- FALSE
perl_flavor(regex)
regex

perl_flavor(regex) <- TRUE
perl_flavor(regex)
regex
```

`print.assoc_scores` *Print an object*

Description

This base method prints objects; here the arguments specific to mclm implementations are described.

Usage

```
## S3 method for class 'assoc_scores'
print(
  x,
  n = 20,
  from = 1,
  freeze_cols = NULL,
  keep_cols = NULL,
  drop_cols = NULL,
  from_col = 1,
  sort_order = c("none", "G_signed", "PMI", "alpha"),
  extra = NULL,
  ...
)

## S3 method for class 'conc'
print(x, n = 30, ...)

## S3 method for class 'fnames'
print(
  x,
  n = 20,
  from = 1,
  sort_order = c("none", "alpha"),
  extra = NULL,
  hide_path = NULL,
  ...
)

## S3 method for class 'freqlist'
print(x, n = 20, from = 1, extra = NULL, ...)

## S3 method for class 'slma'
print(x, n = 20, from = 1, ...)

## S3 method for class 'tokens'
print(x, n = 20, from = 1, extra = NULL, ...)

## S3 method for class 'types'
print(x, n = 20, from = 1, sort_order = c("none", "alpha"), extra = NULL, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`n`	Maximum number of items in the object to be printed at once.
`from`	Index of the first item to be printed.
`freeze_cols`	Names of columns that should not be affected by the argument `from_col`. Frozen columns are always printed to the left of non-frozen columns, even if in their original order was different. The names of the types are always and unavoidably printed as the leftmost column. If this argument is `NULL`, then the default setting applies, meaning that the following columns, if present, are displayed in the "frozen area": `a`, `PMI` and `G_signed`. To avoid any columns for being frozen, `freeze_cols` should be `NA` or `character(0)`.
`keep_cols, drop_cols`	A vector of column names or `NULL`. If both arguments are `NULL`, all columns are printed (or as many as fit on the screen). If `keep_cols` is not `NULL`, it indicates the columns that should be printed. If it is `NULL` but `drop_cols` is not, then `drop_cols` indicates the columns that should *not* be printed. Note that they have **no effect** on the frozen area. Columns that are blocked from printing by these arguments are still available to `sort_order`.
`from_col`	Index of the first column to be displayed in the regular area (among all selected columns, including frozen columns). If `from_col` points
`sort_order`	Order in which the items are to be printed. In general, possible values are `"alpha"` (meaning that the items are to be sorted alphabetically), and `"none"` (meaning that the items are not to be sorted). If `x` is an object of class `assoc_scores`, a column name or vector of column names may be provided instead.

`extra`	Extra settings, as an environment. Arguments defined here take precedence over other arguments. For instance, if `extra$from_col` is not `NULL`, it will overrule the `from_col` argument.
`...`	Additional printing arguments.
`hide_path`	A character string with a regular expression or `NULL`. If it is not `NULL`, the character string will be removed from the paths when printing.

Value

Invisibly, `x`. For objects of class `assoc_scores`, the output consists of two areas: the 'frozen area' on the left and the 'regular area' on the right. Both areas are visually separated by a vertical line (`|`). The distinction between them is more intuitive in `explore()`, where the frozen columns do not respond to horizontal movements (with the `r` and `l` commands). The equivalent in this method is the `from_col` argument.

`print_kwic` *Print a concordance in KWIC format*

Description

This function prints a concordance in KWIC format.

Usage

```
print_kwic(
  x,
  min_c_left = NA,
  max_c_left = NA,
  min_c_match = NA,
  max_c_match = NA,
  min_c_right = NA,
  max_c_right = NA,
  from = 1,
  n = 30,
  drop_tags = TRUE
)
```

Arguments

`x`	An object of class `conc`.
`min_c_left, max_c_left`	Minimum and maximum size, expressed in number of characters, of the left co-text in the KWIC display.
`min_c_match, max_c_match`	Minimum and maximum size, expressed in number of characters, of the match in the KWIC display.

`min_c_right, max_c_right`	Minimum and maximum size, expressed in number of characters, of the right co-text in the KWIC display.
`from`	Index of the first item of `x` to be displayed.
`n`	Number of consecutive items in `x` to be displayed.
`drop_tags`	Logical. Should tags be hidden?

Value

Invisibly, `x`.

See Also

print

`p_to_chisq1` *P right quantile in chi-squared distribution with 1 degree of freedom*

Description

P right quantile that takes as its argument a probability `p` and that returns the `p` *right quantile* in the χ^2 distribution with one degree of freedom. In other words, it returns a value q such that a proportion `p` χ^2 distribution with one degree of freedom lies above q.

Usage

```
p_to_chisq1(p)
```

Arguments

`p`	A proportion.

Value

The `p` *right quantile* in the χ^2 distribution with one degree of freedom.

See Also

`chisq1_to_p()`

ranks — *Retrieve the current ranks for frequency counts.*

Description

`ranks` retrieves from the ranks of its items in an object. These ranks are integer values running from one up to the number of items in `x`. Each items receives a unique rank. Items are first ranked by frequency in descending order. Items with identical frequency are further ranked by alphabetic order.

Usage

```
ranks(x, ...)

## S3 method for class 'freqlist'
ranks(x, with_names = FALSE, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`...`	Additional arguments.
`with_names`	Logical. Whether or not the items in the output should be given names. If `TRUE`, then the names of the types in the frequency list are used as names.

Details

The mclm method `ranks()` is not to be confused with `base::rank()`. There are two important differences.

First,`base::rank()` always ranks items from low values to high values and `ranks()` ranks from high frequency items to low frequency items.

Second, `base::rank()` allows the user to choose among a number of different ways to handle ties. In contrast, `ranks()` always handles ties in the same way. More specifically, items with identical frequencies are always ranked in alphabetical order.

In other words, `base::rank()` is a flexible tool that supports a number of different ranking methods that are commonly used in statistics. In contrast, `ranks()` is a rigid tool that supports only one type of ranking, which is a type of ranking that is atypical from a statistics point of view, but is commonly used in linguistic frequency lists. Also, it is designed to be unaffected by the order of the items in the frequency list.

Value

Numeric vector representing the current ranks, with as its names the types to which the ranks apply.

See Also

Other getters and setters: `n_tokens()`, `n_types()`, `orig_ranks()`, `tot_n_tokens()`, `type_names()`

Examples

```
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

orig_ranks(flist)
ranks(flist)
ranks(flist, with_names = TRUE)

(flist2 <- keep_types(flist, c("man", "and")))

orig_ranks(flist2)
ranks(flist2)
```

---

re *Build a regular expression*

Description

Create an object of class `re` or coerce a character vector to an object of class `re`.

Usage

```
re(x, perl = TRUE, ...)

as_re(x, perl = TRUE, ...)

as.re(x, perl = TRUE, ...)
```

Arguments

`x`	Character vector of length one. The value of this character vector is assumed to be a well-formed regular expression. In the current implementation this is assumed, not checked.
`perl`	Logical. If `TRUE`, `x` is assumed to use PCRE (i.e. Perl Compatible Regular Expressions) notation. If `FALSE`, `x` is assumed to use base R's default regular expression notation. Contrary to base R's regular expression functions, `re()` assumes that the PCRE regular expression flavor is used by default.
`...`	Additional arguments.

Details

This class exists because some functions in the mclm package require their arguments to be marked as being regular expressions. For example, `keep_re()` does not need its `pattern` argument to be a `re` object, but if the user wants to subset items with brackets using a regular expression, they must use a `re` object.

Value

An object of class `re`, which is a wrapper around a character vector flagging it as containing a regular expression. In essence it is a named list: the `x` item contains the `x` input and the `perl` item contains the value of the `perl` argument (`TRUE` by default).

It has basic methods such as `print()`, `summary()` and `as.character()`.

See Also

`perl_flavor()`, `scan_re()`, `cat_re()`

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
  It consisted of three sentences. And it lived happily ever after."

(tks <- tokenize(toy_corpus))

# In `keep_re()`, the use of `re()` is optional
keep_re(tks, re("^.{3,}"))
keep_re(tks, "^.{3,}")

# When using brackets notation, `re()` is necessary
tks[re("^.{3,}")]
tks["^.{3,}"]

# build and print a `re` object
re("^.{3,}")
as_re("^.{3,}")
as.re("^.{3,}")
print(re("^.{3,}"))
```

---

read_assoc *Read association scores from file*

Description

This function reads a file written by `write_assoc()`.

Usage

```
read_assoc(file, sep = "\t", file_encoding = "UTF-8", ...)
```

Arguments

`file`	Path of the input file.
`sep`	Field separator in the input file.
`file_encoding`	Encoding of the input file.
`...`	Additional arguments.

Value

An object of class assoc_scores.

See Also

write_assoc()

Other reading functions: read_conc(), read_fnames(), read_freqlist(), read_tokens(), read_txt(), read_types()

Examples

```
txt1 <- "we're just two lost souls swimming in a fish bowl,
year after year, running over the same old ground,
what have we found? the same old fears.
wish you were here."
flist1 <- freqlist(txt1, as_text = TRUE)
txt2 <- "picture yourself in a boat on a river
with tangerine dreams and marmelade skies
somebody calls you, you answer quite slowly
a girl with kaleidoscope eyes"
flist2 <- freqlist(txt2, as_text = TRUE)
(scores <- assoc_scores(flist1, flist2, min_freq = 0))

write_assoc(scores, "example_scores.tab")
(scores2 <- read_assoc("example_scores.tab"))
```

read_conc *Read a concordance from a file*

Description

This function reads concordance-based data frames that are written to file with the function write_conc().

Usage

```
read_conc(
  file,
  sep = "\t",
  file_encoding = "UTF-8",
  stringsAsFactors = FALSE,
  ...
)
```

Arguments

`file`	Name of the input file.
`sep`	Field separator used in the input file.
`file_encoding`	Encoding of the input file.
`stringsAsFactors`	Logical. Whether character data should automatically be converted to factors. It applies to all columns except for `"source"`, `"left"`, `"match"` and `"right"`, which are never converted.
`...`	Additional arguments, not implemented.

Value

Object of class `conc`.

See Also

`import_conc()` for reading files not generated with `write_conc()`.

Other reading functions: `read_assoc()`, `read_fnames()`, `read_freqlist()`, `read_tokens()`, `read_txt()`, `read_types()`

Examples

```
(d <- conc('A very small corpus.', '\\w+', as_text = TRUE))

write_conc(d, "example_data.tab")
(d2 <- read_conc("example_data.tab"))
```

`read_fnames` *Read a collection of filenames from a text file*

Description

This function reads an object of class `fnames` from a text file, which is assumed to contain one filename on each line.

Usage

```
read_fnames(file, sep = NA, file_encoding = "UTF-8", trim_fnames = FALSE, ...)
```

Arguments

`file`	Path to input file.
`sep`	Character vector of length 1 or `NA`. If it is a character, it indicates a separator between input files, in addition to the new line.
`file_encoding`	Encoding used in the input file.
`trim_fnames`	Boolean. Should leading and trailing whitespace be stripped from the filenames?
`...`	Additional arguments (not implemented).

Value

An object of class fnames.

See Also

write_fnames()

Other reading functions: read_assoc(), read_conc(), read_freqlist(), read_tokens(), read_txt(), read_types()

Examples

```
cwd_fnames <- as_fnames(c("file1.txt", "file2.txt"))
write_fnames(cwd_fnames, "file_with_filenames.txt")
cwd_fnames_2 <- read_fnames("file_with_filenames.txt")
```

read_freqlist *Read a frequency list from a csv file*

Description

This function reads an object of the class freqlist from a csv file. The csv file is assumed to contain two columns, the first being the type and the second being the frequency of that type. The file is also assumed to have a header line with the names of both columns.

Usage

```
read_freqlist(file, sep = "\t", file_encoding = "UTF-8", ...)
```

Arguments

file	Character vector of length 1. Path to the input file.
sep	Character vector of length 1. Column separator.
file_encoding	File encoding used in the input file.
...	Additional arguments (not implemented).

Details

read_freqlist not only reads the file file, but also checks whether a configuration file exists with a name that is identical to file, except that it has the filename extension ".yaml".

If such a file exists, then that configuration file is taken to 'belong' to file and is also read and the frequency list attributes "tot_n_tokens" and "tot_n_types" are retrieved from it.

If no such configuration file exists, then the values for "tot_n_tokens" and "tot_n_types" are calculated on the basis of the frequencies in the frequency list.

Value

Object of class freqlist.

See Also

write_freqlist()

Other reading functions: read_assoc(), read_conc(), read_fnames(), read_tokens(), read_txt(), read_types()

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."
freqs <- freqlist(toy_corpus, as_text = TRUE)

print(freqs, n = 1000)

write_freqlist(freqs, "example_freqlist.csv")
freqs2 <- read_freqlist("example_freqlist.csv")
print(freqs2, n = 1000)
```

`read_tokens` *Read a* `tokens` *object from a text file*

Description

This function reads an object of the class tokens from a text file, typically stored with write_tokens(). The text file is assumed to contain one token on each line and not to have a header.

Usage

```
read_tokens(file, file_encoding = "UTF-8", ...)
```

Arguments

file	Name of the input file.
file_encoding	Encoding to read the input file.
...	Additional arguments (not implemented).

Value

An object of class tokens.

See Also

write_tokens()

Other reading functions: read_assoc(), read_conc(), read_fnames(), read_freqlist(), read_txt(), read_types()

Examples

```
(tks <- tokenize("The old man and the sea."))
write_tokens(tks, "file_with_tokens.txt")
(tks2 <- read_tokens("file_with_tokens.txt"))
```

---

read_txt *Read a text file into a character vector*

Description

This function reads a text file and returns a character vector containing the lines in the text file.

Usage

```
read_txt(file, file_encoding = "UTF-8", line_glue = NA, ...)
```

Arguments

file	Name of the input file.
file_encoding	Encoding of the input file.
line_glue	A character vector or NA. If NA, the output is a character vector in which each input line is a separate item, as in readr::read_lines(). Otherwise, the output is a character vector of length 1 in which all input lines are concatenated, using the value of line_glue[1] as line separator and as end-of-last-line marker.
...	Additional arguments (not implemented).

Value

A character vector.

See Also

write_txt()

Other reading functions: read_assoc(), read_conc(), read_fnames(), read_freqlist(), read_tokens(), read_types()

Examples

```
x <- "This is
a small
text."

# write the text to a text file
write_txt(x, "example-text-file.txt")
# read a text from file
y <- read_txt("example-text-file.txt")
y
```

`read_types` *Read a vector of types from a text file*

Description

This function read an object of the class `types` from a text file. By default, the text file is assumed to contain one type on each line.

Usage

```
read_types(
  file,
  sep = NA,
  file_encoding = "UTF-8",
  trim_types = FALSE,
  remove_duplicates = FALSE,
  sort = FALSE,
  ...
)
```

Arguments

`file`	Name of the input file.
`sep`	If not `is.na(sep)`, then `sep` must be a character vector of length one. In that case, `sep` is interpreted as a type separator in the input file. This separator the serves as an additional type separator, next to the end of each line. The end of a line always indicated a separator between types (in other words, types cannot cross lines).
`file_encoding`	The file encoding used in the input file.
`trim_types`	Logical. Should leading and trailing white space should be stripped from the types.
`remove_duplicates`	Logical. Should duplicates be removed from `x` prior to coercing to a vector of types.

`sort`	Logical. Should x be alphabetically sorted prior to coercing to a vector of types; this argument is ignored if `remove_duplicates` is `TRUE`, because the result of removing duplicates is always sorted.
`...`	Additional arguments (not implemented).

Value

Object of class `types`.

See Also

`write_types()`

Other reading functions: `read_assoc()`, `read_conc()`, `read_fnames()`, `read_freqlist()`, `read_tokens()`, `read_txt()`

Examples

```
types <- as_types(c("first", "second", "third"))
write_types(types, "file_with_types.txt")
types_2 <- read_types("file_with_types.txt")
```

`re_convenience` *Convenience functions in support of regular expressions*

Description

These functions are essentially simple wrappers around base R functions such as `regexpr()`, `gregexpr()`, `grepl()`, `grep()`, `sub()` and `gsub()`. The most important differences between the functions documented here and the R base functions is the order of the arguments (`x` before `pattern`) and the fact that the argument `perl` is set to `TRUE` by default.

Usage

```
re_retrieve_first(
  x,
  pattern,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  requested_group = NULL,
  drop_NA = FALSE,
  ...
)
```

```
re_retrieve_last(
  x,
  pattern,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  requested_group = NULL,
  drop_NA = FALSE,
  ...
)

re_retrieve_all(
  x,
  pattern,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  requested_group = NULL,
  unlist = TRUE,
  ...
)

re_has_matches(
  x,
  pattern,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  ...
)

re_which(
  x,
  pattern,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  ...
)

re_replace_first(
  x,
  pattern,
  replacement,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  ...
)

re_replace_all(
  x,
  pattern,
  replacement,
  ignore.case = FALSE,
  perl = TRUE,
  fixed = FALSE,
  useBytes = FALSE,
  ...
)
```

Arguments

`x`	Character vector to be searched or modified.
`pattern`	Regular expression specifying what is to be searched.
`ignore.case`	Logical. Should the search be case insensitive?
`perl`	Logical. Whether the regular expressions use the PCRE flavor of regular expression. Unlike in base R functions, the default is `TRUE`.
`fixed`	Logical. If `TRUE`, `pattern` is a string to be matched as is, i.e. wildcards and special characters are not interpreted.
`useBytes`	Logical. If `TRUE` the matching is done byte-by-byte rather than character-by-character. See 'Details' in `grep()`.
`requested_group`	Numeric. If `NULL` or `0`, the output will contain matches for `pattern` as a whole. If another number `n` is provided, then the output will not contain matches for `pattern` but instead will only contain the matches for the nth capturing group in `pattern` (the first if `requested_group = 1`, the second if `requested_group = 2`...).
`drop_NA`	Logical. If `FALSE`, the output always has the same length as the input `x` and items that do not contain a match for `pattern` yield `NA`. If `TRUE`, such `NA` values are removed and therefore the result might contain fewer items than `x`.
`...`	Additional arguments.
`unlist`	Logical. If `FALSE`, the output always has the same length as the input `x`. More specifically, the result will be a list in which input items that do not contain a match for `pattern` yield an empty vector, whereas input items that do match will yield a vector of at least length one (depending on the number of matches). If `TRUE`, the output is a single vector the length of which may be shorter or longer than `x`.

`replacement`	Character vector of length one specifying the replacement string. It is to be taken literally, except that the notation `\\1`, `\\2`, etc. can be used to refer to groups in `pattern`.

Details

For some of the arguments (e.g. `perl`, `fixed`) the reader is directed to base R's regex documentation.

Value

`re_retrieve_first()`, `re_retrieve_last()` and `re_retrieve_all()` return either a single vector of character data or a list containing such vectors. `re_replace_first()` and `re_replace_all()` return the same type of character vector as `x`.

`re_has_matches()` returns a logical vector indicating whether a match was found in each of the elements in `x`; `re_which()` returns a numeric vector indicating the indices of the elements of `x` for which a match was found.

Functions

- `re_retrieve_first()`: Retrieve from each item in `x` the first match of `pattern`.
- `re_retrieve_last()`: Retrieve from each item in `x` the last match of `pattern`.
- `re_retrieve_all()`: Retrieve from each item in `x` all matches of `pattern`.
- `re_has_matches()`: Simple wrapper around `grepl()`.
- `re_which()`: Simple wrapper around `grep()`.
- `re_replace_first()`: Simple wrapper around `sub()`.
- `re_replace_all()`: Simple wrapper around `gsub()`.

Examples

```
x <- tokenize("This is a sentence with a couple of words in it.")
pattern <- "[oe](.)(.)"

re_retrieve_first(x, pattern)
re_retrieve_first(x, pattern, drop_NA = TRUE)
re_retrieve_first(x, pattern, requested_group = 1)
re_retrieve_first(x, pattern, drop_NA = TRUE, requested_group = 1)
re_retrieve_first(x, pattern, requested_group = 2)

re_retrieve_last(x, pattern)
re_retrieve_last(x, pattern, drop_NA = TRUE)
re_retrieve_last(x, pattern, requested_group = 1)
re_retrieve_last(x, pattern, drop_NA = TRUE, requested_group = 1)
re_retrieve_last(x, pattern, requested_group = 2)

re_retrieve_all(x, pattern)
re_retrieve_all(x, pattern, unlist = FALSE)
re_retrieve_all(x, pattern, requested_group = 1)
re_retrieve_all(x, pattern, unlist = FALSE, requested_group = 1)
re_retrieve_all(x, pattern, requested_group = 2)

re_replace_first(x, "([oe].)", "{\\1}")
re_replace_all(x, "([oe].)", "{\\1}")
```

scan_re *Scan a regular expression from console*

Description

The functions `scan_re()` and `scan_re2()` can be used to scan a regular expression from the console.

Usage

```
scan_re(perl = TRUE, ...)

scan_re2(perl = TRUE, ...)
```

Arguments

`perl`	Logical. If `TRUE`, the regular expression being scanned is assumed to use PCRE (Perl Compatible Regular Expressions) notation. If `FALSE`, it is assumed to use base R's default regular expression notation (see base::regex). Contrary to base R's regular expression functions, the default is `TRUE`.
`...`	Additional arguments.

Details

After the function call, R will continue scanning your input until it encounters an empty input line, i.e. until it encounters two consecutive newline symbols (or until it encounters a line with nothing but whitespace characters). In other words, press ENTER twice in a row if you want to stop inputting characters. The function will then return your input as a character vector of length one.

These functions are designed to allow you to input complex text, in particular regular expressions, without dealing with the restrictions of string literals, such as having to use `\\` for `\`.

Value

An object of class `re`.

See Also

`scan_txt()`, `cat_re()`

`scan_txt` *Scan a character string from console*

Description

The functions `scan_txt()` and `scan_txt2()`, which take no arguments, can be used to scan a text string from the console.

Usage

```
scan_txt()

scan_txt2()
```

Details

After the function call, R will continue scanning your input until it encounters an empty input line, i.e. until it encounters two consecutive newline symbols (or until it encounters a line with nothing but whitespace characters). In other words, press ENTER twice in a row if you want to stop inputting characters. The function will then return your input as a character vector of length one.

These functions are designed to allow you to input complex text, in particular regular expressions, without dealing with the restrictions of string literals, such as having to use `\\` for `\`.

Value

A character vector of length one that contains the string that has been scanned from the console.

See Also

`scan_re()`

`short_names` *Shorten filenames*

Description

Helper functions that make the paths to a file shorter.

Usage

```
drop_path(x, ...)

drop_extension(x, ...)

short_names(x, ...)
```

Arguments

x	An object of class `fnames` or a character vector.
...	Additional arguments.

Value

An object of the same class as `x`.

Functions

- `drop_path()`: Extract the base name of a path, removing the paths leading to it.
- `drop_extension()`: Remove extension from a filename.
- `short_names()`: Remove both paths leading to a file and its extension.

Examples

```
cwd_fnames <- as_fnames(c("folder/file1.txt", "folder/file2.txt", "folder/file3.txt"))
drop_path(cwd_fnames)
drop_extension(cwd_fnames)
short_names(cwd_fnames) # same as drop_path(drop_extension(cwd_fnames))
```

`slma` *Stable lexical marker analysis*

Description

This function conducts a stable lexical marker analysis.

Usage

```
slma(
  x,
  y,
  file_encoding = "UTF-8",
  sig_cutoff = qchisq(0.95, df = 1),
  small_pos = 1e-05,
  keep_intermediate = FALSE,
  verbose = TRUE,
  min_rank = 1,
  max_rank = 5000,
  keeplist = NULL,
  stoplist = NULL,
  ngram_size = NULL,
  max_skip = 0,
  ngram_sep = "_",
  ngram_n_open = 0,
  ngram_open = "[]",
  ...
)
```

Arguments

x, y	Character vector or fnames object with filenames for the two sets of documents.
file_encoding	Encoding of all the files to read.
sig_cutoff	Numeric value indicating the cutoff value for 'significance in the stable lexical marker analysis. The default value is qchist(.95, df = 1), which is about 3.84.
small_pos	Alternative (but sometimes inferior) approach to dealing with zero frequencies, compared to haldane. The argument small_pos only applies when haldane is set to FALSE. (See the Details section.) If haldane is FALSE, and there is at least one zero frequency in a contingency table, adding small positive values to the zero frequency cells is done systematically for all measures calculated for that table, not just for measures that need this to be done.
keep_intermediate	Logical. If TRUE, results from intermediate calculations are kept in the output as the "intermediate" element. This is necessary if you want to inspect the object with the details() method.
verbose	Logical. Whether progress should be printed to the console during analysis.
min_rank, max_rank	Minimum and maximum frequency rank in the first corpus (x) of the items to take into consideration as candidate stable markers. Only tokens or token n-grams with a frequency rank greater than or equal to min_rank and lower than or equal to max_rank will be included.
keeplist	List of types that must certainly be included in the list of candidate markers regardless of their frequency rank and of stoplist.
stoplist	List of types that must not be included in the list of candidate markers, although, if a type is included in keeplist, its inclusion in stoplist is disregarded.
ngram_size	Argument in support of ngrams/skipgrams (see also max_skip). If one wants to identify individual tokens, the value of ngram_size should be NULL or 1. If one wants to retrieve token ngrams/skipgrams, ngram_size should be an integer indicating the size of the ngrams/skipgrams. E.g. 2 for bigrams, or 3 for trigrams, etc.
max_skip	Argument in support of skipgrams. This argument is ignored if ngram_size is NULL or is 1. If ngram_size is 2 or higher, and max_skip is 0, then regular ngrams are being retrieved (albeit that they may contain open slots; see ngram_n_open). If ngram_size is 2 or higher, and max_skip is 1 or higher, then skipgrams are being retrieved (which in the current implementation cannot contain open slots; see ngram_n_open). For instance, if ngram_size is 3 and max_skip is 2, then 2-skip trigrams are being retrieved. Or if ngram_size is 5 and max_skip is 3, then 3-skip 5-grams are being retrieved.

`ngram_sep`	Character vector of length 1 containing the string that is used to separate/link tokens in the representation of ngrams/skipgrams in the output of this function.
`ngram_n_open`	If `ngram_size` is `2` or higher, and moreover `ngram_n_open` is a number higher than `0`, then ngrams with 'open slots' in them are retrieved. These ngrams with 'open slots' are generalizations of fully lexically specific ngrams (with the generalization being that one or more of the items in the ngram are replaced by a notation that stands for 'any arbitrary token'). For instance, if `ngram_size` is `4` and `ngram_n_open` is `1`, and if moreover the input contains a 4-gram `"it_is_widely_accepted"`, then the output will contain all modifications of `"it_is_widely_accepted"` in which one (since `ngram_n_open` is `1`) of the items in this n-gram is replaced by an open slot. The first and the last item inside an ngram are never turned into an open slot; only the items in between are candidates for being turned into open slots. Therefore, in the example, the output will contain `"it_[]_widely_accepted"` and `"it_is_[]_accepted"`. As a second example, if `ngram_size` is `5` and `ngram_n_open` is `2`, and if moreover the input contains a 5-gram `"it_is_widely_accepted_that"`, then the output will contain `"it_[]_[]_accepted_that"`, `"it_[]_widely_[]_that"`, and `"it_is_[]_[]_that"`.
`ngram_open`	Character string used to represent open slots in ngrams in the output of this function.
`...`	Additional arguments.

Details

A stable lexical marker analysis of the *A*-documents in `x` versus the *B*-documents in `y` starts from a separate keyword analysis for all possible document couples (a, b), with *a* an *A*-document and *b* a *B*-document. If there are *n* *A*-documents and *m* *B*-documents, then $n * m$ keyword analyses are conducted. The 'stability' of a linguistic item *x*, as a marker for the collection of *A*-documents (when compared to the *B*-documents) corresponds to the frequency and consistency with which *x* is found to be a keyword for the *A*-documents across all aforementioned keyword analyses.

In any specific keyword analysis, *x* is considered a keyword for an *A*-document if `G_signed` is positive and moreover `p_G` is less than `sig_cutoff` (see `assoc_scores()` for more information on the measures). Item *x* is considered a keyword for the *B*-document if `G_signed` is negative and moreover `p_G` is less than `sig_cutoff`.

Value

An object of class `slma`, which is a named list with at least the following elements:

- A `scores` dataframe with information about the stability of the chosen lexical items. (See below.)
- An `intermediate` list with a register of intermediate values if `keep_intermediate` was `TRUE`.
- Named items registering the values of the arguments with the same name, namely `sig_cutoff`, `small_pos`, `x`, and `y`.

The `slma` object has `as_data_frame()` and `print` methods as well as an ad-hoc `details()` method. Note that the `print` method simply prints the main dataframe.

Contents of the `scores` element:

The `scores` element is a dataframe of which the rows are linguistic items for which a stable lexical marker analysis was conducted and the columns are different 'stability measures' and related statistics. By default, the linguistic items are sorted by decreasing 'stability' according to the `S_lor` measure.

Column	Name	Computation
`S_abs`	Absolute stability	`S_att` - `S_rep`
`S_nrm`	Normalized stability	`S_abs` / $n * m$
`S_att`	Stability of attraction	Number of (a, b) couples in which the linguistic item is a keyword for the *A*-document
`S_rep`	Stability of repulsion	Number of (a, b) couples in which the linguistic item is a keyword for the *B*-document
`S_lor`	Log of odds ratio stability	Mean of `log_OR` across all (a, b) couples but setting to 0 the value when `p_G` is larger t

`S_lor` is then computed as a fraction with as its numerator the sum of all `log_OR` values across all (a, b) couples for which `p_G` is lower than `sig_cutoff` and as its denominator $n * m$. For more on `log_OR`, see the Value section on on `assoc_scores()`. The final three columns on the output are meant as a tool in support of the interpretation of the `log_OR` column. Considering all (a, b) couples for which `p_G` is smaller than `sig_cutoff`, `lor_min`, `lor_max` and `lor_sd` are their minimum, maximum and standard deviation for each element.

Examples

```
a_corp <- get_fnames(system.file("extdata", "cleveland", package = "mclm"))
b_corp <- get_fnames(system.file("extdata", "roosevelt", package = "mclm"))
slma_ex <- slma(a_corp, b_corp)
```

`sort.assoc_scores` *Sort an 'assoc_scores' object*

Description

Sort a full object of class `assoc_scores` based on some criterion. It's the same that `print` does but with a bit more flexibility.

Usage

```
## S3 method for class 'assoc_scores'
sort(x, decreasing = TRUE, sort_order = "none", ...)
```

Arguments

`x`	Object of class `assoc_scores`.
`decreasing`	Boolean value. If `sort_order = "alpha"` and `decreasing = FALSE`, the rows will follow the alphabetic order of the types. If `decreasing = TRUE` instead, it will follow an inverted alphabetic order (from Z to A). This follows the behavior of applying

	`sort()` to a character vector: note that the default value is probably not what you would want. If `sort_order` is a column for which a *lower* value indicates a higher association, i.e. it's a form of p-value, `decreasing` = `TRUE` will place lower values on top and higher values at the bottom. For any other column, `decreasing` = `TRUE` will place higher values on top and lower values at the bottom.
`sort_order`	Criterion to order the rows. Possible values are `"alpha"` (meaning that the items are to be sorted alphabetically), `"none"` (meaning that the items are not to be sorted) and any present column name.
`...`	Additional arguments.

Value

An object of class `assoc_scores`.

Examples

```
a <- c(10,    30,    15,    1)
b <- c(200, 1000,  5000,  300)
c <- c(100,   14,    16,    4)
d <- c(300, 5000, 10000, 6000)
types <- c("four", "fictitious", "toy", "examples")
(scores <- assoc_abcd(a, b, c, d, types = types))

print(scores, sort_order = "PMI")
sorted_scores <- sort(scores, sort_order = "PMI")
sorted_scores

sort(scores, decreasing = FALSE, sort_order = "PMI")
```

`sort.freqlist` *Sort a frequency list*

Description

This method sorts an object of class `freqlist`.

Usage

```
## S3 method for class 'freqlist'
sort(
  x,
  decreasing = FALSE,
  sort_crit = c("ranks", "names", "orig_ranks", "freqs"),
  na_last = TRUE,
  ...
)
```

Arguments

`x`	Object of class `freqlist`.
`decreasing`	Logical. If `TRUE` items are sorted from large to small; if `FALSE`, from small to large. Note, however, that ranking in frequency lists is such that lower ranks correspond to higher frequencies. Therefore, sorting by rank (either `"ranks"` or `"orig_ranks"`) with `decreasing` set to its default value `FALSE` results in the highest frequencies ending up at the beginning of the sorted list.
`sort_crit`	Character string determining the sorting criterion. If `sort_crit` is `"ranks"`, then the items in the frequency list are sorted by their current frequency rank. If `sort_crit` is `"names"`, then the items in the frequency list are sorted alphabetically their name. If `sort_crit` is `"orig_ranks"`, then the items in the frequency list are sorted by their original ranks (if those are present), or by their current frequency ranks (if no original ranks are present). Finally, sorting with `sort_crit` set to `"freqs"` is identical to sorting by frequency ranks, but with the meaning of the argument `decreasing` being reversed. In other words, sorting by frequencies (`"freqs"`) with `decreasing` set to its default value `FALSE` results in the lowest frequencies ending up at the beginning of the sorted list.
`na_last`	Logical defining the behavior of `NA` elements. This argument is only relevant when `sort_crit` is `"orig_ranks"` because currently names and frequencies are not allowed to be `NA` in frequency lists. If `na_last` is `TRUE`, then items with a sorting criterion of `NA` end up at the end of the sorted frequency list. If `na_last` is `FALSE`, then items with a sorting criterion of `NA` end up at the start of the sorted frequency list. If `na_last` is `NA`, then items with a sorting criterion of `NA` are removed from the sorted frequency list.
`...`	Additional arguments.

Details

Because of the way ranks are calculated for ties (with lower ranks being assigned to ties earlier in the list), sorting the list may affect the ranks of ties. More specifically, ranks among ties may differ depending on the criterion that is used to sort the frequency list.

Value

Object of class `freqlist`.

Examples

```
(flist <- freqlist(tokenize("the old story of the old man and the sea.")))
sort(flist)
sort(flist, decreasing = TRUE)
```

tokens *Create or coerce an object into class* tokens

Description

tokenize() splits a text into a sequence of tokens, using regular expressions to identify them, and returns an object of the class tokens.

Usage

```
tokenize(
  x,
  re_drop_line = NULL,
  line_glue = NULL,
  re_cut_area = NULL,
  re_token_splitter = re("[^_\\p{L}\\p{N}\\p{M}'-]+"),
  re_token_extractor = re("[_\\p{L}\\p{N}\\p{M}'-]+"),
  re_drop_token = NULL,
  re_token_transf_in = NULL,
  token_transf_out = NULL,
  token_to_lower = TRUE,
  perl = TRUE,
  ngram_size = NULL,
  max_skip = 0,
  ngram_sep = "_",
  ngram_n_open = 0,
  ngram_open = "[]"
)
```

Arguments

x	Either a character vector or an object of class NLP::TextDocument that contains the text to be tokenized.
re_drop_line	NULL or character vector. If NULL, it is ignored. Otherwise, a character vector (assumed to be of length 1) containing a regular expression. Lines in x that contain a match for re_drop_line are treated as not belonging to the corpus and are excluded from the results.
line_glue	NULL or character vector. If NULL, it is ignored. Otherwise, all lines in a corpus file (or in x, if as_text is TRUE), are glued together in one character vector of length 1, with the string line_glue pasted in between consecutive lines. The value of line_glue can also be equal to the empty string "". The 'line glue' operation is conducted immediately after the 'drop line' operation.
re_cut_area	NULL or character vector. If NULL, it is ignored. Otherwise, all matches in a corpus file (or in x, if as_text is TRUE), are 'cut out' of the text prior to the identification of the tokens in the text (and are therefore not taken into account when identifying the tokens). The 'cut area' operation is conducted immediately after the 'line glue' operation.

`re_token_splitter`	Regular expression or `NULL`. Regular expression that identifies the locations where lines in the corpus files are split into tokens. (See Details.) The 'token identification' operation is conducted immediately after the 'cut area' operation.
`re_token_extractor`	Regular expression that identifies the locations of the actual tokens. This argument is only used if `re_token_splitter` is `NULL`. (See Details.) The 'token identification' operation is conducted immediately after the 'cut area' operation.
`re_drop_token`	Regular expression or `NULL`. If `NULL`, it is ignored. Otherwise, it identifies tokens that are to be excluded from the results. Any token that contains a match for `re_drop_token` is removed from the results. The 'drop token' operation is conducted immediately after the 'token identification' operation.
`re_token_transf_in`	Regular expression that identifies areas in the tokens that are to be transformed. This argument works together with the argument `token_transf_out`. If both `re_token_transf_in` and `token_transf_out` differ from `NA`, then all matches, in the tokens, for the regular expression `re_token_transf_in` are replaced with the replacement string `token_transf_out`. The 'token transformation' operation is conducted immediately after the 'drop token' operation.
`token_transf_out`	Replacement string. This argument works together with `re_token_transf_in` and is ignored if `re_token_transf_in` is `NULL` or `NA`.
`token_to_lower`	Logical. Whether tokens must be converted to lowercase before returning the result. The 'token to lower' operation is conducted immediately after the 'token transformation' operation.
`perl`	Logical. Whether the PCRE regular expression flavor is being used in the arguments that contain regular expressions.
`ngram_size`	Argument in support of ngrams/skipgrams (see also `max_skip`). If one wants to identify individual tokens, the value of `ngram_size` should be `NULL` or `1`. If one wants to retrieve token ngrams/skipgrams, `ngram_size` should be an integer indicating the size of the ngrams/skipgrams. E.g. `2` for bigrams, or `3` for trigrams, etc.
`max_skip`	Argument in support of skipgrams. This argument is ignored if `ngram_size` is `NULL` or is `1`. If `ngram_size` is `2` or higher, and `max_skip` is `0`, then regular ngrams are being retrieved (albeit that they may contain open slots; see `ngram_n_open`). If `ngram_size` is `2` or higher, and `max_skip` is `1` or higher, then skipgrams are being retrieved (which in the current implementation cannot contain open slots; see `ngram_n_open`). For instance, if `ngram_size` is `3` and `max_skip` is `2`, then 2-skip trigrams are being retrieved. Or if `ngram_size` is `5` and `max_skip` is `3`, then 3-skip 5-grams are being retrieved.

`ngram_sep`	Character vector of length 1 containing the string that is used to separate/link tokens in the representation of ngrams/skipgrams in the output of this function.
`ngram_n_open`	If `ngram_size` is `2` or higher, and moreover `ngram_n_open` is a number higher than `0`, then ngrams with 'open slots' in them are retrieved. These ngrams with 'open slots' are generalizations of fully lexically specific ngrams (with the generalization being that one or more of the items in the ngram are replaced by a notation that stands for 'any arbitrary token'). For instance, if `ngram_size` is `4` and `ngram_n_open` is `1`, and if moreover the input contains a 4-gram `"it_is_widely_accepted"`, then the output will contain all modifications of `"it_is_widely_accepted"` in which one (since `ngram_n_open` is `1`) of the items in this n-gram is replaced by an open slot. The first and the last item inside an ngram are never turned into an open slot; only the items in between are candidates for being turned into open slots. Therefore, in the example, the output will contain `"it_[]_widely_accepted"` and `"it_is_[]_accepted"`. As a second example, if `ngram_size` is `5` and `ngram_n_open` is `2`, and if moreover the input contains a 5-gram `"it_is_widely_accepted_that"`, then the output will contain `"it_[]_[]_accepted_that"`, `"it_[]_widely_[]_that"`, and `"it_is_[]_[]_that"`.
`ngram_open`	Character string used to represent open slots in ngrams in the output of this function.

Details

If the output contains ngrams with open slots, then the order of the items in the output is no longer meaningful. For instance, let's imagine a case where `ngram_size` is `5` and `ngram_n_open` is `2`. If the input contains a 5-gram `"it_is_widely_accepted_that"`, then the output will contain `"it_[]_[]_accepted_that"`, `"it_[]_widely_[]_that"` and `"it_is_[]_[]_that"`. The relative order of these three items in the output must be considered arbitrary.

Value

An object of class `tokens`, i.e. a sequence of tokens. It has a number of attributes and method such as:

- base `print`, `as_data_frame()`, `summary()` (which returns the number of items), `sort()` and `rev()`,
- `tibble::as_tibble()`,
- an interactive `explore()` method,
- some getters, namely `n_tokens()` and `n_types()`,
- subsetting methods such as `keep_types()`, `keep_pos()`, etc. including `[]` subsetting (see brackets).

Additional manipulation functions include the `trunc_at()` method to ??, `tokens_merge()` and `tokens_merge_all()` to combine token lists and an `as_character()` method to convert to a character vector.

Objects of class `tokens` can be saved to file with `write_tokens()`; these files can be read with `read_freqlist()`.

See Also

`as_tokens()`

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."

tks <- tokenize(toy_corpus)
print(tks, n = 1000)

tks <- tokenize(toy_corpus, re_token_splitter = "\\W+")
print(tks, n = 1000)
sort(tks)
summary(tks)

tokenize(toy_corpus, ngram_size = 3)

tokenize(toy_corpus, ngram_size = 3, max_skip = 2)

tokenize(toy_corpus, ngram_size = 3, ngram_n_open = 1)
```

`tot_n_tokens` *Retrieve or set the total number of tokens*

Description

These methods retrieve or set the total number of tokens in the corpus on which the frequency counts are based. This total number of tokens may be higher than the sum of all frequency counts in `x`, for instance, if `x` contains frequency counts for a selection of items only, and not for all tokens in the corpus.

Usage

```
tot_n_tokens(x)

tot_n_tokens(x) <- value

## S3 replacement method for class 'freqlist'
tot_n_tokens(x) <- value

## S3 method for class 'freqlist'
tot_n_tokens(x)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`value`	Numerical value.

Value

A number.

See Also

Other getters and setters: `n_tokens()`, `n_types()`, `orig_ranks()`, `ranks()`, `type_names()`

Examples

```
x <- freqlist("The man and the mouse.",
              re_token_splitter = "(?xi) [:\\s.;,?!\"]+",
              as_text = TRUE)
x
tot_n_tokens(x)

y <- keep_types(x, c("man", "and"))
tot_n_tokens(y)
y

tot_n_tokens(y) <- sum(y)
y
tot_n_tokens(y)
```

`trunc_at` *Truncate a sequence of character data*

Description

This method takes as its argument `x` an object that represents a sequence of character data, such as an object of class `tokens`, and truncates it at the position where a match for the argument `pattern` is found. Currently it is only implemented for `tokens` objects.

Usage

```
trunc_at(x, pattern, ...)

## S3 method for class 'tokens'
trunc_at(
  x,
  pattern,
  keep_this = FALSE,
  last_match = FALSE,
  from_end = FALSE,
  ...
)
```

Arguments

`x`	An object that represents a sequence of character data.
`pattern`	A regular expression.
`...`	Additional arguments.
`keep_this`	Logical. Whether the matching token itself should be kept. If `TRUE`, the truncating happens right after the matching token; if `FALSE`, right before.
`last_match`	Logical. In case there are several matching tokens, if `last_match` is `TRUE`, the last match will be used as truncating point; otherwise, the first match will.
`from_end`	Logical. If `FALSE`, the match starts from the first token progressing forward; if `TRUE`, it starts from the last token progressing backward. If `from_end` is `FALSE`, the part of `x` that is kept after truncation is the head of `x`. If it is `TRUE` instead, the part that is kept after truncation is the tail of `x`.

Value

A truncated version of `x`.

Examples

```
(toks <- tokenize('This is a first sentence . This is a second sentence .',
re_token_splitter = '\\s+'))

trunc_at(toks, re("[.]"))

trunc_at(toks, re("[.]"), last_match = TRUE)

trunc_at(toks, re("[.]"), last_match = TRUE, from_end = TRUE)
```

`types` *Build a 'types' object*

Description

This function builds an object of the class `types`.

Usage

```
types(
  x,
  re_drop_line = NULL,
  line_glue = NULL,
  re_cut_area = NULL,
  re_token_splitter = re("[^_\\p{L}\\p{N}\\p{M}'-]+"),
  re_token_extractor = re("[_\\p{L}\\p{N}\\p{M}'-]+"),
  re_drop_token = NULL,
  re_token_transf_in = NULL,
  token_transf_out = NULL,
  token_to_lower = TRUE,
  perl = TRUE,
  blocksize = 300,
  verbose = FALSE,
  show_dots = FALSE,
  dot_blocksize = 10,
  file_encoding = "UTF-8",
  ngram_size = NULL,
  ngram_sep = "_",
  ngram_n_open = 0,
  ngram_open = "[]",
  as_text = FALSE
)
```

Arguments

`x` Either a list of filenames of the corpus files (if `as_text` is `TRUE`) or the actual text of the corpus (if `as_text` is `FALSE`).

If `as_text` is `TRUE` and the length of the vector `x` is higher than one, then each item in `x` is treated as a separate line (or a separate series of lines) in the corpus text. Within each item of `x`, the character `"\\n"` is also treated as a line separator.

`re_drop_line` `NULL` or character vector. If `NULL`, it is ignored. Otherwise, a character vector (assumed to be of length 1) containing a regular expression. Lines in `x` that contain a match for `re_drop_line` are treated as not belonging to the corpus and are excluded from the results.

`line_glue` `NULL` or character vector. If `NULL`, it is ignored. Otherwise, all lines in a corpus file (or in `x`, if `as_text` is `TRUE`), are glued together in one character vector of length 1, with the string `line_glue` pasted in between consecutive lines. The value of `line_glue` can also be equal to the empty string `""`. The 'line glue' operation is conducted immediately after the 'drop line' operation.

`re_cut_area` `NULL` or character vector. If `NULL`, it is ignored. Otherwise, all matches in a corpus file (or in `x`, if `as_text` is `TRUE`), are 'cut out' of the text prior to the identification of the tokens in the text (and are therefore not taken into account when identifying the tokens). The 'cut area' operation is conducted immediately after the 'line glue' operation.

`re_token_splitter`
Regular expression or `NULL`. Regular expression that identifies the locations where lines in the corpus files are split into tokens. (See Details.)

The 'token identification' operation is conducted immediately after the 'cut area' operation.

`re_token_extractor`
Regular expression that identifies the locations of the actual tokens. This argument is only used if `re_token_splitter` is `NULL`. (See Details.)

The 'token identification' operation is conducted immediately after the 'cut area' operation.

`re_drop_token`	Regular expression or `NULL`. If `NULL`, it is ignored. Otherwise, it identifies tokens that are to be excluded from the results. Any token that contains a match for `re_drop_token` is removed from the results. The 'drop token' operation is conducted immediately after the 'token identification' operation.
`re_token_transf_in`	Regular expression that identifies areas in the tokens that are to be transformed. This argument works together with the argument `token_transf_out`. If both `re_token_transf_in` and `token_transf_out` differ from `NA`, then all matches, in the tokens, for the regular expression `re_token_transf_in` are replaced with the replacement string `token_transf_out`. The 'token transformation' operation is conducted immediately after the 'drop token' operation.
`token_transf_out`	Replacement string. This argument works together with `re_token_transf_in` and is ignored if `re_token_transf_in` is `NULL` or `NA`.
`token_to_lower`	Logical. Whether tokens must be converted to lowercase before returning the result. The 'token to lower' operation is conducted immediately after the 'token transformation' operation.
`perl`	Logical. Whether the PCRE regular expression flavor is being used in the arguments that contain regular expressions.
`blocksize`	Number that indicates how many corpus files are read to memory `at each individual step' during th` should not be changed, but when one works with exceptionally small corpus files, it may be worthwhile to use a higher number, and when one works with exceptionally large corpus files, it may be worthwhile to use a lower number.
`verbose`	If`TRUE`, messages are printed to the console to indicate progress.
`show_dots, dot_blocksize`	If `TRUE`, dots are printed to the console to indicate progress.
`file_encoding`	File encoding that is assumed in the corpus files.
`ngram_size`	Argument in support of ngrams/skipgrams (see also `max_skip`). If one wants to identify individual tokens, the value of `ngram_size` should be `NULL` or `1`. If one wants to retrieve token ngrams/skipgrams, `ngram_size` should be an integer indicating the size of the ngrams/skipgrams. E.g. `2` for bigrams, or `3` for trigrams, etc.
`ngram_sep`	Character vector of length 1 containing the string that is used to separate/link tokens in the representation of ngrams/skipgrams in the output of this function.
`ngram_n_open`	If `ngram_size` is `2` or higher, and moreover `ngram_n_open` is a number higher than `0`, then ngrams with 'open slots' in them are retrieved. These ngrams with 'open slots' are generalizations of fully lexically specific ngrams (with the generalization being that one or more of the items in the ngram are replaced by a notation that stands for 'any arbitrary token'). For instance, if `ngram_size` is `4` and `ngram_n_open` is `1`, and if moreover the input contains a 4-gram `"it_is_widely_accepted"`, then the output will contain all modifications of `"it_is_widely_accepted"` in which one (since `ngram_n_open` is `1`) of the items in this n-gram is replaced by an open slot. The first and the last

item inside an ngram are never turned into an open slot; only the items in between are candidates for being turned into open slots. Therefore, in the example, the output will contain "it_[]_widely_accepted" and "it_is_[]_accepted".

As a second example, if ngram_size is 5 and ngram_n_open is 2, and if moreover the input contains a 5-gram "it_is_widely_accepted_that", then the output will contain "it_[]_[]_accepted_that", "it_[]_widely_[]_that", and "it_is_[]_[]_that".

ngram_open — Character string used to represent open slots in ngrams in the output of this function.

as_text — Logical. Whether x is to be interpreted as a character vector containing the actual contents of the corpus (if as_text is TRUE) or as a character vector containing the names of the corpus files (if as_text is FALSE). If if as_text is TRUE, then the arguments blocksize, verbose, show_dots, dot_blocksize, and file_encoding are ignored.

Details

The actual token identification is either based on the re_token_splitter argument, a regular expression that identifies the areas between the tokens, or on re_token_extractor, a regular expression that identifies the area that are the tokens. The first mechanism is the default mechanism: the argument re_token_extractor is only used if re_token_splitter is NULL. Currently the implementation of re_token_extractor is a lot less time-efficient than that of re_token_splitter.

Value

An object of the class types, which is based on a character vector. It has additional attributes and methods such as:

- base print(), as_data_frame(), sort() and base::summary() (which returns the number of items and of unique items),
- tibble::as_tibble(),
- the n_types() getter and the explore() method,
- subsetting methods such as keep_types(), keep_pos(), etc. including [] subsetting (see brackets).

An object of class types can be merged with another by means of types_merge(), written to file with write_types() and read from file with write_types().

See Also

as_types()

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."
(tps <- types(toy_corpus, as_text = TRUE))
print(tps)
```

```
as.data.frame(tps)
as_tibble(tps)

sort(tps)
sort(tps, decreasing = TRUE)
```

---

type_freqs *Retrieve frequencies from 'freqlist' object*

Description

`type_freq` and `type_freqs` retrieve the frequency of all or some of the items of a `freqlist` object.

Usage

```
type_freqs(x, types = NULL, with_names = FALSE, ...)

type_freq(x, types = NULL, with_names = FALSE, ...)
```

Arguments

`x`	Object of class `freqlist`.
`types`	`NULL` or a character vector or an object of the class `types`. If the argument `types` is `NULL`, then the frequencies of all the items in `x` are returned, in the order in which these items appear in `x`. If the argument `types` is a character vector or an object of the class `types`, then only the frequencies (in `x`) of the items in `types` are given, in the order in which these items appear in `types`. For all items in `types` that do not occur in `x`, a frequency of zero is returned.
`with_names`	Logical. Whether or not the items in the output should be given names. If `with_names` is `TRUE`, then the names of the types in the frequency list are used as names.
`...`	Additional arguments.

Value

Numeric vector representing the frequencies of the items.

See Also

type_names

Examples

```
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

type_freqs(flist) # frequencies of all items
type_names(flist) # names of all items

type_freqs(flist, with_names = TRUE) # frequencies of all types, with names
type_freqs(flist, c("man", "the")) # frequencies of specific items ...
type_freqs(flist, c("the", "man")) # ... in the requested order
type_freq(flist, "the")            # frequency of one item

# frequencies of specific items can also be printed using subsetting
flist[c("the", "man")]
flist["the"]
```

`type_names` *Return the names of the types in an object*

Description

This method returns the names of the types represented in an object.

Usage

```
type_names(x, ...)

## S3 method for class 'assoc_scores'
type_names(x, ...)

## S3 method for class 'freqlist'
type_names(x, ...)
```

Arguments

`x`	An object of any of the classes for which the method is implemented.
`...`	Additional arguments.

Value

Character vector.

See Also

Other getters and setters: `n_tokens()`, `n_types()`, `orig_ranks()`, `ranks()`, `tot_n_tokens()`

Examples

```
# for a freqlist object
(flist <- freqlist("The man and the mouse.", as_text = TRUE))

type_names(flist)

# for an assoc_scores object
a <- c(10,    30,    15,    1)
b <- c(200, 1000,  5000,  300)
c <- c(100,   14,    16,    4)
d <- c(300, 5000, 10000, 6000)
types <- c("four", "fictitious", "toy", "examples")

(scores <- assoc_abcd(a, b, c, d, types = types))
type_names(scores)
```

---

write_assoc *Write association scores to file*

Description

This function writes an object of class assoc_scores to a file.

Usage

```
write_assoc(x, file = "", sep = "\t")
```

Arguments

x	An object of class assoc_scores.
file	Name of the output file.
sep	Field separator for the output file.

Value

Invisibly, x.

See Also

read_assoc()

Other writing functions: write_conc(), write_fnames(), write_freqlist(), write_tokens(), write_txt(), write_types()

Examples

```
txt1 <- "we're just two lost souls swimming in a fish bowl,
year after year, running over the same old ground,
what have we found? the same old fears.
wish you were here."
flist1 <- freqlist(txt1, as_text = TRUE)
txt2 <- "picture yourself in a boat on a river
with tangerine dreams and marmelade skies
somebody calls you, you answer quite slowly
a girl with kaleidoscope eyes"
flist2 <- freqlist(txt2, as_text = TRUE)
(scores <- assoc_scores(flist1, flist2, min_freq = 0))

write_assoc(scores, "example_scores.tab")
(scores2 <- read_assoc("example_scores.tab"))
```

---

write_conc *Write a concordance to file.*

Description

This function writes an object of class conc to a file.

Usage

```
write_conc(x, file = "", sep = "\t")
```

Arguments

x	Object of class conc.
file	Path to output file.
sep	Field separator for the columns in the output file.

Value

Invisibly, x.

See Also

read_conc()

Other writing functions: write_assoc(), write_fnames(), write_freqlist(), write_tokens(), write_txt(), write_types()

Examples

```
(d <- conc('A very small corpus.', '\\w+', as_text = TRUE))

write_conc(d, "example_data.tab")
(d2 <- read_conc("example_data.tab"))
```

`write_fnames` *Write a collection of filenames to a text file*

Description

This function writes an object of class `fnames` to a text file. Each filename is written in a separate line. The file encoding is always `"UTF-8"`. In addition, it can store metadata in an additional configuration file.

Usage

```
write_fnames(x, file, ...)
```

Arguments

`x`	Object of class `fnames`.
`file`	Path to output file.
`...`	Additional arguments (not implemented).

Value

Invisibly, `x`.

See Also

`read_fnames()`

Other writing functions: `write_assoc()`, `write_conc()`, `write_freqlist()`, `write_tokens()`, `write_txt()`, `write_types()`

Examples

```
cwd_fnames <- as_fnames(c("file1.txt", "file2.txt"))
write_fnames(cwd_fnames, "file_with_filenames.txt")
cwd_fnames_2 <- read_fnames("file_with_filenames.txt")
```

write_freqlist *Write a frequency list to a csv file*

Description

This function writes an object of the class `freqlist` to a csv file. The resulting csv file contains two columns, the first being the type and the second being the frequency of that type. The file also contains a header line with the names of both columns.

Usage

```
write_freqlist(x, file, sep = "\t", make_config_file = TRUE, ...)
```

Arguments

`x`	Object of class `freqlist`.
`file`	Character vector of length 1. Path to the output file.
`sep`	Character vector of length 1. Column separator.
`make_config_file`	Logical. Whether or not a configuration file needs to be created. In most circumstances, this should be set to `TRUE`.
`...`	Additional arguments (not implemented).

Details

`write_freqlist` not only writes to the file `file`, but also creates a configuration file with a name that is identical to `file`, except that it has the filename extension `".yaml"`. The frequency list attributes `"tot_n_tokens"` and `"tot_n_types"` are stored to that configuration file.

Value

Invisibly, `x`.

See Also

`read_freqlist()`

Other writing functions: `write_assoc()`, `write_conc()`, `write_fnames()`, `write_tokens()`, `write_txt()`, `write_types()`

Examples

```
toy_corpus <- "Once upon a time there was a tiny toy corpus.
It consisted of three sentences. And it lived happily ever after."
freqs <- freqlist(toy_corpus, as_text = TRUE)

print(freqs, n = 1000)
```

```
write_freqlist(freqs, "example_freqlist.csv")
freqs2 <- read_freqlist("example_freqlist.csv")
print(freqs2, n = 1000)
```

write_tokens — *Write a* `tokens` *object to a text file*

Description

This function writes an object of the class `tokens` to a text file. Each token is written to a separate line. The file encoding is always "UTF-8". This file can later be read with `read_tokens()`.

Usage

```
write_tokens(x, file, ...)
```

Arguments

`x`	An object of class `tokens`.
`file`	Name of the output file.
`...`	Additional arguments (not implemented).

Value

Invisibly, `x`.

See Also

`read_tokens()`

Other writing functions: `write_assoc()`, `write_conc()`, `write_fnames()`, `write_freqlist()`, `write_txt()`, `write_types()`

Examples

```
(tks <- tokenize("The old man and the sea."))
write_tokens(tks, "file_with_tokens.txt")
(tks2 <- read_tokens("file_with_tokens.txt"))
```

write_txt *Write a character vector to a text file*

Description

This function writes a character vector to a text file. By default, each item in the character vector becomes a line in the text file.

Usage

```
write_txt(x, file = "", line_glue = "\n")
```

Arguments

x	A character vector.
file	Name of the output file.
line_glue	Character string to be used as end-of-line marker on disk or NA for no end-of-line marker (so that x becomes a single line).

Value

Invisibly, x.

See Also

read_txt()

Other writing functions: write_assoc(), write_conc(), write_fnames(), write_freqlist(), write_tokens(), write_types()

Examples

```
x <- "This is
a small
text."

# write the text to a text file
write_txt(x, "example-text-file.txt")
# read a text from file
y <- read_txt("example-text-file.txt")
y
```

write_types — *Write a vector of types to a text file*

Description

This function writes an object of the class `types` to a text file. Each type is written to a separate line. The file encoding that is used is `"UTF-8"`.

Usage

```
write_types(x, file, ...)
```

Arguments

`x`	Object of class `types`.
`file`	Name of the output file
`...`	Additional arguments (not implemented).

Value

Invisibly, `x`.

See Also

`read_types()`

Other writing functions: `write_assoc()`, `write_conc()`, `write_fnames()`, `write_freqlist()`, `write_tokens()`, `write_txt()`

Examples

```
types <- as_types(c("first", "second", "third"))
write_types(types, "file_with_types.txt")
types_2 <- read_types("file_with_types.txt")
```

zero_plus *Make all values strictly higher than zero*

Description

This is an auxiliary function that makes all values in numeric vector x strictly positive by replacing all values equal to or lower than zero with the values in `small.pos`. `small_pos` stands for 'small positive constant'.

Usage

```
zero_plus(x, small_pos = 1e-05)
```

Arguments

`x`	A numeric vector.
`small_pos`	A (small) positive number to replace negative values and 0s.

Value

A copy of `x` in which all values equal to or lower than zero are replaced by `small_pos`.

Examples

```
(x <- rnorm(30))
zero_plus(x)
```

Index