Package: act (via r-universe)

November 1, 2026

Type Package

Title Aligned Corpus Toolkit

Version 3.4.6

Date 2026-08-04

Maintainer Oliver Ehmer <oliver.ehmer@uos.de>

Description The Aligned Corpus Toolkit (act) is designed for linguists that work with time aligned transcription data. It offers functions to import and export various annotation file formats ('ELAN' .eaf, 'EXMARaLDA .exb and 'Praat' .TextGrid files), create print transcripts in the style of conversation analysis, search transcripts (span searches across multiple annotations, search in normalized annotations, make concordances etc.), export and re-import search results (.csv and 'Excel' .xlsx format), create cuts for the search results (print transcripts, audio/video cuts using 'FFmpeg' and video sub titles in 'Subrib title' .srt format), modify the data in a corpus (search/replace, delete, filter etc.), interact with 'Praat' using 'Praat'-scripts, and exchange data with the 'rPraat' package. The package is itself written in R and may be expanded by other users.

License GPL-3

Encoding UTF-8

LazyData TRUE

Depends R (>= 4.2.0)

Imports cli, digest, exifr, fs, dplyr, jsonlite, methods, processx, srt, stringr, stringi, tools, textutils, utils, progress, XML, xml2, officer, openxlsx, openxlsx2, glue

Suggests knitr, rmarkdown, rstudioapi

SystemRequirements FFmpeg

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

URL http://www.oliverehmer.de, https://github.com/oliverehmer/act

NeedsCompilation no

Author Oliver Ehmer [aut, cre]

Config/roxygen2/version 8.1.1

Config/pak/sysreqs
cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev perl

Repository https://oliverehmer.r-universe.dev

Date/Publication 2026-11-01 20:05:00 UTC

RemoteUrl https://github.com/oliverehmer/act

RemoteRef HEAD

RemoteSha aefb99cd0099fbaf78c5d5670205a5f2646784d4

Contents

act . . . 5
annotations_add . . . 9
annotations_all . . . 11
annotations_delete . . . 11
annotations_delete_empty . . . 12
annotations_fill_tier_section . . . 14
annotations_insert_from_search_to_tier . . . 15
annotations_matrix . . . 17
annotations_replace_copy . . . 18
corpus-class . . . 20
corpus_export . . . 21
corpus_import . . . 23
corpus_merge . . . 25
corpus_new . . . 26
examplecorpus . . . 28
examplecorpus_media . . . 30
export_create_pfsx . . . 30
export_docx . . . 31
export_eaf . . . 33
export_edl . . . 35
export_exb . . . 37
export_rpraat . . . 38
export_srt . . . 39
export_textgrid . . . 41
export_txt . . . 42
helper_annotationfile_read . . . 45
helper_annotations_columns_order . . . 46
helper_audio_filter_build . . . 46
helper_audio_filter_parts . . . 47
helper_corpus_export_cured . . . 49

helper_cutlist_lines . . . 49
helper_cutlist_save . . . 51
helper_encoding_file_detect . . . 52
helper_encoding_name_normalize . . . 53
helper_encoding_names_equivalent . . . 53
helper_ffmpeg_args . . . 54
helper_ffmpeg_args_audio . . . 56
helper_ffmpeg_args_clip . . . 57
helper_ffmpeg_check . . . 60
helper_ffmpeg_input_flags . . . 61
helper_ffmpeg_keyframe . . . 61
helper_ffmpeg_path . . . 62
helper_ffmpeg_run . . . 63
helper_ffprobe_path . . . 64
helper_file_app_available . . . 64
helper_file_date_key_make . . . 65
helper_file_hash_compute . . . 66
helper_file_newest_select . . . 66
helper_file_open_in . . . 67
helper_file_reveal . . . 68
helper_file_state . . . 68
helper_file_state_changed . . . 69
helper_history_changed . . . 70
helper_history_mark_export . . . 70
helper_layout_anchors . . . 71
helper_layout_bracket_pairs . . . 72
helper_layout_render . . . 73
helper_layout_symbol_matches . . . 75
helper_media_audio_track_exists . . . 76
helper_media_type . . . 77
helper_media_video_timing_read . . . 77
helper_metadata_exif_argv . . . 78
helper_metadata_exif_write . . . 78
helper_metadata_ffmpeg_args . . . 79
helper_metadata_ffmpeg_argv . . . 80
helper_metadata_probe_fps . . . 81
helper_program_paths_detect . . . 81
helper_progress_callback_get . . . 82
helper_progress_callback_set . . . 82
helper_progress_eta_format . . . 83
helper_progress_set . . . 84
helper_progress_tick . . . 84
helper_style_acronym . . . 85
helper_style_appearance . . . 86
helper_style_docx . . . 86
helper_style_list . . . 87
helper_style_read . . . 87
helper_style_video . . . 88

helper_text_graphemes_split . . . 89
helper_tiers_filter_create . . . 89
helper_tiers_merge_tables . . . 90
helper_tiers_new_table . . . 91
helper_tiers_sort_table . . . 92
helper_time_format . . . 94
helper_time_formats_list . . . 95
helper_transcript_names_get . . . 96
helper_transcript_names_make . . . 97
helper_transcript_names_set . . . 98
helper_transcript_patch_apply . . . 99
helper_transcript_patch_make . . . 100
helper_transcripts_compare . . . 101
import . . . 102
import_eaf . . . 104
import_exb . . . 105
import_rpraat . . . 106
import_srt . . . 108
import_textgrid . . . 109
info . . . 111
info_summarized . . . 112
layers_convert_column_to_tier . . . 113
matrix_load . . . 114
matrix_save . . . 115
media_assign . . . 116
media_build . . . 117
media_delete . . . 118
media_metadata_fill . . . 119
media_metadata_read . . . 119
media_path_to_existing_file . . . 120
media_select . . . 121
merge_docx . . . 123
options_delete . . . 124
options_reset . . . 124
options_show . . . 124
search-class . . . 126
search_concordance . . . 128
search_cuts . . . 129
search_cuts_media . . . 130
search_cuts_printtranscript . . . 134
search_cuts_srt . . . 136
search_makefilter . . . 138
search_new . . . 140
search_openresult_inelan . . . 142
search_openresult_inpraat . . . 143
search_openresult_inquicktime . . . 145
search_playresults_inquicktime . . . 146
search_results_export . . . 147

search_results_import . . . 149
search_run . . . 150
search_searchandopen_inpraat . . . 151
search_stills . . . 152
search_sub . . . 154
search_transcript_content . . . 155
search_transcript_fulltext . . . 156
tiers_add . . . 157
tiers_all . . . 159
tiers_convert . . . 160
tiers_delete . . . 162
tiers_rename . . . 163
tiers_sort . . . 164
transcript-class . . . 165
transcripts_add . . . 167
transcripts_cure . . . 168
transcripts_cure_single . . . 170
transcripts_delete . . . 173
transcripts_filter . . . 174
transcripts_filter_remove_single . . . 175
transcripts_filter_single . . . 176
transcripts_merge . . . 178
transcripts_openin_elan . . . 180
transcripts_openin_praat . . . 182
transcripts_rename . . . 183
transcripts_update_fulltexts . . . 185
transcripts_update_normalization . . . 186

Index **188**

`act` *Aligned Corpus Toolkit*

Description

The Aligned Corpus Toolkit (act) is designed for linguists that work with time aligned transcription data. It offers functions to import and export various annotation file formats ('ELAN' .eaf, 'EXMARaLDA .exb and 'Praat' .TextGrid files), create print transcripts in the style of conversation analysis, search transcripts (span searches across multiple annotations, search in normalized annotations, make concordances etc.), export and re-import search results (.csv and 'Excel' .xlsx format), create cuts for the search results (print transcripts, audio/video cuts using 'FFmpeg' and video sub titles in 'Subrib title' .srt format), modify the data in a corpus (search/replace, delete, filter etc.), interact with 'Praat' using 'Praat'-scripts, and exchange data with the 'rPraat' package. The package is itself written in R and may be expanded by other users.

act functions

...

Vignettes

- `vignette("export_docx_styles", package = "act")` – transcript profiles (format of the print transcripts)
- `vignette("install_ffmpeg", package = "act")` – Install FFmpeg
- `vignette("praat_remote_control", package = "act")` – Praat remote control

Package options

The package has numerous options that change the internal workings of the package. Please see `act::options_show()` and the information given there.

Examples

```
library(act)

# ========== Example data set
# There is an example data set consisting of annotation files and corresponding
# media files.
# While the annotation files are copied to your computer when installing
# the media files are not.
# You can either download the full data set from GitHub or decide to work
# only with the annotation files.

# The example data set (only annotation files) is stored at the following location:
path <- system.file("extdata", "examplecorpus", package="act")

# Since this folder is quite difficult to access, you might consider copying the
# contents of this folder to a more convenient location.
# The following commands will create a new folder called 'examplecorpus' in the
# folder 'path'.
# You will find the data there.
## Not run:
path <- "EXISTING_FOLDER_ON_YOUR_COMPUTER"
sourcepath <- system.file("extdata", "examplecorpus", package="act")
if (!dir.exists(path)) {dir.create(path)}
file.copy(sourcepath, dirname(path), recursive=TRUE)

## End(Not run)

# To download the full data set (including media files) from GitHub
# use the following code.
# In the first line specify an existing folder on your computer.
# The following lines will then download the example data set from GitHub
# and copy them to a sub folder called 'examplecorpus' in the folder 'path'.
# You will find the data there.
## Not run:
path <- "EXISTING_FOLDER_ON_YOUR_COMPUTER"
path <- "/Users/oliverehmer/Desktop"
sourceurl <- "https://github.com/oliverehmer/act_examplecorpus/archive/master.zip"
temp <- tempfile()
download.file(sourceurl, temp)
```

```
unzip(zipfile=temp, exdir=path)
path <- file.path(path, "act_examplecorpus-main")

## End(Not run)

# ========== Create a corpus object and load data
# Now that we have the example data accessible, we can create a corpus object.
# The corpus object is a structured collection of all the information that you can
# work with using act.
# It will contain the information of each transcript, links to media files and further
# meta data.

# --- Locate folder with annotation files
# When creating a corpus object you will need to specify where your annotation
# files ('Praat' '.TextGrids' or 'ELAN' .eaf) are located.
# We will use the example data, that we have just located in 'path'.
path

# In case that you want to use your own data, you can set the path here:
## Not run:
 path <- "EXISTING_FOLDER_ON_YOUR_COMPUTER"

## End(Not run)

# --- Create corpus object and load annotation files
# The following command will create a corpus object, with the name 'examplecorpus'.
examplecorpus <- act::corpus_new(
 pathsAnnotationFiles = path,
 pathsMediaFiles = path,
 name = "examplecorpus"
)

# The act package assumes, that annotation files and media files have the same base
# name and differ only in the suffix (e.g. 'filename.TextGrid' and 'filename.wav'/
# 'filename.mp4').
# This allows act to automatically link media files to the transcripts.

# --- Information about your corpus
# The following command will give you a summary of the data contained in your corpus object.
examplecorpus
# More detailed information about the transcripts in your corpus object is available by
# calling the function act::info()
act::info(examplecorpus)
# If you are working in R studio, a nice way of inspecting this information is the following:
## Not run:
 View(act::info(examplecorpus)$transcripts)
 View(act::info(examplecorpus)$tiers)

## End(Not run)

# ========== all data
# You can also get all data that is in the loaded annotation files in a data frame:
all_annotations <- act::annotations_all(examplecorpus)
```

```
## Not run:
 View(all_annotations)

## End(Not run)

# ========== Search
# Let's do some searches in the data.
# Search for the 1. Person Singular Pronoun in Spanish 'yo' in the examplecorpus
mysearch <- act::search_new(x=examplecorpus,
       pattern= "yo")
# Have a look at the result:
mysearch

# Directly view all search results in the viewer
## Not run:
 View(mysearch@results)

## End(Not run)

# --- Search original vs. normalized content
# You can either search in the original 'content' of the annotations,
# or you can search in a 'normalized' version of the annotations.
# Let's compare the two modes.
mysearch.norm  <- act::search_new(examplecorpus, pattern="yo", searchNormalized=TRUE)
mysearch.org   <- act::search_new(examplecorpus, pattern="yo", searchNormalized=FALSE)
# There is a difference in the number of results.
mysearch.norm@results.nr
mysearch.org@results.nr

# The difference is because during in the normalized version, for instance, capital letters
# will be converted to small letters.
# In our case, one annotation in the example corpus contains a "yO" with a
# capital letter:
mysearch <- act::search_new(examplecorpus, pattern="yO", searchNormalized=FALSE)
mysearch@results$hit

# During normalization a range of normalization procedures will be applied, using a
# replacement matrix. This matrix searches and replaces certain patterns, that you want to
# exclude from the normalized content.
# By default, normalization gets rid of all transcription conventions of GAT.
# You may, in addition, also customize the replacement matrix to your own needs/transcription
# conventions.

# --- Search original content vs. full text
# There are two search modes.
# The 'fulltext' mode will will find matches across annotations.
# The 'content' mode will will respect the temporal boundaries of the original annotations.

# Let's define a search pattern with a certain span.
myRegEx <- "\\bno\\b.{1,20}pero"
# This regular expression matches the Spanish word "no" 'no' followed by a "pero" 'but'
# in a distance ranging from 1 to 20 characters.
```

```
# The 'content' search mode will not find any hit.
mysearch <- act::search_new(examplecorpus, pattern=myRegEx, searchMode="content")
mysearch@results.nr

# The 'fulltext' search mode will not find two hits that extend over several annotations.
mysearch <- act::search_new(examplecorpus, pattern=myRegEx, searchMode="fulltext")
mysearch@results.nr
cat(mysearch@results$hit[1])
cat(mysearch@results$hit[2])
```

`annotations_add` *Add annotation to transcript with overlap handling*

Description

Adds a single annotation (interval or point) to a transcript object. Automatically checks for overlapping annotations and handles them based on the overwrite parameter. Always ensures no invalid overlaps exist.

Usage

```
annotations_add(
  x,
  tierName,
  startsec,
  endsec,
  content = "",
  content.norm = "",
  overwrite = FALSE
)
```

Arguments

`x`	Transcript object.
`tierName`	Character string; name of the tier (exact name, not regex).
`startsec`	Numeric; start time in seconds.
`endsec`	Numeric; end time in seconds.
`content`	Character string; content of the annotation. Default `""`.
`content.norm`	Character string; normalized content. Default `""`.
`overwrite`	Logical; if `FALSE`, throws error when overlap exists. If `TRUE`, trims/splits existing overlapping annotations. Default `FALSE`.

Details

Overlap Handling:

The function always checks for overlapping annotations in the specified tier:

When `overwrite = FALSE` **(default):**

- If overlap exists → Error with `stop()`
- If no overlap → Insert new annotation

When `overwrite = TRUE`**:**

- Existing overlapping annotations are trimmed/split:
  - Left remainder: `[old_start, new_start]` (if `old_start < new_start`)
  - Right remainder: `[new_end, old_end]` (if `new_end < old_end`)
  - Middle part is replaced by new annotation
- New annotation is inserted

Automatic Operations:

- Assigns unique annotation ID
- Sets all char.* fields to NA (filled by fulltext operations)
- Maintains proper data.frame structure
- Updates modification time and history

Value

Transcript object with added annotation.

Examples

```
## Not run:
library(act)

# Add annotation (error if overlap exists)
t <- annotations_add(
  x = examplecorpus@transcripts[[1]],
  tierName = "status",
  startsec = 10.5,
  endsec = 15.2,
  content = "o0.s0"
)

# Add annotation and overwrite overlaps
t <- annotations_add(
  x = examplecorpus@transcripts[[1]],
  tierName = "status",
  startsec = 10.5,
  endsec = 15.2,
  content = "new content",
  overwrite = TRUE
)

## End(Not run)
```

annotations_all *All annotations in a corpus*

Description

Merges annotations from all transcripts in a corpus and returns a data frame.

Usage

```
annotations_all(x)
```

Arguments

x	Corpus object.

Value

data frame

Examples

```
library(act)

#Get data frame with all annotations
allannotations <- act::annotations_all(examplecorpus)

#Have a look at the number of annotations
nrow(allannotations)
```

annotations_delete *Delete annotations*

Description

Delete annotations in a corpus object. If only certain transcripts or tiers should be affected set the parameter `filterTranscriptNames` and `filterTierNames`. In case that you want to select transcripts and/or tiers by using regular expressions use the function `act::search_makefilter` first.

Usage

```
annotations_delete(
  x,
  pattern = "",
  filterTranscriptNames = NULL,
  filterTierNames = NULL
)
```

Arguments

x	Corpus object.
pattern	Character string; regular expression; all annotations that match this expression will be deleted.
filterTranscriptNames	Vector of character strings; names of the transcripts to be included.
filterTierNames	Character string; names of the tiers to be included.

Value

Corpus object.

Examples

```
library(act)

# Set the regular expression which annotations should be deleted.
# In this case: all annotations that contain the letter "a"
myRegEx <- "a"

# Have a look at all annotations in the first transcript
examplecorpus@transcripts[[1]]@annotations$content

# Some of them match to the regular expression
hits <- grep(pattern=myRegEx, x=examplecorpus@transcripts[[1]]@annotations$content)
examplecorpus@transcripts[[1]]@annotations$content[hits]
# Others don't match the regular expression
examplecorpus@transcripts[[1]]@annotations$content[-hits]

# Run the function and delete the annotations that match the regular expression
test <- act::annotations_delete (x=examplecorpus, pattern=myRegEx)

# Compare how many data rows are in the first transcript in
# the example corpus and in the newly created test corpus:
nrow(examplecorpus@transcripts[[1]]@annotations)
nrow(test@transcripts[[1]]@annotations)

# Only the annotations are left, that did not match the regular expression:
test@transcripts[[1]]@annotations$content
```

---

annotations_delete_empty

Delete empty annotations

Description

Delete empty annotations in a corpus object. If only certain transcripts or tiers should be affected set the parameter `filterTranscriptNames` and `filterTierNames`. In case that you want to select transcripts and/or tiers by using regular expressions use the function `act::search_makefilter` first.

Usage

```
annotations_delete_empty(
  x,
  trim = FALSE,
  filterTranscriptNames = NULL,
  filterTierNames = NULL
)
```

Arguments

`x`	Corpus object.
`trim`	Logical; if `TRUE` leading and trailing spaces will be deleted before checking (as a consequence record sets that contain only spaces will be deleted, too).
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be included.
`filterTierNames`	Character string; names of the tiers to be included.

Value

Corpus object.

Examples

```
library(act)

# In the example corpus are no empty annotations.
# Empty annotations are deleted by default when annotation files are loaded.
# So let's first make an empty annotation.

# Check the first annotation in the first transcript
examplecorpus@transcripts[[1]]@annotations$content[[1]]

# Empty the contents of this annotation
examplecorpus@transcripts[[1]]@annotations$content[[1]] <- ""

# Run the function
test <- act::annotations_delete_empty (x=examplecorpus)

# Compare how many data rows are in the first transcript in
# the example corpus and in the newly created test corpus:
nrow(examplecorpus@transcripts[[1]]@annotations)
nrow(test@transcripts[[1]]@annotations)
```

`annotations_fill_tier_section` *Fill tier section with content*

Description

Fills annotations in a specific tier and time range with content. By default, only fills empty annotations (does not overwrite existing content).

Usage

```
annotations_fill_tier_section(
  x,
  tier_pattern,
  content,
  startsec,
  endsec,
  overwrite = FALSE,
  create_if_missing = TRUE,
  perl = TRUE
)
```

Arguments

`x`	Transcript object.
`tier_pattern`	Character string; regex pattern to identify target tier(s).
`content`	Character string; content to insert.
`startsec`	Numeric; start time in seconds.
`endsec`	Numeric; end time in seconds.
`overwrite`	Logical; if `TRUE`, overwrites existing content. If `FALSE`, only fills empty annotations. Default `FALSE`.
`create_if_missing`	Logical; if `TRUE` and no annotation exists in the time range, creates a new annotation spanning the entire range. Default `TRUE`.
`perl`	Logical; use Perl-compatible regex for tier_pattern. Default `TRUE`.

Value

Transcript object with modified annotations.

Examples

```
## Not run:
library(act)

# Fill status tier with "o0.s0"
t <- annotations_fill_tier_section(
  x = examplecorpus@transcripts[[1]],
  tier_pattern = "(?i)status",
  content = "o0.s0",
  startsec = 10,
  endsec = 20
)

## End(Not run)
```

`annotations_insert_from_search_to_tier` *Inserts search results as annotations in tiers*

Description

The function will insert the results of a search as annotations into a specified destination tier. Results from different tiers will all be inserted into the same destination tier. It can be specified how overlapping search results or search results from the same annotation will be handled. Please note that hits with exactly the same start and end time will always be merged (e.g. they are not treated as overlapping).

Usage

```
annotations_insert_from_search_to_tier(
  x,
  s,
  destTier = "destinationTier",
  destTierAddMissing = TRUE,
  contentFromColname = "hit",
  interruptOverlap = FALSE,
  filterTranscriptNames = NULL,
  filterTierNames = NULL,
  collapseString = " | "
)
```

Arguments

`x`	Corpus object.
`s`	Search object.
`destTier`	Character string; name of the tier to which the hit should be copied (if no copying is intended set to NA).

`destTierAddMissing`
: Logical; if `TRUE` the destination tier will be added if missing in the transcript object, if `FALSE` an error will be raised if the destination tier is missing.

`contentFromColname`
: Character string; names of the search results column from which the content of the new annotation shall be copied. Chose for example "resultID", “hit” or “content” or the name of any column that you have added to the results data frame.

`interruptOverlap`
: Integer; How to proceed in case of overlapping search results: TRUE=insertion will be interrupted, FALSE=overlapping annotations will be merged.

`filterTranscriptNames`
: Vector of character strings; names of the transcripts to be included from search results.

`filterTierNames`
: Character string; names of the tiers to be included form search results.

`collapseString`
: Character string; will be used to collapse multiple search results into one string.

Value

Corpus object.

Examples

```
library(act)

# Have a look at the first transcript in the example corpus:
printtranscript <- act::export_txt(examplecorpus@transcripts[[1]])
cat(printtranscript)
# In line 01 there is the word "UN".

# Replace this word by "XXX" in the entire corpus
test <- act::annotations_replace_copy(x=examplecorpus,
           pattern="\\bUN\\b",
           replacement="XXX")

# Have a look at the first transcript in the corpus object test:
printtranscript <- act::export_txt(test@transcripts[[1]])
cat(printtranscript)
# In line 01 there is now "XXX" instead of "UN"

# Insert a tier called "newTier" into all transcripts in the corpus:
for (t in examplecorpus@transcripts) {
 sortVector <- c(t@tiers$name, "newTier")
 examplecorpus <- act::tiers_sort(x=examplecorpus,
 sortVector=sortVector,
 filterTranscriptNames=t@name,
 tiersAddMissing=TRUE)
}
# Check that the first transcript now contains the newTier
examplecorpus@transcripts[[1]]@tiers
```

```
# Now replace "UN" by "YYY" in the entire corpus and
# copy the search hit to "newTier".
test <- act::annotations_replace_copy(x=examplecorpus,
           pattern="\\bUN\\b",
           replacement="YYY",
           destTier = "newTier")

# Have a look again at the first transcript in the corpus object test.
printtranscript <- act::export_txt(test@transcripts[[1]])
cat(printtranscript)
# In line 01 you see that "UN" has been replaced by "YYY.
# In line 02 you see that it has been copied to the tier "newTier".

# If you only want to copy a search hit but not replace it in the original
# leave replacement="", which is the default
test <- act::annotations_replace_copy(x=examplecorpus,
           pattern="\\bUN\\b",
           destTier = "newTier")
printtranscript <- act::export_txt(test@transcripts[[1]])
cat(printtranscript)
# In line 01 you see that "UN" has been maintained.
# In line 02 you see that "UN" it has been copied to the tier "newTier".
```

`annotations_matrix` *Search and replace contents of annotations using a matrix*

Description

This functions performs a search and replace in the contents of an annotation. A simple matrix consisting of two columns will be used. The first column of the matrix needs to contain the search string, the second column the replacement string. The matrix needs to be in CSV format.

Usage

```
annotations_matrix(x, pathReplacementMatrix, filterTranscriptNames = NULL)
```

Arguments

`x`	Corpus object.
`pathReplacementMatrix`	Character string; path to replacement matrix (a CSV file).
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be included.

Value

Corpus object.

See Also

matrix_load() for loading the matrix and matrix_save() for saving the matrix to a CSV file.

If only certain transcripts or tiers should be affected set the parameter filterTranscriptNames. In case that you want to select transcripts by using regular expressions use the function act::search_makefilter first.

media_delete, media_path_to_existing_file

Examples

```
library(act)

# An example replacement matrix comes with the package.
# It replaces most of the GAT conventions.
path <- system.file("extdata", "normalization", "normalization_matrix.csv", package="act")

# Have a look at the matrix
mymatrix <- act::matrix_load(path)
mymatrix

# Apply matrix to examplecorpus
test <- act::annotations_matrix(x=examplecorpus, pathReplacementMatrix=path)

# Compare some annotations in the original examplecorpus object and
# in the modified corpus object test
examplecorpus@transcripts[[1]]@annotations$content[[1]]
test@transcripts[[1]]@annotations$content[[1]]

examplecorpus@transcripts[[2]]@annotations$content[[3]]
test@transcripts[[2]]@annotations$content[[3]]
```

---

annotations_replace_copy

Search, replace and copy the contents of annotations

Description

The function searches within the contents of annotations and replaces the search hits. In addition the search hit may be copied to another tier. In case that there is NO overlapping annotation in the destination tier a new annotation will be created (based on the time values of the original annotation). In case that there is an overlapping annotation in the destination tier, the search result will be added at the end.

Usage

```
annotations_replace_copy(
  x,
  pattern,
```

```
  replacement = NULL,
  destTier = NULL,
  destTierAddMissing = TRUE,
  filterTranscriptNames = NULL,
  filterTierNames = NULL,
  collapseString = " | "
)
```

Arguments

x	Corpus object.
pattern	Character string; search pattern as regular expression.
replacement	Character string; replacement.
destTier	Character string; name of the tier to which the hit should be copied (if no copying is intended set to NA).
destTierAddMissing	Logical; if TRUE the destination tier will be added if missing in the transcript object, if FALSE an error will be raised if the destination tier is missing.
filterTranscriptNames	Vector of character strings; names of the transcripts to be included.
filterTierNames	Character string; names of the tiers to be included.
collapseString	Character string; will be used to collapse multiple search results into one string.

Details

If only certain transcripts or tiers should be affected set the parameter `filterTranscriptNames` and `filterTierNames`. In case that you want to select transcripts and/or tiers by using regular expressions use the function `act::search_makefilter` first.

Value

Corpus object.

Examples

```
library(act)

# Have a look at the first transcript in the example corpus:
printtranscript <- act::export_txt(examplecorpus@transcripts[[1]])
cat(printtranscript)
# In line 01 there is the word "UN".

# Replace this word by "XXX" in the entire corpus
test <- act::annotations_replace_copy(x=examplecorpus,
           pattern="\\bUN\\b",
           replacement="XXX")

# Have a look at the first transcript in the corpus object test:
```

```
printtranscript <- act::export_txt(test@transcripts[[1]])
cat(printtranscript)
# In line 01 there is now "XXX" instead of "UN"

# Insert a tier called "newTier" into all transcripts in the corpus:
for (t in examplecorpus@transcripts) {
 sortVector <- c(t@tiers$name, "newTier")
 examplecorpus <- act::tiers_sort(x=examplecorpus,
 sortVector=sortVector,
 filterTranscriptNames=t@name,
 tiersAddMissing=TRUE)
}
# Check that the first transcript now contains the newTier
examplecorpus@transcripts[[1]]@tiers

# Now replace "UN" by "YYY" in the entire corpus and
# copy the search hit to "newTier".
test <- act::annotations_replace_copy(x=examplecorpus,
           pattern="\\bUN\\b",
           replacement="YYY",
           destTier = "newTier")

# Have a look again at the first transcript in the corpus object test.
printtranscript <- act::export_txt(test@transcripts[[1]])
cat(printtranscript)
# In line 01 you see that "UN" has been replaced by "YYY.
# In line 02 you see that it has been copied to the tier "newTier".

# If you only want to copy a search hit but not replace it in the original
# leave replacement="", which is the default
test <- act::annotations_replace_copy(x=examplecorpus,
           pattern="\\bUN\\b",
           destTier = "newTier")
printtranscript <- act::export_txt(test@transcripts[[1]])
cat(printtranscript)
# In line 01 you see that "UN" has been maintained.
# In line 02 you see that "UN" it has been copied to the tier "newTier".
```

`corpus-class` *Corpus object*

Description

This is the main object the act package uses. It collects the annotations and meta data from loaded annotation files.

Details

Some of the slots are defined by the user. Some slots report results, such as `@import.results` and `@media.assign.results`. What the functions did to a transcript is recorded in the `@history` of

that transcript object. Other slots are settings and are used when performing functions on the corpus object.

Slots

`name` Character string; Name of the corpus.

`transcripts` List of transcript objects; Each annotation file that has been load is stored in this list as a transcript object.

`paths.annotation.files` Vector of character strings; Path(s) to one or several folders where your annotation files are located.

`paths.media.files` Vector of character strings; Path(s) to one or several folders where your media files are located.

`normalization.matrix` Data.frame; Replacement matrix used for normalizing the annotations. To change the normalization matrix use `x@normalization.matrix <- act::matrix_load(path="...")`

`import.skip.double.files` Character string; controls handling of duplicate transcript names. `"warn_keep_newest"` keeps the newest file and warns. `"warn_keep_first"` keeps the first file and warns. `"error"` aborts on duplicates. `"allow"` allows duplicates without warning. For backward compatibility, logical values are accepted: `TRUE` is treated as `"warn_keep_first"`, `FALSE` as `"allow"`.

`import.names.include` Vector of character strings; Only files matching this regular expression will be imported into the corpus.

`import.names.exclude` Vector of character strings; Files matching this regular expression will be skipped and not imported into the corpus.

`import.names.modify` List; Options how to modify the names of the transcript objects when they are added to the corpus. These options are useful, for instacne, if your annotation files contain character sequences that you do not want to include into the transcript name in the corpus (e.g. if you regularly add a date to the file name of your annotations files as 'myFile_2020-09-21.TextGrid').

`import.results` Data.frame; information about the import of the annotation files.

`media.assign.results` Data.frame; information about the media assignment.

Examples

```
library(act)

examplecorpus
```

`corpus_export` *Export transcripts of a corpus*

Description

Exports all (or some) transcript objects in a corpus object to different annotation file formats. If only some transcripts or tiers should be affected set the parameter `filterTranscriptNames` and `filterTierNames`. In case that you want to select transcripts and/or tiers by using regular expressions use the function `act::search_makefilter` first.

Usage

```
corpus_export(
  x,
  folderOutput,
  filterTranscriptNames = NULL,
  filterTierNames = NULL,
  formats = c("docx", "eaf", "exb", "edl", "srt", "textgrid", "txt"),
  createMediaLinks = TRUE,
  createFolderOutput = TRUE,
  style = NULL
)
```

Arguments

`x`	Corpus object.
`folderOutput`	Character string; path to a folder where the transcription files will be saved. By default the forlder will be created recursively it does not exist.
`filterTranscriptNames`	Vector of character strings; names of transcripts to be included. If left unspecified, all transcripts will be exported.
`filterTierNames`	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
`formats`	Vector with one or more character strings; output formats, accepted values: 'eaf', 'exb', and 'textgrid', 'srt' and 'edl', 'docx' and 'txt'. If left unspecified, all supported formats will be exported.
`createMediaLinks`	Logical; if `TRUE` media links will be created (affects only 'eaf' and 'exb' files).
`createFolderOutput`	Logical; if `TRUE` the output folder will be created recursively in case that it does not exist.
`style`	Transcript profile: the name of a profile, the path of a profile file or a profile read with `helper_style_read`. Affects only 'txt' and 'docx' files; `NULL` uses the profile `"act"`.

See Also

export_eaf, export_textgrid, export_exb, export_txt, export_docx, export_rpraat, export_srt

Examples

```
library(act)

# Set destination folder
folderOutput <- tempdir()

# It makes more sense, however, to you define a folder
# that is easier to access on your computer
## Not run:
folderOutput <- "PATH_TO_AN_EMPTY_FOLDER_ON_YOUR_COMPUTER"

## End(Not run)

# Exports all transcript objects in all supported formats
act::corpus_export(x=examplecorpus,
          folderOutput=folderOutput)

# Exports all transcript objects in 'Praat' .TextGrid format
act::corpus_export(x=examplecorpus,
                             folderOutput=folderOutput,
                             formats="textgrid")

# Exports all transcript objects in 'ELAN' .eaf format.
# By default WITH media links
act::corpus_export(x=examplecorpus,
      folderOutput=folderOutput,
      formats="eaf")


# Same same, but now WITHOUT media links.
# Only Media links are only exported that are in
# the '@media' table in the transcript object(s))
act::corpus_export(x=examplecorpus,
      folderOutput=folderOutput,
      formats="eaf",
      createMediaLinks=FALSE)

# Exports in 'ELAN' .eaf and Praat' .TextGrid format
act::corpus_export(x=examplecorpus,
                               folderOutput=folderOutput,
                               formats=c("eaf", "textgrid"))
```

`corpus_import` *Import annotation files into corpus object*

Description

Scans all path specified in if `x@paths.annotation.files` for annotation files. Supported file formats will be loaded as transcript objects into the corpus object. All previously loaded transcript objects will be deleted.

Usage

```
corpus_import(
  x,
  createNormalization = TRUE,
  createFulltext = TRUE,
  assignMedia = TRUE,
  cureTranscripts = TRUE,
  verbose = TRUE
)
```

Arguments

x	Corpus object.
createNormalization	Logical; if TRUE the normalized content will be created.
createFulltext	Logical; if TRUE full text will be created.
assignMedia	Logical; if TRUE the folder(s) specified in @paths.media.files of your corpus object will be scanned for media.
cureTranscripts	Logical; if TRUE transcripts are cured (cleaned up) after loading. Default is TRUE.
verbose	Logical; if TRUE progress and result messages are printed to the console. Default is TRUE.

Details

If assignMedia=TRUE the paths defined in x@paths.media.files will be scanned for media files. Based on their file names the media files and annotations files will be matched. Only the the file types set in options()$act.media.fileformats.audio and options()$act.media.fileformats.video will be recognized. You can modify these options to recognize other media types.

See @import.results of the corpus object to check the results of importing the files. To get a detailed overview of the corpus object use act::info(x), for a summary use act::info_summarized(x).

Value

Corpus object.

See Also

corpus_new, examplecorpus

Examples

```
library(act)

# The example files that come with the act library are located here:
path <- system.file("extdata", "examplecorpus", package="act")

# This is the examplecorpus object that comes with the library
```

```
examplecorpus

# Make sure that the input folder of the example corpus object is set correctly
examplecorpus@paths.annotation.files <- path
examplecorpus@paths.media.files <- path

# Load annotation files into the corpus object (again)
examplecorpus <- act::corpus_import(x=examplecorpus)

# Creating the full texts may take a long time.
# If you do NOT want to create the full texts immediately use the following command:
examplecorpus <- act::corpus_import(x=examplecorpus, createFulltext=FALSE )
```

`corpus_merge` *Merge corpora into a target corpus*

Description

Combines one or several source corpora into a target corpus. The target corpus (`x`) is the winner for all configuration slots (`normalization.matrix`, `import.*`). Source transcripts are added to the target's transcript list.

Usage

```
corpus_merge(x, insert, conflictTranscripts = c("error", "skip", "overwrite"))
```

Arguments

`x`	Corpus object; the target corpus that receives the merged content.
`insert`	Either a corpus object or a list of corpus objects to merge into `x`.
`conflictTranscripts`	Character; how to handle name conflicts between target and source transcripts. One of `"error"` (abort, default), `"skip"` (keep target, ignore source), `"overwrite"` (replace target with source).

Details

If a source corpus has a different `normalization.matrix` than the target, its transcripts get their `normalization.signature` and `fulltext.signature` invalidated, so that the next call to `transcripts_update_normaliz` and `transcripts_update_fulltexts()` will re-normalize and re-create the fulltexts using the target's matrix.

Value

Corpus object.

corpus_new *Create a new corpus object*

Description

Create a new corpus object and loads annotation files. Currently 'ELAN' .eaf, 'EXMARaLDA .exb and 'Praat' .TextGrid files are supported.

The parameter `pathsAnnotationFiles` defines where the annotation files are located. If `skipDoubleFiles=TRUE` duplicated files will be skipped, otherwise the will be renamed. If `importFiles=TRUE` the corpus object will be created but files will not be loaded. To load the files then call corpus_import.

Usage

```
corpus_new(
  pathsAnnotationFiles = NULL,
  pathsMediaFiles = NULL,
  name = "New Corpus",
  importFiles = TRUE,
  skipDoubleFiles = TRUE,
  skipDoubleFilesDatePattern = "",
  createNormalization = TRUE,
  createFulltext = TRUE,
  assignMedia = TRUE,
  pathNormalizationMatrix = NULL,
  namesInclude = character(),
  namesExclude = character(),
  namesExtractPatterns = character(),
  namesSearchPatterns = character(),
  namesSearchReplacements = character(),
  namesToUpper = FALSE,
  namesToLower = FALSE,
  namesTrim = TRUE,
  namesDefault = "no_name",
  cureTranscripts = TRUE,
  verbose = TRUE
)
```

Arguments

`pathsAnnotationFiles`
: Vector of character strings; paths to annotations files or folders that contain annotation files.

`pathsMediaFiles`
: Vector of character strings; paths to media files or folders that contain media files.

`name`
: Character string; name of the corpus to be created.

`importFiles` Logical; if `TRUE` annotation files will be imported immediately when the function is called, if `FALSE` corpus object will be created without importing the annotation files.

`skipDoubleFiles`
Logical; if `TRUE` transcripts with the same names will be skipped (only one of them will be added), if `FALSE` transcripts will be renamed to make the names unique.

`skipDoubleFilesDatePattern`
Character string; regular expression matching a date (optionally followed by a version letter) in the file name. When set and `skipDoubleFiles = "warn_keep_newest"`, the newest of several same-named files is chosen by this date in the file name instead of by the file modification time. Leave empty (default) to keep the modification-time behaviour.

`createNormalization`
Logical; if `TRUE` the normalized content will be created.

`createFulltext` Logical; if `TRUE` full text will be created.

`assignMedia` Logical; if `TRUE` the folder(s) specified in `@paths.media.files` of your corpus object will be scanned for media.

`pathNormalizationMatrix`
Character string; path to the replacement matrix used for normalizing the annotations; if argument left open, the default normalization matrix of the package will be used.

`namesInclude` Character strings; Only files matching this regular expression will be imported into the corpus.

`namesExclude` Character strings; Files matching this regular expression will be skipped and not imported into the corpus.

`namesExtractPatterns`
Vector of character strings; Only the part of the file name matching these expressions will be taken as trasncript name.

`namesSearchPatterns`
Vector of character strings; Search pattern as regular expression. Leave empty for no search-replace in the names.

`namesSearchReplacements`
Vector of character strings; Replacements for search. Leave empty for no search-replace in the names.

`namesToUpper` Logical; Convert transcript names all to upper case.

`namesToLower` Logical; Convert transcript names all to lower case.

`namesTrim` Logical; Remove leading and trailing spaces in names.

`namesDefault` Character string; Default value for empty transcript names (e.g., resulting from search-replace operations)

`cureTranscripts`
Logical; if `TRUE` (default) imported transcripts are automatically cured via transcripts_cure_single. Set to `FALSE` to skip the cure step.

`verbose` Logical; if `TRUE` informational messages (e.g., cure summary) are shown. Set to `FALSE` to suppress them.

Details

The parameter `pathsMediaFiles` defines where the corresponding media files are located. If `assignMedia=TRUE` the paths defined in `x@paths.media.files` will be scanned for media files and will be matched to the transcript object based on their names. Only the the file types set in `options()$act.media.fileformats.audio` and `options()$act.media.fileformats.video` will be recognized. You can modify these options to recognize other media types.

See `@import.results` of the corpus object to check the results of importing the files. To get a detailed overview of the corpus object use `act::info(x)`, for a summary use `act::info_summarized(x)`.

Value

Corpus object.

See Also

corpus_import, examplecorpus

Examples

```
library(act)

# The example files that come with the act library are located here:
path <- system.file("extdata", "examplecorpus", package="act")

# The example corpus comes without media files.
# It is recommended to download a full example corpus also including the media files.
# You can use the following commands.
## Not run:
   path <- "EXISTING_FOLDER_ON_YOUR_COMPUTER/examplecorpus"
   temp <- tempfile()
   download.file(options()$act.examplecorpusURL, temp)
   unzip(zipfile=temp, exdir=path)

## End(Not run)

# The following command creates a new corpus object
mycorpus <- act::corpus_new(name = "mycorpus",
 pathsAnnotationFiles = path,
 pathsMediaFiles = path)

# Get a summary
mycorpus
```

---

examplecorpus *Example corpus for the act package*

Description

Example corpus with data loaded from the example annotations files that come with the package

Usage

```
data(examplecorpus)
```

Format

An object of class "corpus"

Details

The corpus comes without media files. To try it with its media (about 64 MB, downloaded once from GitHub, https://github.com/oliverehmer/act_examplecorpus/) use examplecorpus_media: `x <- act::examplecorpus_media()`. The paths in the corpus are relative to the example folder `system.file("extdata", "examplecorpus", package = "act")`. It is built by `data-raw/examplecorpus_build.R` of the package sources.

Source

- GAT: Ehmer, Oliver/Satti, Luis Ignacio/Martinez, Angelita/Pfaender, Stefan (2019): Un sistema para transcribir el habla en la interaccion: GAT 2.0 Gespraechsforschung - Online-Zeitschrift zur verbalen Interaktion (www.gespraechsforschung-ozs.de) 20, 64-114. http://www.gespraechsforschung-online.de/2019.html
- SYNC: Ehmer, Oliver (2020, in press): Synchronization in demonstrations. Multimodal practices for instructing body knowledge. Linguistics Vanguard. https://www.degruyter.com/view/journals/lingvan/lingvan-overview.xml

See Also

, corpus_new, corpus_import

Examples

```
library(act)

# Summary of the data in the corpus
examplecorpus

# Summary of the data in the second transcript in the corpus
examplecorpus@transcripts[[2]]

## Not run:
# The example corpus with its media files (downloaded once, about 64 MB)
x <- act::examplecorpus_media()
x@transcripts[[1]]@media

## End(Not run)
```

examplecorpus_media *Example corpus with its media files*

Description

Returns the example corpus (examplecorpus) with its media files linked. The media (about 64 MB of wav and mp4 files) are not part of the package: at the first call they are downloaded from GitHub (option `act.examplecorpusURL`) into a folder of act on this computer (`tools::R_user_dir("act", "data")`); later calls use them from there without a download.

Usage

```
examplecorpus_media(folder = NULL, download = TRUE, verbose = TRUE)
```

Arguments

`folder`	Character string; a folder that holds the media files of the example corpus (e.g. a download of your own). `NULL` (default): the folder of act on this computer, downloaded when it is empty.
`download`	Logical; if `FALSE`, nothing is downloaded (the media that are found are linked).
`verbose`	Logical; if `TRUE`, progress is shown.

Value

Corpus object: the example corpus with its media files.

Examples

```
## Not run:
library(act)
x <- act::examplecorpus_media()
x@transcripts[[1]]@media

## End(Not run)
```

export_create_pfsx *Create an 'ELAN' .pfsx preferences file*

Description

Creates a .pfsx preferences file next to an .eaf file. The .pfsx file makes 'ELAN' select and zoom to a time region when the .eaf file is opened. The .pfsx file is written to the same location as the .eaf file, with the extension replaced by .pfsx.

Usage

```
export_create_pfsx(
  path.out.eaf,
  startSec,
  endSec,
  tierName = NULL,
  overwrite = FALSE
)
```

Arguments

path.out.eaf	Character string; path to the .eaf file. The .pfsx file will be created at the same location with the extension replaced by .pfsx.
startSec	Double; start of the selection in 'ELAN', in seconds.
endSec	Double; end of the selection in 'ELAN', in seconds.
tierName	Character string, optional; name of the tier to select in the 'ELAN' grid view.
overwrite	Logical; if TRUE an existing .pfsx file will be overwritten. If FALSE (default) an existing .pfsx file is kept.

Value

Invisibly returns NULL. The function is called for its side effect of creating a .pfsx file.

---

export_docx *Export print transcript in .docx format*

Description

Writes a print transcript as a Word file. Everything about its look comes from a transcript profile (parameter style): mode (gat or score), width, line numbers, speaker acronyms, tier and character styles, the Word template and the advanced engine settings. Profiles are JSON files; the ones shipped with act are listed by helper_style_list(), and helper_style_read() reads one for inspection or modification.

Usage

```
export_docx(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  insertArrowStartsec = NA_real_,
  insertArrowEndsec = NA_real_,
  insertArrowTierName = NA_character_,
  headerPreface = NULL,
```

```
  headerTitle = NULL,
  headerSubtitle = NULL,
  headerDescription = NULL,
  headerInsertSource = TRUE,
  layerOrder = NULL,
  report = FALSE,
  pathReport = NULL,
  mainTierNames = NULL,
  alignChars = NULL,
  alignModes = NULL,
  style = NULL
)
```

Arguments

t	Transcript object.
pathOutput	Character string; path where to save the transcript.
filterTierNames	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
filterSectionStartsec	Double; start of selection in seconds.
filterSectionEndsec	Double; end of selection in seconds.
insertArrowStartsec	Numeric; start time (seconds) of the hit annotation for arrow placement. The annotation is marked with the arrow of the profile; all lines get room for it before the line number. Used with `insertArrowEndsec` and `insertArrowTierName` to locate the annotation by time and tier. If `NA`, no arrow is placed.
insertArrowEndsec	Numeric; end time (seconds) of the hit annotation for arrow placement.
insertArrowTierName	Character string; tier name of the hit annotation for arrow placement.
headerPreface	Character string; text used as preface before title.
headerTitle	Character string; text used as title.
headerSubtitle	Character string; text used as sub title.
headerDescription	Character string; text used as description after sub title.
headerInsertSource	Logical; if `TRUE` standard information about the source and location of the sequence will be inserted after the heading.
layerOrder	Vector of character strings; order of the multimodal layers within a block. `NULL` keeps the tier order of the annotation file.
report	Logical; if `TRUE` and `pathOutput` is set, an alignment report is written next to the output file.

pathReport	Character string; explicit path for the alignment report.
mainTierNames	Vector of character strings; exact names of the tiers to treat as main tiers. NULL derives the main flag from the tier styles of the profile; without any tier style every tier counts as a main tier.
alignChars	Named vector of character strings; anchor characters per layer tier (names = tier names, values = the characters). NULL derives them from the tier styles of the profile.
alignModes	Named vector of character strings; alignment mode per layer tier ("bracket" or "point"). NULL derives the mode from the tier styles of the profile.
style	Transcript profile: the name of a profile, the path of a profile file or a profile read with helper_style_read. NULL uses the profile "act".

Details

The Word styles of the profile are looked up in the template of the profile (word$file); with word$look = "profile" styles missing there are created from the profile.

Value

Officer doc; transcript as object from library officer.

See Also

corpus_export, export_eaf, export_exb, export_rpraat, export_srt, export_textgrid, vignette("export_docx_styles", package = "act")

Examples

```
library(act)

# Get a transcript
t <- examplecorpus@transcripts[[1]]

# Create print transcript
printtranscript <- act::export_docx (t=t)

# Display on screen
printtranscript
```

---

export_eaf *Export 'ELAN' .eaf file*

Description

Advice: In most situations it is more convenient to use act::corpus_export for exporting annotation files.

Usage

```
export_eaf(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  createMediaLinks = TRUE,
  mediaMaxFiles = NULL,
  mediaWriteOffset = FALSE
)
```

Arguments

t	Transcript object; transcript to be exported.
pathOutput	Character string; path where .eaf file will be saved.
filterTierNames	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
filterSectionStartsec	Double; start of selection in seconds.
filterSectionEndsec	Double; end of selection in seconds.
createMediaLinks	Logical; if TRUE media links will be created.
mediaMaxFiles	Integer or NULL; maximum number of media files to link (NULL = all).
mediaWriteOffset	Logical; if TRUE a negative TIME_ORIGIN attribute (in milliseconds) is written for clip media (rows with kind == "clip" and startsec > 0), derived from startsec. Default FALSE (no offset). Note: clip media are currently not linked at all in the exported .eaf, because 'ELAN' does not support negative media offsets - only full recordings (offset 0) are linked. This parameter takes effect once clip linking is re-enabled in the code.

Details

The .eaf file will be written to the file specified in pathOutput. If pathOutput is left empty, the function will return the contents of the .eaf itself.

Value

Contents of the .eaf file (only if pathOutput is left empty)

See Also

corpus_export, export_exb, export_txt, export_rpraat, export_srt, export_textgrid, export_docx

Examples

```
library(act)

# Get the transcript you want to export
t <- examplecorpus@transcripts[[1]]

# Create temporary file path
path <- tempfile(pattern = t@name, tmpdir = tempdir(), fileext = ".eaf")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                  paste(t@name, ".eaf", sep=""))

## End(Not run)

# Export WITH media links
act::export_eaf(t=t, pathOutput=path)

# Export WITHOUT media links
act::export_eaf(t=t, pathOutput=path, createMediaLinks = FALSE)
```

---

export_edl *Export .edl marker file for ''Davinci Resolve'*

Description

Advice: In most situations it is more convenient to use `act::corpus_export` for exporting annotation files.

Usage

```
export_edl(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  fps = 50
)
```

Arguments

`t`	Transcript object; transcript to be saved.
`pathOutput`	Character string; path where .edl will be saved.

`filterTierNames`
: Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.

`filterSectionStartsec`
: Double; start of selection in seconds.

`filterSectionEndsec`
: Double; end of selection in seconds.

`fps`
: Double; Frame rate per seconds of your project, e.g. 60, 50, 30, 29

Details

Creates a 'event list text' .edl file with definitions for Blackmagic DaVinci Resolve It will be written to the file specified in `pathOutput`. If `pathOutput` is left empty, the function will return the contents of the .edl itself.

Value

Contents of the .edl file (only if `pathOutput` is left empty)

See Also

`corpus_export`, export_eaf, export_exb, export_srt, export_txt, export_docx, export_rpraat, export_textgrid

Examples

```
library(act)

# Get the transcript you want to export
t <- examplecorpus@transcripts[[1]]

# Create temporary file path
path <- tempfile(pattern = t@name, tmpdir = tempdir(),
                 fileext = ".srt")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                    paste(t@name, ".srt", sep=""))

## End(Not run)

# Export
act::export_srt(t=t, pathOutput=path)
```

export_exb *Export 'EXMARaLDA' .exb file*

Description

Advice: In most situations it is more convenient to use `act::corpus_export` for exporting annotation files.

Usage

```
export_exb(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  createMediaLinks = TRUE
)
```

Arguments

`t`	Transcript object; transcript to be exported.
`pathOutput`	Character string; path where .exb file will be saved.
`filterTierNames`	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
`filterSectionStartsec`	Double; start of selection in seconds.
`filterSectionEndsec`	Double; end of selection in seconds.
`createMediaLinks`	Logical; if `TRUE` media links will be created.

Details

The .exb file will be written to the file specified in `pathOutput`. If `pathOutput` is left empty, the function will return the contents of the .exb itself.

Value

Contents of the .exb file (only if `pathOutput` is left empty)

See Also

corpus_export, export_eaf, export_txt, export_docx, export_rpraat, export_srt, export_textgrid

Examples

```
library(act)

# Get the transcript you want to export
t <- examplecorpus@transcripts[[1]]

# Create temporary file path
path <- tempfile(pattern = t@name, tmpdir = tempdir(), fileext = ".exb")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                  paste(t@name, ".exb", sep=""))

## End(Not run)

# Export WITH media links
act::export_exb(t=t, pathOutput=path)

# Export WITHOUT media links
act::export_exb(t=t, pathOutput=path, createMediaLinks = FALSE)
```

---

export_rpraat *Export 'rPraat' TextGrid object*

Description

Advice: In most situations it is more convenient to use `act::corpus_export` for exporting annotation files.

Usage

```
export_rpraat(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL
)
```

Arguments

`t`	Transcript object; transcript to be converted.
`filterTierNames`	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.

`filterSectionStartsec`
: Double; start of selection in seconds.

`filterSectionEndsec`
: Double; end of selection in seconds.

Details

This function is to create compatibility with the `rPraat` package. It converts an act transcript to a rPraat TextGrid object.

Credits: Thanks to Tomáš Bořil, the author of the rPraat package, for commenting on the exchange functions.

Value

rPraat TextGrid object

See Also

import_rpraat, corpus_export, export_eaf, export_exb, export_txt, export_docx, export_srt, export_textgrid

Examples

```
library(act)

# Convert
rpraat.tg <- act::export_rpraat(t=examplecorpus@transcripts[[1]])

# Now you can use the object in the rPraat pachage.
# For instance you can plot the TextGrid
## Not run:
 rPraat::tg.plot(rpraat.tg)

## End(Not run)
```

---

export_srt *Export .srt subtitle file*

Description

Advice: In most situations it is more convenient to use `act::corpus_export` for exporting annotation files.

Usage

```
export_srt(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  speakerShow = TRUE,
  speakerWidth = 3,
  speakerEnding = ":"
)
```

Arguments

t	Transcript object; transcript to be saved.
pathOutput	Character string; path where .srt will be saved.
filterTierNames	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
filterSectionStartsec	Double; start of selection in seconds.
filterSectionEndsec	Double; end of selection in seconds.
speakerShow	Logical; if TRUE name of speaker will be shown before the content of the annotation.
speakerWidth	Integer; width of speaker abbreviation, -1 for full name without shortening.
speakerEnding	Character string; string that is added at the end of the speaker name.

Details

Creates a 'Subrip title' .srt subtitle file. It will be written to the file specified in pathOutput. If pathOutput is left empty, the function will return the contents of the .srt itself.

Value

Contents of the .srt file (only if pathOutput is left empty)

See Also

corpus_export, export_eaf, export_exb, export_txt, export_docx, export_rpraat, export_textgrid

Examples

```
library(act)

# Get the transcript you want to export
t <- examplecorpus@transcripts[[1]]
```

```
# Create temporary file path
path <- tempfile(pattern = t@name, tmpdir = tempdir(),
                 fileext = ".srt")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                    paste(t@name, ".srt", sep=""))

## End(Not run)

# Export
act::export_srt(t=t, pathOutput=path)
```

`export_textgrid` *Export 'Praat' .TextGrid file*

Description

Advice: In most situations it is more convenient to use `act::corpus_export` for exporting annotation files.

Usage

```
export_textgrid(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL
)
```

Arguments

`t`	Transcript object; transcript to be saved.
`pathOutput`	Character string; path where .TextGrid will be saved.
`filterTierNames`	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
`filterSectionStartsec`	Double; start of selection in seconds.
`filterSectionEndsec`	Double; end of selection in seconds.

Details

The .TextGrid file will be written to the file specified in `pathOutput`. If `pathOutput` is left empty, the function will return the contents of the .TextGrid itself.

Value

Contents of the .TextGrid file (only if `pathOutput` is left empty)

See Also

corpus_export, export_eaf, export_exb, export_txt, export_docx, export_rpraat, export_srt

Examples

```
library(act)

# Get the transcript you want to export
t <- examplecorpus@transcripts[[1]]

# Create temporary file path
path <- tempfile(pattern = t@name, tmpdir = tempdir(),
                 fileext = ".TextGrid")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                  paste(t@name, ".TextGrid", sep=""))

## End(Not run)

# Export
act::export_textgrid(t=t, pathOutput=path)
```

---

export_txt *Export print transcript in .txt format*

Description

Writes a print transcript as plain text. Its layout comes from a transcript profile (parameter `style`); see export_docx.

Usage

```
export_txt(
  t,
  pathOutput = NULL,
  filterTierNames = NULL,
```

```
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  insertArrowStartsec = NA_real_,
  insertArrowEndsec = NA_real_,
  insertArrowTierName = NA_character_,
  headerPreface = NULL,
  headerTitle = NULL,
  headerSubtitle = NULL,
  headerDescription = NULL,
  headerInsertSource = TRUE,
  collapse = TRUE,
  layerOrder = NULL,
  report = FALSE,
  pathReport = NULL,
  mainTierNames = NULL,
  alignChars = NULL,
  alignModes = NULL,
  style = NULL
)

export_printtranscript(...)
```

Arguments

`t`	Transcript object.
`pathOutput`	Character string; path where to save the transcript.
`filterTierNames`	Vector of character strings; names of tiers to be included. If left unspecified, all tiers will be exported.
`filterSectionStartsec`	Double; start of selection in seconds.
`filterSectionEndsec`	Double; end of selection in seconds.
`insertArrowStartsec`	Numeric; start time (seconds) of the hit annotation for arrow placement. The annotation is marked with the arrow of the profile; all lines get room for it before the line number. Used with `insertArrowEndsec` and `insertArrowTierName` to locate the annotation by time and tier. If `NA`, no arrow is placed.
`insertArrowEndsec`	Numeric; end time (seconds) of the hit annotation for arrow placement.
`insertArrowTierName`	Character string; tier name of the hit annotation for arrow placement.
`headerPreface`	Character string; text used as preface before title.
`headerTitle`	Character string; text used as title.
`headerSubtitle`	Character string; text used as sub title.
`headerDescription`	Character string; text used as description after sub title.

`headerInsertSource`
: Logical; if `TRUE` standard information about the source and location of the sequence will be inserted after the heading.

`collapse`
: Logical; if `FALSE` a vector will be created, each element corresponding to one annotation. if `TRUE` a single string will be created, collapsed by linebreaks \n.

`layerOrder`
: Vector of character strings; order of the multimodal layers within a block. `NULL` keeps the tier order of the annotation file.

`report`
: Logical; if `TRUE` and `pathOutput` is set, an alignment report is written next to the output file.

`pathReport`
: Character string; explicit path for the alignment report.

`mainTierNames`
: Vector of character strings; exact names of the tiers to treat as main tiers. `NULL` derives the main flag from the tier styles of the profile; without any tier style every tier counts as a main tier.

`alignChars`
: Named vector of character strings; anchor characters per layer tier (names = tier names, values = the characters). `NULL` derives them from the tier styles of the profile.

`alignModes`
: Named vector of character strings; alignment mode per layer tier (`"bracket"` or `"point"`). `NULL` derives the mode from the tier styles of the profile.

`style`
: Transcript profile: the name of a profile, the path of a profile file or a profile read with `helper_style_read`. `NULL` uses the profile `"act"`.

`...`
: Arguments passed to `export_txt()`.

Value

Character string; transcript as text.

See Also

corpus_export, export_eaf, export_exb, export_rpraat, export_srt, export_textgrid, export_docx

export_txt

Examples

```
library(act)

# Get a transcript
t <- examplecorpus@transcripts[[1]]

# Create print transcript
printtranscript <- act::export_txt (t=t)

# Display on screen
cat(printtranscript)
```

helper_annotationfile_read *Read an annotation file with automatic encoding detection*

Description

High-level reader that combines encoding detection with decoding, UTF-8 BOM stripping, optional XML declaration check (for EAF/EXB files), and optional header verification (e.g. TextGrid).

Usage

```
helper_annotationfile_read(
  filePath,
  expectedHeader = NULL,
  fileType = NULL,
  verbose = TRUE
)
```

Arguments

filePath	Character string; path to the file.
expectedHeader	Character vector; required prefixes of the first N lines, e.g. c('File type = "ooTextFile"', 'Object class = "TextGrid"'). If NULL, no header check is performed.
fileType	Character string; one of "textgrid", "eaf", "exb", "srt", or NULL. Controls XML declaration detection for EAF/EXB files even when line 1 does not start with "<?xml".
verbose	Logical; if TRUE (default) warnings are emitted via cli::cli_warn(). If FALSE, only the return value contains the diagnostics.

Details

The function reads the file with the detected encoding, removes a leading UTF-8 BOM character from line 1 if present, and (for XML files) extracts the declared encoding from the <?xml ... ?> header and compares it to the detected one. Mismatches are reported as a warning when verbose = TRUE but the file is still read with the detected encoding.

Value

A list with elements:

lines Character vector with file contents, or NULL on error.

encoding_detected Detected encoding (see helper_encoding_file_detect).

encoding_declared Encoding declared in the XML header, or NA_character_.

encoding_match Logical; whether declared and detected encoding belong to the same family, or NA if not applicable.

`status` "ok" or "error".

`message` Collected diagnostic messages, separated by "; ".

`hash` Fingerprint of the bytes read (see `helper_file_hash_compute`), `NA` when nothing was read.

`state` State of the file as it was read (see `helper_file_state`), an empty list when nothing was read.

See Also

`helper_encoding_file_detect`, `helper_encoding_names_equivalent`

`helper_annotations_columns_order` *Reorder columns of an annotations data.frame*

Description

Ensures consistent column ordering: standard annotation columns first, then layer columns, then search/concordance columns (if present).

Usage

```
helper_annotations_columns_order(df)
```

Arguments

`df`	Data.frame; annotations or search results data.frame.

Value

Data.frame with reordered columns.

`helper_audio_filter_build` *Build a combined FFmpeg audio filter string*

Description

Constructs the audio filter arguments for an FFmpeg command that optionally combines loudness normalization (`loudnorm`) and audio anonymization (`silence`, `beep`, or `noise`) over specified time ranges. The filters are built by helper_audio_filter_parts; the anonymization comes first, the normalization last. The beep and the noise are generated inside the filter graph.

Usage

```
helper_audio_filter_build(audio_normalize = FALSE, audio_anonymize = NULL)
```

Arguments

audio_normalize
: Logical. If TRUE and option act.ffmpeg.audio.normalize is TRUE, the loudness is normalized. Default is FALSE.

audio_anonymize
: data.frame with columns start and end (numeric, seconds relative to cut start), or NULL for no anonymization. Default is NULL.

Details

Audio filter and beep frequency are read from options: act.ffmpeg.audio.anony_filter, act.ffmpeg.audio.anony_bee act.ffmpeg.audio.normalize.

Value

A named list with:

type "none", "simple", or "filter_complex"

af_string For "simple": the full -af "..." argument string. "" otherwise.

fc_parts For "filter_complex": character vector of filter graph segments. character(0) otherwise.

out_label For "filter_complex": the final audio stream label, e.g. "[aout]". "" otherwise.

map_a For "filter_complex": the -map argument, e.g. "-map \"[aout]\"". "" otherwise.

---

helper_audio_filter_parts

Build the FFmpeg sound filters: channel, loudness, anonymization

---

Description

One builder for the sound filters of every export (viewer clips and sound files, act cuts): the channel selection, the loudness normalization and the anonymization of time ranges by beep, noise, mute or a voice distortion. The result is either a simple chain (af, for -af) or a filter graph (graph) that reads [0:a] and writes [aout]. The beep and the noise are generated inside the graph, so no further input is needed.

Usage

```
helper_audio_filter_parts(
  channel = "stereo",
  mono = FALSE,
  normalize = FALSE,
  ranges = NULL,
  mode = .audio_default_mode(),
  strength = 0.8,
  distort = NULL,
  beepFreq = getOption("act.ffmpeg.audio.anony_beep_freq", 800)
)
```

Arguments

channel	Character; "stereo" (default) keeps the channels, "left" / "right" put the left / right channel on both sides (a stereo file with the same signal twice).
mono	Logical; TRUE writes one channel (the mix of both, or the chosen side).
normalize	Logical; TRUE applies the loudness normalization (nothing if the option act.ffmpeg.audio.normalize is FALSE).
ranges	Data frame with columns start and end (seconds from the start of the output) or NULL: the ranges to anonymize. NULL switches the anonymization off; an empty data frame keeps it on without ranges (a distortion with scope "all" still applies).
mode	Character; "beep", "noise", "mute" or "distort". Default is the option act.ffmpeg.audio.anony_fi ("silence" is read as "mute"), else "beep".
strength	Numeric between 0 and 1; loudness of beep and noise.
distort	List or NULL; the voice distortion effects (pitch, vibrato, tremolo, robot, flanger, each a list with on and its parameters) and scope ("segments" or "all").
beepFreq	Numeric; frequency of the beep in Hz. Default is the option act.ffmpeg.audio.anony_beep_freq.

Details

Order: channel first, then the anonymization, then the loudness (in front of the timed volume the loudness filter shifted its time base). The timed volume is evaluated in blocks of 64 samples (1.5 ms), so a range starts and ends where it is annotated. Beep and noise are mixed in the channel layout of the output (stereo, or mono with mono = TRUE).

Value

A list with af (character or NULL) and graph (character or NULL); at most one of them is set.

See Also

helper_ffmpeg_args_audio, helper_ffmpeg_args_clip

Examples

```
act::helper_audio_filter_parts(channel = "left",
                               ranges = data.frame(start = 1, end = 2.5))
```

helper_corpus_export_cured *Helper: Export cured transcripts*

Description

Exports only those transcripts from a corpus that were modified by transcripts_cure_single (either during import or via an explicit cure call). A transcript is considered cured if its most recent history entry of type `transcripts_cure_single` contains at least one non-zero correction counter.

Usage

```
helper_corpus_export_cured(x, folderOutput, formats = "textgrid")
```

Arguments

`x`	Corpus object.
`folderOutput`	Character string; path to the output folder.
`formats`	Character vector; annotation formats to export. Defaults to `"textgrid"`. See corpus_export for supported formats.

Details

All exported files are written flat into `folderOutput`, without sub-folders.

Value

Invisibly, a character vector with the names of the exported transcripts.

See Also

transcripts_cure_single, corpus_export

helper_cutlist_lines *Write the lines of a terminal script (cut list)*

Description

One writer for every line of the render scripts that act and iclo save with helper_cutlist_save: ffmpeg and exiftool calls, folders, deletions, comments, messages and calls of other scripts. The caller collects the parts of each line (for ffmpeg the argument vector of helper_ffmpeg_args, helper_ffmpeg_args_clip or helper_ffmpeg_args_audio); this function quotes them for the shell of the target system. Arguments that need no quoting stay as they are.

Usage

```
helper_cutlist_lines(
  entries,
  os = c("mac", "win"),
  executable = getOption("act.cutlist.ffmpeg", "ffmpeg"),
  inputVariable = NULL,
  header = FALSE
)
```

Arguments

`entries`	List of entries, each a list with `type` and `args` (character vector).
`os`	Character; `"mac"` (POSIX sh, also Linux) or `"win"` (cmd).
`executable`	Character; how the script calls ffmpeg. Default is the option `act.cutlist.ffmpeg` (`"ffmpeg"`, found via the PATH of the machine that runs the script).
`inputVariable`	Character or `NULL`; an ffmpeg argument equal to this value is written as the script variable of the input file (`$PATH_INPUT` / `%PATH_INPUT%`).
`header`	Logical; `TRUE` starts a Windows script with `@echo off` and the UTF-8 code page. No effect for `"mac"`.

Details

Entry types (`type`) and their `args`:

- `"ffmpeg"`: the ffmpeg arguments (without the executable).
- `"exiftool"`: the exiftool arguments, as from helper_metadata_exif_argv; the call is skipped when exiftool is not installed.
- `"mkdir"`: folders to create (with parents).
- `"rm"`: files to delete (no error when missing).
- `"comment"`: one text, written as a comment line.
- `"echo"`: one text, printed when the script runs.
- `"script"`: one script file to run.
- `"blank"`: an empty line.

On Windows paths starting with `/` are written with backslashes and `%` is doubled.

Value

Character vector with the script lines.

See Also

helper_cutlist_save

Examples

```
act::helper_cutlist_lines(list(
  list(type = "comment", args = "==== clip 1 ===="),
  list(type = "mkdir", args = "/tmp/my clips"),
 list(type = "ffmpeg", args = c("-i", "/tmp/in.mp4", "-t", "2", "-y", "/tmp/my clips/out.mp4"))),
  os = "mac")
```

---

helper_cutlist_save *Helper: Save cutlist*

Description

Saves FFMPEG cut list fpr mac or windows For windows: simly saves to a .cmd file For mac: saves as a shell script and makes it executable (also adds "#!/bin/sh")

Usage

```
helper_cutlist_save(
  cutlistMac = NULL,
  cutlistWin = NULL,
  outFolder,
  outFilename
)
```

Arguments

cutlistMac	Character string; Content if file, if NULL no file will be saved.
cutlistWin	Character string; Content if file, if NULL no file will be saved.
outFolder	Character string; Destination folder.
outFilename	Character string; Destination filename

Examples

```
library(act)

# --- Create two tier tables from scratch
tierTable1 <- act::helper_tiers_new_table(c("a","b","c","d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable2 <- act::helper_tiers_new_table(c("a","b","x","y"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable3 <- act::helper_tiers_merge_tables(tierTable1,tierTable2)
tierTable3
```

`helper_encoding_file_detect` *Detect the text encoding of an annotation file*

Description

Determines the encoding of a text file deterministically using a four-step strategy:

Usage

```
helper_encoding_file_detect(filePath)
```

Arguments

`filePath`	Character string; path to the file.

Details

1. BOM detection in the first bytes: UTF-8 BOM (`EF BB BF`), UTF-16 LE BOM (`FF FE`), UTF-16 BE BOM (`FE FF`).
2. NULL-byte guard in the first 1000 bytes. Plain text files never contain `0x00`; UTF-16 without BOM would be full of NULL bytes and would otherwise pass `base::validUTF8()` (because ASCII-range bytes are formally valid UTF-8).
3. UTF-8 validation via `stringi::stri_enc_isutf8()` on the whole file.
4. Fallback to LATIN1 (every byte sequence is valid LATIN1).

Value

A list with three elements:

`encoding` One of `"UTF-8"`, `"UTF-8-BOM"`, `"UTF-16LE"`, `"UTF-16BE"`, `"LATIN1"`, `"UTF-16-no-BOM-likely"`, or `NA_character_` on error.

`confidence` One of `"bom"`, `"validated"`, `"fallback"`, `"unknown"`, or `"error"`.

`message` Human-readable explanation when the result is not `"ok"`.

See Also

`helper_annotationfile_read`

helper_encoding_name_normalize

Normalize an encoding name for synonym-tolerant comparison

Description

Normalizes encoding labels to a canonical family name by lowercasing, removing hyphens/underscores/whitespace, and mapping common aliases.

Usage

```
helper_encoding_name_normalize(x)
```

Arguments

x	Character string; encoding name. NULL, NA or empty string return NA_character_.

Details

Examples:

- "UTF-8" -> "utf8"
- "utf-8-BOM" -> "utf8"
- "ISO-8859-1" -> "latin1"
- "UTF-16LE" -> "utf16"

Value

Character string with the canonical family name, or NA_character_.

See Also

helper_encoding_names_equivalent

helper_encoding_names_equivalent

Test whether two encoding labels refer to the same family

Description

Compares two encoding labels after normalization via helper_encoding_name_normalize. Returns NA when either input is missing or empty.

Usage

```
helper_encoding_names_equivalent(a, b)
```

Arguments

a	Character string; first encoding name.
b	Character string; second encoding name.

Value

Logical or NA.

See Also

helper_encoding_name_normalize

---

helper_ffmpeg_args *Helper: Build the FFmpeg arguments for an image output*

Description

Builds the argument vector (without the program name) for one FFmpeg call that writes a single image (JPG or PNG). All image outputs of act and iclo are built here, so the rules for reading video sources apply in one place:

- The seek is done in two stages: a fast seek to one second before the target in front of the input, then an exact seek of the rest after it (`-ss t-1 -i ... -ss 1`). A single seek in front of the input (`-ss t -i`) can miss the frames that are displayed just before a keyframe but decoded after it (B-frames): it then delivers the keyframe instead. With a filter, the rest is cut by a `trim` at the start of the filter chain instead, so the filter only processes the target frame.
- The seek target is the time as players that honour the edit list of the file show it (ELAN with AVFoundation, browsers): the edit list offset of the video track is added, and the frame that is on screen at that moment is taken (not the next one). With the option `act.media.timeline = "raw"` (ELAN with VLC) the offset is not added. Camera files with B-frames carry an offset of their reorder delay (e.g. 2 frames), cuts made with stream copy may carry any offset.
- Video inputs (mp4, mov, m4a, m4v) get `-ignore_editlist 1 -avoid_negative_ts make_zero` in front of `-i`. Without these flags a file with an edit list delivers a black frame. Image inputs never get them (the image demuxer rejects the option).
- JPG outputs are written with `-q:v` derived from `imageQuality`.

Usage

```
helper_ffmpeg_args(
  output,
  input = NULL,
  startsec = NULL,
  duration = NULL,
  inputsExtra = NULL,
  videoFilter = NULL,
  filterComplex = NULL,
```

```
  filterMap = NULL,
  imageQuality = getOption("act.ffmpeg.image.quality", 100),
  maxHeight = NULL,
  videoOffset = NULL,
  videoFps = NULL
)
```

Arguments

output	Character; path of the image to write. The extension decides the format: jpg/jpeg or png.
input	Character; path of the source file (video or image). Its extension decides whether the edit list flags are set, so the real path must be given (also when the call is written into a cut list).
startsec	Numeric or NULL; seek position in seconds, on the timeline of the file as players show it.
duration	Numeric or NULL; length of the input window in seconds (-t before -i).
inputsExtra	Character vector or NULL; further inputs (e.g. a palette image) appended after the main input.
videoFilter	Character or NULL; a simple filter chain (-vf).
filterComplex	Character or NULL; a filter graph (-filter_complex). Cannot be combined with videoFilter.
filterMap	Character or NULL; output label of the filter graph that is written (-map).
imageQuality	Numeric; JPG quality from 1 (worst) to 100 (best). Default is the option act.ffmpeg.image.quality.
maxHeight	Integer or NULL; scale the image down to this height (never up). Only together with videoFilter or without a filter.
videoOffset	Numeric or NULL; edit list offset of the video track in seconds, e.g. column video.editlist.offset of media_metadata_read. NULL reads it from the file.
videoFps	Numeric or NULL; frame rate of the video, e.g. column video.fps. NULL reads it from the file.

Value

Character vector of FFmpeg arguments.

See Also

helper_ffmpeg_run, helper_ffmpeg_path

Examples

```
act::helper_ffmpeg_args(output = "still.jpg", input = "video.mp4", startsec = 12.5)
```

`helper_ffmpeg_args_audio` *Helper: Build the FFmpeg arguments for a sound file*

Description

Builds the argument vector (without the program name) for one FFmpeg call that writes a sound file (wav, mp3 or m4a) from a sound or video source. The sound of an MP4/MOV source is read with its edit list honoured, as ELAN plays it (with `act.media.timeline = "raw"` it is ignored, as VLC does). Without filter, a wav/aif from a source of the same format and an mp3 from an mp3 are copied as they are; otherwise a wav/aif is written with the bit depth and sample rate of the source, mp3 and m4a are encoded with `act.ffmpeg.audio.bitrate`. A source without sound track is refused with an error.

Usage

```
helper_ffmpeg_args_audio(
  output,
  input,
  startsec,
  duration,
  audioFilter = NULL,
  filterComplex = NULL,
  audioMap = "[aout]",
  audioBitrate = getOption("act.ffmpeg.audio.bitrate", "192k"),
  metadataArgs = NULL
)
```

Arguments

`output`	Character; path of the sound file (wav, aif, aiff, mp3 or m4a).
`input`	Character; path of the source.
`startsec`	Numeric; start in seconds.
`duration`	Numeric; length in seconds.
`audioFilter`	Character or `NULL`; a simple filter chain (`-af`), e.g. `af` of helper_audio_filter_parts.
`filterComplex`	Character or `NULL`; a filter graph reading `[0:a]` and writing `audioMap`, e.g. `graph` of helper_audio_filter_parts.
`audioMap`	Character; output label of `filterComplex`.
`audioBitrate`	Character; bit rate of mp3 and m4a. Default is the option `act.ffmpeg.audio.bitrate`.
`metadataArgs`	Character vector or `NULL`; further output arguments.

Value

Character vector of FFmpeg arguments.

See Also

helper_audio_filter_parts, helper_ffmpeg_args_clip, helper_ffmpeg_run

Examples

```
act::helper_ffmpeg_args_audio(output = "cut.wav", input = "sound.wav",
                              startsec = 12.5, duration = 4)
```

`helper_ffmpeg_args_clip`

Helper: Build the FFmpeg arguments for a video clip

Description

Builds the argument vector (without the program name) for one FFmpeg call that writes a video clip (MP4). The rules of helper_ffmpeg_args apply to the picture: exact two-stage seek, edit list flags for video inputs, and the picture of a clip starts with the frame that players honouring the edit list (ELAN with AVFoundation) show at `startsec`. The sound is read from a second input of the same file (or `audioInput`) with its own seek: its edit list is honoured, so the encoder delay at the start of AAC tracks is skipped like ELAN does, and picture and sound are in sync as in ELAN. An AAC encoder shifts its output by 1024 samples, which no player compensates in a file without edit list; the first 1024 samples are therefore cut at the very end of the sound chain, so the clip plays at the right time. Filters (e.g. an anonymization in a graph) still see the time of the clip, so their ranges stay exact. The sound keeps the sample rate of the source (a loudness filter would otherwise raise it and shift that delay). In a filter graph the sound of the source is therefore addressed as `[0:a]` as usual - it is rewritten to the sound input.

Usage

```
helper_ffmpeg_args_clip(
  output,
  input,
  startsec,
  duration,
  inputsExtra = NULL,
  videoFilter = NULL,
  filterComplex = NULL,
  videoMap = NULL,
  audioMap = NULL,
  audioFilter = NULL,
  maxHeight = getOption("act.ffmpeg.video.max_height"),
  maxWidth = getOption("act.ffmpeg.video.max_width"),
  videoBitrate = getOption("act.ffmpeg.video.bitrate", "8M"),
  videoCrf = getOption("act.ffmpeg.video.crf"),
  keyframeInterval = getOption("act.ffmpeg.video.keyframe_interval", 25),
 audioCodecArgs = c("-c:a", "aac", "-b:a", getOption("act.ffmpeg.audio.bitrate",
```

```
    "192k")),
  metadataArgs = NULL,
  videoOffset = NULL,
  videoFps = NULL,
  audioInput = NULL,
  videoCodecCopy = FALSE,
  withAudio = TRUE,
  videoHardware = getOption("act.ffmpeg.video.hardware", TRUE),
  videoCodecCopyFail = getOption("act.ffmpeg.video.codec.copy.fail", "abort")
)
```

Arguments

`output`	Character; path of the clip (mp4, mov or m4v).
`input`	Character; path of the source video.
`startsec`	Numeric; start in seconds on the timeline of the file as players show it.
`duration`	Numeric; length in seconds.
`inputsExtra`	List or `NULL`; further inputs after the video, each a path or a complete argument vector (e.g. `c("-f", "lavfi", "-t", "3", "-i", "sine")`). Their indices in a filter graph follow the video (1, 2, ...); the sound input comes last.
`videoFilter`	Character or `NULL`; a simple video filter chain.
`filterComplex`	Character or `NULL`; a filter graph using `[0:v]` and `[0:a]`.
`videoMap`	Character or `NULL`; output label of the picture in `filterComplex`.
`audioMap`	Character or `NULL`; output label of the sound in `filterComplex`. `NULL` takes the sound of the source.
`audioFilter`	Character or `NULL`; filter chain for the sound of the source (`-af`), when `audioMap` is `NULL`.
`maxHeight, maxWidth`	Integer or `NULL`; scale the picture down to fit into this height and/or width (never up). Defaults are the options `act.ffmpeg.video.max_height` and `act.ffmpeg.video.max_width`.
`videoBitrate`	Character; video bit rate, used when `videoCrf` is `NULL`: a number with `M` (Mbit/s) or `k` (kbit/s), e.g. `"4M"`, `"8M"`, `"12M"`, `"16M"`. The same value applies to the software and the hardware encoder. Default is the option `act.ffmpeg.video.bitrate`.
`videoCrf`	Numeric or `NULL`; constant rate factor (e.g. 18 or 23, lower is better) instead of a bit rate. Only the software encoder (libx264) knows it: with a value set the clip is always encoded in software. Default is the option `act.ffmpeg.video.crf`.
`keyframeInterval`	Integer; distance of keyframes in frames. Default is the option `act.ffmpeg.video.keyframe_interval`.
`audioCodecArgs`	Character vector; codec arguments of the sound.
`metadataArgs`	Character vector or `NULL`; further output arguments, e.g. from helper_metadata_ffmpeg_argv.
`videoOffset, videoFps`	See helper_ffmpeg_args.
`audioInput`	Character or `NULL`; file the sound is taken from. `NULL` takes it from `input`.

videoCodecCopy Logical; copy the video stream instead of encoding it. Fast, but a copy can only start at a keyframe: the clip starts at the keyframe at or before `startsec` (see helper_ffmpeg_keyframe; cameras often set one every few seconds) and `duration` is extended by that lead, so the clip still ends at `startsec` + `duration`. The sound starts at the same keyframe, picture and sound stay in sync, and the picture starts at 0 without an edit list. No filter and no scaling are possible.

withAudio Logical; `FALSE` writes the picture only. A source without sound track is written without sound as well (a filter graph with a sound part is then refused).

videoHardware Logical; encode with the video encoder of the Mac (`h264_videotoolbox`, several times faster) instead of libx264. Without a Mac or with an ffmpeg that lacks this encoder a warning is shown once and libx264 is used. Windows cut lists always get libx264 (helper_cutlist_lines). Default is the option `act.ffmpeg.video.hardware`.

videoCodecCopyFail
Character; what happens when `videoCodecCopy` is `TRUE` and no keyframe could be found (ffprobe missing or failed, source not readable or not an MP4/MOV file): `"abort"` stops with an error, `"encode"` encodes the clip as with `videoCodecCopy = FALSE` and shows a warning. Default is the option `act.ffmpeg.video.codec.copy.fail`.

Details

With the option `act.ffmpeg.video.exact_timing` (default `TRUE`) an encoded clip with sound is written with `-avoid_negative_ts disabled`: the AAC encoder starts the sound at a negative time, and without this flag the muxer shifts the picture from the second frame on by that amount (about 21 ms) behind the sound. Set the option to `FALSE` if PowerPoint has problems with such clips, e.g. when cropping the picture.

Every clip is written as H.264 with `yuv420p` (PowerPoint plays nothing else), with `-use_editlist 0 -movflags +faststart`, without B-frames (`-bf 0`): with B-frames the picture of a file without edit list starts two frames after the sound. The frame rate is set explicitly (`-r`): after the trim the encoder would otherwise assume the time base as frame rate and, with a bit rate, write grey frames at the start.

Value

Character vector of FFmpeg arguments.

See Also

helper_ffmpeg_args, helper_ffmpeg_run

Examples

```
act::helper_ffmpeg_args_clip(output = "cut.mp4", input = "video.mp4",
                             startsec = 12.5, duration = 4)
```

`helper_ffmpeg_check` *Helper: Check whether FFmpeg or ffprobe can run*

Description

Runs the executable resolved by helper_ffmpeg_path (or helper_ffprobe_path) with `-version` and reports whether it works. An executable can exist and still not run, e.g. an FFmpeg built for Intel processors on a Mac with Apple silicon that has no Rosetta. The result is remembered for the session.

Usage

```
helper_ffmpeg_check(executable = c("ffmpeg", "ffprobe"))
```

Arguments

`executable`	Character string; `"ffmpeg"` (default) or `"ffprobe"`.

Details

For FFmpeg the check also lists the filters act needs for burnt-in text that are missing: `ass` (subtitles, needs libass) and `drawtext` (timecode, needs freetype). FFmpeg still counts as working without them (clips, sound and stills without text are possible). On a Mac the plain Homebrew formula `ffmpeg` is built without both; install `ffmpeg-full` instead (see the README of act).

Value

`TRUE` or `FALSE`, with the attributes `path` (the executable that was tested), `reason` (why it does not run; `NA` when it runs) and `filters_missing` (character vector of the missing filters, empty when none is missing or when the executable is ffprobe or does not run) and `hint` (character vector: how to install FFmpeg with these filters on the running operating system; empty when nothing is missing).

See Also

helper_ffmpeg_path, helper_ffprobe_path

Examples

```
ok <- act::helper_ffmpeg_check()
if (!ok) print(attr(ok, "reason"))
```

`helper_ffmpeg_input_flags` *Helper: FFmpeg input flags of an MP4/MOV source*

Description

The demuxer flags that go directly before the `-i` of a source in an MP4/MOV container (mp4, m4a, m4v, mov) whose picture is read: `-ignore_editlist 1 -avoid_negative_ts make_zero`. Other files (images, wav, ...) get none - the image demuxer would abort on them.

Usage

```
helper_ffmpeg_input_flags(filePath)
```

Arguments

`filePath`	Character string; path of the source.

Value

Character vector of FFmpeg arguments (empty for other files).

Examples

```
act::helper_ffmpeg_input_flags("video.mp4")
```

`helper_ffmpeg_keyframe` *Find the keyframe a copied clip can start at*

Description

A video stream can only be copied (`videoCodecCopy` in helper_ffmpeg_args_clip) from a keyframe on. This function returns the time of the last keyframe at or before `startsec`, on the same timeline as `startsec` (the one players and ELAN show, see the option `act.media.timeline`). Only a short stretch before the start is read.

Usage

```
helper_ffmpeg_keyframe(input, startsec, videoOffset = NULL)
```

Arguments

`input`	Character; path of the source video.
`startsec`	Numeric; time in seconds.
`videoOffset`	Numeric or `NULL`; edit list offset of the video track, see helper_ffmpeg_args. `NULL` reads it from the file.

Value

Numeric; the keyframe time in seconds, or `startsec` when the file is not a video or no keyframe was found. The attribute `found` tells the two apart: `TRUE` for a keyframe read from the file, `FALSE` when the lookup failed (not an MP4/MOV file, file missing, ffprobe missing, failed or timed out, no keyframe at or before `startsec`).

See Also

helper_ffmpeg_args_clip

`helper_ffmpeg_path` *Helper: Resolve the path to the FFmpeg executable*

Description

Returns the path to the FFmpeg executable using the following cascade:

1. Option `act.path.ffmpeg`, if it is set to a concrete path (any value other than the default `"ffmpeg"`). This path is returned as it is, without a test.
2. The first executable that runs and has the filters for burnt-in text (`ass`, `drawtext`), checked in this order: `Sys.which("ffmpeg")` (the PATH of the R session), then the known installation locations (`/opt/homebrew/bin`, `/opt/homebrew/opt/ffmpeg-full/bin`, `/usr/local/bin`, `/usr/bin`). So a Homebrew `ffmpeg-full` is found even when it is not linked and the plain formula `ffmpeg` is installed as well.
3. The first of these executables that runs, without these filters.
4. The first of these executables that exists, even if it does not run (helper_ffmpeg_check then reports why).
5. The literal `"ffmpeg"` as last resort (resolution is left to the shell).

Usage

```
helper_ffmpeg_path()
```

Details

Whether an executable runs is tested once per session (`ffmpeg -version`).

Value

Character string; path to the FFmpeg executable or `"ffmpeg"`.

See Also

helper_ffprobe_path, helper_ffmpeg_check

Examples

```
act::helper_ffmpeg_path()
```

`helper_ffmpeg_run` *Helper: Run FFmpeg and check the written file*

Description

Runs FFmpeg with an argument vector (as built by helper_ffmpeg_args) and checks the result at the file: FFmpeg ends with status 0 even when it wrote nothing (e.g. a seek behind the end of the file). A file of the same name is removed first, so an old file never counts as a new result.

Usage

```
helper_ffmpeg_run(args, output, what = "ffmpeg", quiet = FALSE)
```

Arguments

`args`	Character vector; FFmpeg arguments without the program name.
`output`	Character; path of the file the call is expected to write.
`what`	Character; short description used in the failure message.
`quiet`	Logical; if `TRUE` no message is printed on failure.

Value

Logical; `TRUE` if the file was written and is not empty.

See Also

helper_ffmpeg_args, helper_ffmpeg_path

Examples

```
## Not run:
a <- act::helper_ffmpeg_args(output = "still.jpg", input = "video.mp4", startsec = 12.5)
act::helper_ffmpeg_run(a, output = "still.jpg")

## End(Not run)
```

helper_ffprobe_path *Helper: Resolve the path to the ffprobe executable*

Description

Returns the path to the ffprobe executable. ffprobe is installed together with FFmpeg and lives in the same directory. The cascade is:

1. `ffprobe` next to the FFmpeg executable resolved by helper_ffmpeg_path.
2. The first executable that runs: `Sys.which("ffprobe")`, then the known installation locations (`/opt/homebrew/bin`, `/opt/homebrew/opt/ffmpeg-full/bin`, `/usr/local/bin`, `/usr/bin`).
3. The first of these executables that exists, even if it does not run.
4. The literal `"ffprobe"` as last resort (resolution is left to the shell).

Usage

```
helper_ffprobe_path()
```

Value

Character string; path to the ffprobe executable or `"ffprobe"`.

See Also

helper_ffmpeg_path, helper_ffmpeg_check

Examples

```
act::helper_ffprobe_path()
```

helper_file_app_available *Check whether an external application is available*

Description

Check whether an external application is available

Usage

```
helper_file_app_available(app = c("elan", "praat"))
```

Arguments

app	Character string; `"elan"` or `"praat"`.

Value

Logical; TRUE if the option act.path.elan / act.path.praat points to an existing file.

---

helper_file_date_key_make *Helper: Sort key from the date in a file name*

Description

Builds a sortable key from the date (and optional version letter) in file names, e.g. 2026-10-04b. A plain date sorts before the same date with a version letter (no letter counts as version 0). Files without a date get an empty key, which sorts lowest.

Usage

```
helper_file_date_key_make(fileNames, datePattern)
```

Arguments

fileNames	Vector of character strings; file names (or paths).
datePattern	Character string; regular expression matching the date (optionally followed by a version letter) in a file name. Empty or NULL: every key is empty.

Value

Vector of character strings, one key per file name.

See Also

helper_file_newest_select

Examples

```
act::helper_file_date_key_make(c("a__2026-10-04.eaf", "a__2026-10-04b.eaf", "a.eaf"),
                               "__\\d{4}-\\d{2}-\\d{2}[a-z]?")
```

`helper_file_hash_compute`

Fingerprint of files (Git blob hash)

Description

Computes the fingerprint of the content of files as Git does (`git hash-object`): SHA-1 of `"blob <size>\0"` followed by the bytes of the file. Two files with the same fingerprint have the same content, byte for byte (a byte order mark and the line endings count).

Usage

```
helper_file_hash_compute(filePaths)
```

Arguments

`filePaths`	Vector of character strings; paths of the files.

Value

Vector of character strings, one fingerprint per file; `NA` for a file that does not exist or cannot be read.

Examples

```
library(act)
path <- system.file("extdata", "examplecorpus", "sync",
                    "SYNC_rotar_y_flexionar.eaf", package = "act")
helper_file_hash_compute(path)
```

`helper_file_newest_select`

Helper: The newest of several files

Description

Chooses the newest of several files: by the date (and version letter) in the file name (see helper_file_date_key_make); when no file carries a date, by the modification time; when that is unknown too, the first file.

Usage

```
helper_file_newest_select(filePaths, datePattern)
```

Arguments

`filePaths`	Vector of character strings; paths of the files.
`datePattern`	Character string; regular expression matching the date in a file name. `NULL` or empty: modification time only.

Value

Integer; index of the newest file (`integer(0)` for no files).

See Also

helper_file_date_key_make

`helper_file_open_in` *Open a file in ELAN or Praat*

Description

Starts the application with the file. The location of the application comes from the options `act.path.elan` and `act.path.praat` unless `pathApp` is given. ELAN opens `.eaf` files, Praat opens `.TextGrid` files; the format is not checked here.

Usage

```
helper_file_open_in(path, app = c("elan", "praat"), pathApp = NULL)
```

Arguments

`path`	Character string; path to the annotation file.
`app`	Character string; `"elan"` or `"praat"`.
`pathApp`	Character string or `NULL`; path to the application. If `NULL` the corresponding option is used.

Value

Logical; `TRUE` if the command was issued (invisibly). Aborts with a message if the file or the application is not found.

Examples

```
## Not run:
helper_file_open_in("/path/to/file.eaf", app = "elan")
helper_file_open_in("/path/to/file.TextGrid", app = "praat")

## End(Not run)
```

helper_file_reveal *Reveal a file in the file manager of the operating system*

Description

Opens the folder of the file and selects the file: Finder on macOS, Explorer on Windows. On Linux only the folder is opened (there is no common way to select a file).

Usage

```
helper_file_reveal(path)
```

Arguments

path	Character string; path to the file (or folder).

Value

Logical; TRUE if the command was issued, FALSE if the path does not exist (invisibly).

Examples

```
## Not run:
helper_file_reveal("/path/to/file.eaf")

## End(Not run)
```

helper_file_state *State of a file on disk*

Description

The state a transcript keeps of the file it was read from or last written to (slot file.state of a transcript object): path, fingerprint, size, modification time and the time the state was taken. It is compared with the file on disk before the file is overwritten (helper_file_state_changed).

Usage

```
helper_file_state(filePath, hash = NULL)
```

Arguments

filePath	Character string; path of the file.
hash	Character string; the fingerprint when it is known already (computed from bytes that were read anyway), NULL to compute it.

Value

A list with `path`, `hash` (see helper_file_hash_compute), `size` (bytes), `mtime` (POSIXct) and `time` (POSIXct, when the state was taken); an empty list when the file does not exist.

Examples

```
library(act)
path <- system.file("extdata", "examplecorpus", "sync",
                    "SYNC_rotar_y_flexionar.eaf", package = "act")
helper_file_state(path)
```

---

helper_file_state_changed

Has a file changed since its state was taken?

Description

Compares the file on disk with a state taken earlier (helper_file_state). The content decides: when size and modification time are the same, the file counts as unchanged without being read; otherwise its fingerprint is compared. A modification time that changed without a change of the content (e.g. by a sync service) therefore does not count as a change.

Usage

```
helper_file_state_changed(state, filePath = NULL)
```

Arguments

`state`	List; a state from helper_file_state (e.g. the slot `file.state` of a transcript).
`filePath`	Character string; the file to compare. Default is the path of the state; a state of another file counts as no state.

Value

`FALSE` unchanged, `TRUE` changed or no longer there, `NA` no state for this file.

Examples

```
library(act)
path <- system.file("extdata", "examplecorpus", "sync",
                    "SYNC_rotar_y_flexionar.eaf", package = "act")
state <- helper_file_state(path)
helper_file_state_changed(state)
```

helper_history_changed

Changed since the last import or export

Description

Reads the `@history` of a transcript (or of every transcript of a corpus) and reports whether entries were added after the last *baseline* entry. Baseline entries are the import entries written by the import functions (`import_eaf`, `import_textgrid`, ...; every entry whose modification starts with `"import_"`) and the export marks written by `helper_history_mark_export`.

Usage

```
helper_history_changed(x)
```

Arguments

`x`	Transcript object or corpus object.

Value

Named logical vector, one element per transcript (`TRUE` = changed since the last import or export). For a single transcript the vector has length one.

See Also

helper_history_mark_export

helper_history_mark_export

Mark transcripts as exported

Description

Appends an `export` entry to the `@history` of the given transcripts of a corpus. `corpus_export` does not change the corpus it is given; call this helper afterwards so that `helper_history_changed` knows that the current state has been written to disk.

Usage

```
helper_history_mark_export(
  x,
  transcriptNames = NULL,
  formats = NULL,
  folderOutput = NULL
)
```

Arguments

x	Corpus object.
transcriptNames	Vector of character strings; names of the exported transcripts. If left unspecified, all transcripts are marked.
formats	Vector of character strings; the exported formats.
folderOutput	Character string; the output folder.

Value

Corpus object with the updated histories.

See Also

helper_history_changed, corpus_export

---

helper_layout_anchors *Anchor positions of a rendered transcript*

Description

Returns the anchor specification of every rendered row: which symbol of which layer row targets which column and line of its verbal source row. Intended for viewers that draw or edit aligned transcripts.

Usage

```
helper_layout_anchors(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  layerOrder = NULL,
  mainTierNames = NULL,
  alignChars = NULL,
  alignModes = NULL,
  style = NULL
)
```

Arguments

t	Transcript object.
filterTierNames	Vector of character strings; names of the tiers to include.
filterSectionStartsec	Numeric; start of a section in seconds.

filterSectionEndsec
: Numeric; end of a section in seconds.

layerOrder
: Vector of character strings; order of the multimodal layers within a block. NULL keeps the tier order of the annotation file.

mainTierNames
: Vector of character strings; exact names of the tiers to treat as main tiers. NULL derives the main flag from the tier styles of the profile; without any tier style every tier counts as a main tier.

alignChars
: Named vector of character strings; anchor characters per layer tier (names = tier names, values = the characters). NULL derives them from the tier styles of the profile. Catch-all multimodal tiers without an entry anchor on the union of all given characters.

alignModes
: Named vector of character strings; alignment mode per layer tier ("bracket" for spans with open and close, "point" for single spots). NULL derives the mode from the tier styles of the profile; without any source the mode defaults to "bracket".

style
: Transcript profile: the name of a profile, the path of a profile file or a profile read with helper_style_read. NULL uses the profile "act".

Value

Data.frame with one row per anchored symbol (columns include row, tierName, char, occurrence, target_col, target_line, source_row, prefix_width, type, status, pair_id, source_occurrence). target_col counts from the text body of the source row - line number and speaker sigle are already subtracted; their width is reported in prefix_width. status explains an NA target_col: "placed", "flush_leading" (leading symbol flush with its segment start, no counterpart needed), "no_match" (no time-matched main symbol), "target_dropped" (matched symbol not present in the rendered line), "beyond_width" (target outside the wrap width). pair_id joins the two brackets of one overlap pair; source_occurrence is the bracket occurrence in the source row.

---

helper_layout_bracket_pairs

Overlap bracket pairs of a transcript

Description

Computes which opening and closing overlap brackets of the verbal tiers belong together, without rendering. Row indices refer to the engine frame returned in the ann element.

Usage

```
helper_layout_bracket_pairs(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  mainTierNames = NULL,
```

```
  alignChars = NULL,
  alignModes = NULL,
  style = NULL
)
```

Arguments

t	Transcript object.
filterTierNames	Vector of character strings; names of the tiers to include.
filterSectionStartsec	Numeric; start of a section in seconds.
filterSectionEndsec	Numeric; end of a section in seconds.
mainTierNames	Vector of character strings; exact names of the tiers to treat as main tiers. NULL derives the main flag from the tier styles of the profile; without any tier style every tier counts as a main tier.
alignChars	Named vector of character strings; anchor characters per layer tier (names = tier names, values = the characters). NULL derives them from the tier styles of the profile. Catch-all multimodal tiers without an entry anchor on the union of all given characters.
alignModes	Named vector of character strings; alignment mode per layer tier ("bracket" for spans with open and close, "point" for single spots). NULL derives the mode from the tier styles of the profile; without any source the mode defaults to "bracket".
style	Transcript profile: the name of a profile, the path of a profile file or a profile read with helper_style_read. NULL uses the profile "act".

Value

List with the engine frame ann and the data.frame pairs (columns i_row, i_occurrence, j_row, j_occurrence).

---

helper_layout_render *Render a transcript with the alignment engine*

Description

Renders a single transcript into aligned print-transcript lines using the wrap-aware alignment engine. Layout, styles and the advanced engine settings (time tolerances, minimum description room, span limit, picture tiers, catch-all multimodal tiers) come from the transcript profile style; its mode decides between "gat" (one line per annotation) and "mondada" (score layout).

Usage

```
helper_layout_render(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  layerOrder = NULL,
  mainTierNames = NULL,
  alignChars = NULL,
  alignModes = NULL,
  insertArrowStartsec = NA_real_,
  insertArrowEndsec = NA_real_,
  insertArrowTierName = NA_character_,
  style = NULL
)
```

Arguments

`t`	Transcript object.
`filterTierNames`	Vector of character strings; names of the tiers to include.
`filterSectionStartsec`	Numeric; start of a section in seconds.
`filterSectionEndsec`	Numeric; end of a section in seconds.
`layerOrder`	Vector of character strings; order of the multimodal layers within a block. `NULL` keeps the tier order of the annotation file.
`mainTierNames`	Vector of character strings; exact names of the tiers to treat as main tiers. `NULL` derives the main flag from the tier styles of the profile; without any tier style every tier counts as a main tier.
`alignChars`	Named vector of character strings; anchor characters per layer tier (names = tier names, values = the characters). `NULL` derives them from the tier styles of the profile. Catch-all multimodal tiers without an entry anchor on the union of all given characters.
`alignModes`	Named vector of character strings; alignment mode per layer tier (`"bracket"` for spans with open and close, `"point"` for single spots). `NULL` derives the mode from the tier styles of the profile; without any source the mode defaults to `"bracket"`.
`insertArrowStartsec`	Numeric; start time (seconds) of an annotation to mark with the arrow of the profile, e.g. a search hit. All lines get room for the arrow before the line number, so the marked line stays in its column. `NA`: no arrow.
`insertArrowEndsec`	Numeric; end time (seconds) of the annotation to mark.
`insertArrowTierName`	Character string; tier name of the annotation to mark.

`style` Transcript profile: the name of a profile, the path of a profile file or a profile read with `helper_style_read`. `NULL` uses the profile `"act"`.

Value

List with the rendered `lines`, the line `plan`, the engine `result` frame (one row per annotation; its column `line_number` carries the printed line number of a numbered main annotation as a character string and `NA` otherwise - in the `mondada` mode, which numbers printed lines rather than annotations, it is always `NA`), the filtered `transcript`, the `engineWidth`, the `layoutMode`, and - computed from the same single rendering - the `anchors` table (as in `helper_layout_anchors`), the bracket `pairs` and the multimodal symbol `matches`.

`helper_layout_symbol_matches` *Multimodal symbol matches of a transcript*

Description

Matches every multimodal layer symbol to its verbal main symbol by time, without rendering. Row indices refer to the engine frame returned in the `ann` element.

Usage

```
helper_layout_symbol_matches(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  mainTierNames = NULL,
  alignChars = NULL,
  alignModes = NULL,
  style = NULL
)
```

Arguments

`t` Transcript object.

`filterTierNames` Vector of character strings; names of the tiers to include.

`filterSectionStartsec` Numeric; start of a section in seconds.

`filterSectionEndsec` Numeric; end of a section in seconds.

`mainTierNames` Vector of character strings; exact names of the tiers to treat as main tiers. `NULL` derives the main flag from the tier styles of the profile; without any tier style every tier counts as a main tier.

alignChars	Named vector of character strings; anchor characters per layer tier (names = tier names, values = the characters). NULL derives them from the tier styles of the profile. Catch-all multimodal tiers without an entry anchor on the union of all given characters.
alignModes	Named vector of character strings; alignment mode per layer tier ("bracket" for spans with open and close, "point" for single spots). NULL derives the mode from the tier styles of the profile; without any source the mode defaults to "bracket".
style	Transcript profile: the name of a profile, the path of a profile file or a profile read with helper_style_read. NULL uses the profile "act".

Value

List with the engine frame ann and the data.frame matches (columns layer_row, char, layer_occurrence, main_row, main_occurrence).

---

helper_media_audio_track_exists

Helper: Does a media file have a sound track?

Description

Asks ffprobe for the streams of a file (cached per file).

Usage

```
helper_media_audio_track_exists(filePath)
```

Arguments

filePath	Character string; path of the media file.

Value

TRUE or FALSE; NA when the file does not exist or could not be read.

helper_media_type *Helper: Media type of files*

Description

Tells video from sound files by their extension, using the options `act.media.fileformats.video` and `act.media.fileformats.audio`. These two options are the only list of media formats in act and iclo.

Usage

```
helper_media_type(filePaths)
```

Arguments

`filePaths`	Vector of character strings; file paths or names.

Value

Vector of character strings: `"video"`, `"audio"` or `NA`, one per path.

Examples

```
act::helper_media_type(c("rec.mp4", "rec.WAV", "still.png"))
```

helper_media_video_timing_read *Helper: Timing of a video file*

Description

Reads the edit list offset of the video track and the frame duration of a video file (cached per file). Values passed in are used instead of reading the file. With the option `act.media.timeline = "raw"` the offset is 0.

Usage

```
helper_media_video_timing_read(filePath, videoOffset = NULL, videoFps = NULL)
```

Arguments

`filePath`	Character string; path of the video file.
`videoOffset`	Numeric or `NULL`; known edit list offset in seconds.
`videoFps`	Numeric or `NULL`; known frame rate.

Value

List with `offset` (seconds) and `frame_dur` (seconds, `NA` when unknown).

helper_metadata_exif_argv

Build the exiftool arguments that write act. tags into JPG files*

Description

Returns the argument vector (without the program name) for ONE exiftool call that writes the EXIF `UserComment` of several JPG files, each with its own value (`-execute` between the files, `-q` and `-overwrite_original` as common arguments). Used for direct writing (`helper_metadata_exif_write()`) and for the exiftool line in cut list scripts.

Usage

```
helper_metadata_exif_argv(file, sourcePath, startsec, clipID)
```

Arguments

`file`	Character vector. Path(s) to the JPG file(s).
`sourcePath`	Character vector. Source media file of each still (recycled).
`startsec`	Numeric vector. Time in the source media in seconds (recycled).
`clipID`	Character vector. Cut identifier (recycled).

Value

Character vector of exiftool arguments.

See Also

`helper_metadata_exif_write()`

helper_metadata_exif_write

Write act. tags into the EXIF UserComment of a JPG file*

Description

Uses the bundled exiftool from the `exifr` package (which itself requires Perl on the system). When `exifr` is not installed or `perl` is not on the system path, the function silently returns without modifying the file. The single setup hint is emitted by `iclo::config()` (not by this helper, to keep it usable as a quiet library primitive).

Usage

```
helper_metadata_exif_write(file, sourcePath, startsec, clipID)
```

Arguments

file	Character vector. Path(s) to the JPG file(s) to tag.
sourcePath	Character vector. Full path to the source media file each still was extracted from (recycled).
startsec	Numeric vector. Time in original source media, in seconds, where each still frame was captured (recycled).
clipID	Character vector. Cut identifier (recycled).

Details

Uses `-overwrite_original` to avoid `.jpg_original` backup files. Several files are tagged in ONE exiftool call (see `helper_metadata_exif_argv()`).

Value

Invisibly `NULL`.

See Also

`media_metadata_read()`, `helper_metadata_ffmpeg_args()`, `helper_metadata_exif_argv()`

---

helper_metadata_ffmpeg_args

Build ffmpeg metadata arguments for cut files

Description

Returns a string that can be appended to ffmpeg command lines to write `act.*` tags into the comment field of the output file. For video files (when `fps` is supplied) a native timecode track is added via the `-timecode` option.

Usage

```
helper_metadata_ffmpeg_args(sourcePath, startsec, clipID, fps = NULL)
```

Arguments

sourcePath	Character string. Full path to the source media file.
startsec	Numeric. Start time in original source media, in seconds.
clipID	Character string. Unique cut identifier (e.g. transcript name for cuts from a corpus, output filename stem otherwise).
fps	Numeric or `NULL`. Frame rate of the source video. When supplied a `-timecode HH:MM:SS:FF` argument is added. `NULL` (default) omits the timecode track (suitable for audio-only outputs).

Details

Returns an empty string when option `act.ffmpeg.write_metadata` is `FALSE`, allowing callers to disable tagging globally without changing call sites.

Tag format: `act.source=<path>;act.startsec=<sec_3nks>;act.clipID=<id>`.

Value

Character string with `-metadata comment="..."` and optionally `-timecode "HH:MM:SS:FF"`. Returns `""` if metadata writing is disabled.

See Also

`media_metadata_read()`, `helper_metadata_exif_write()`

---

helper_metadata_ffmpeg_argv

Build ffmpeg metadata arguments for cut files as an argument vector

---

Description

Same tags as `helper_metadata_ffmpeg_args()`, but returned UNQUOTED as a character vector, ready for `system2()` / an ffmpeg argument list. Callers that assemble commands as vectors must use this variant: the comment carries the source path, which may contain spaces, so splitting the quoted string from `helper_metadata_ffmpeg_args()` again is not safe.

Usage

```
helper_metadata_ffmpeg_argv(sourcePath, startsec, clipID, fps = NULL)
```

Arguments

`sourcePath`	Character string. Full path to the source media file.
`startsec`	Numeric. Start time in original source media, in seconds.
`clipID`	Character string. Unique cut identifier (e.g. transcript name for cuts from a corpus, output filename stem otherwise).
`fps`	Numeric or `NULL`. Frame rate of the source video. When supplied a `-timecode HH:MM:SS:FF` argument is added. `NULL` (default) omits the timecode track (suitable for audio-only outputs).

Details

Returns `character(0)` when option `act.ffmpeg.write_metadata` is `FALSE`.

Value

Character vector of ffmpeg arguments, or `character(0)` if metadata writing is disabled.

See Also

helper_metadata_ffmpeg_args(), media_metadata_read()

---

helper_metadata_probe_fps
Probe the frame rate of a media file

Description

One ffprobe stream query instead of a full json read. Callers that write metadata need the frame rate for the timecode track, see helper_metadata_ffmpeg_argv().

Usage

```
helper_metadata_probe_fps(file_path)
```

Arguments

file_path	Character string. Path to the media file.

Value

Numeric frame rate, or NA_real_ when ffprobe fails or the file holds no video stream.

See Also

helper_metadata_ffmpeg_argv(), helper_metadata_ffmpeg_args()

---

helper_program_paths_detect
Helper: Find Praat and ELAN at their usual places

Description

Fills the options act.path.praat and act.path.elan with the programs found at the usual installation places: on macOS /Applications and ~/Applications, on Windows the program folders, elsewhere /opt, the home folder and the search path. When several ELAN versions are installed side by side (ELAN_6.7, ELAN_7.1), the newest one is taken.

Usage

```
helper_program_paths_detect()
```

Details

An option that already points to an existing program is kept. An option that is empty or points nowhere (for example to an ELAN version removed after an update) is replaced by the program found. act calls the helper when it is loaded.

Value

Invisibly, a data frame with one row per option: `option`, `before` (value before the call), `after` (value after the call) and `changed` (logical).

Examples

```
act::helper_program_paths_detect()
```

---

helper_progress_callback_get

Helper: Get the progress callback

Description

Helper: Get the progress callback

Usage

```
helper_progress_callback_get()
```

Value

The function set with `helper_progress_callback_set`, or `NULL`.

---

helper_progress_callback_set

Helper: Set a progress callback

Description

Registers a function that every progress bar of act (and of packages that use `helper_progress_set` / `helper_progress_tick`) calls when it starts and at every step, also while the console bars are switched off (option `act.showprogress`). A graphical caller (a Shiny app) uses it to show the progress itself.

Usage

```
helper_progress_callback_set(fn)
```

Arguments

fn	Function with the arguments title (character), current (number of finished items) and total (number of all items), or NULL to remove the callback.

Value

The previous callback (or NULL), invisibly - pass it back to restore it.

Examples

```
old <- helper_progress_callback_set(function(title, current, total)
 message(title, ": ", current, " of ", total))
helper_progress_set("Counting", 3)
for (i in 1:3) helper_progress_tick()
helper_progress_callback_set(old)
```

---

helper_progress_eta_format

Helper: Remaining time of a progress bar

Description

Formats the remaining time of a loop as a fixed-width tail " (%7s left)" (15 characters, fits "59m 20s"), estimated from the elapsed time per finished item. The progress bars of act and iclo use it.

Usage

```
helper_progress_eta_format(start, current, total)
```

Arguments

start	POSIXct; start time of the loop.
current	Integer; number of finished items.
total	Integer; number of all items.

Value

Character string.

helper_progress_set *Helper: Set progress bar*

Description

Helper: Set progress bar

Usage

```
helper_progress_set(title, total)
```

Arguments

title	Character string; Title of progress bar.
total	Integer; Number of items to tick.

Examples

```
library(act)

# --- Create two tier tables from scratch
tierTable1 <- act::helper_tiers_new_table(c("a","b","c","d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable2 <- act::helper_tiers_new_table(c("a","b","x","y"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable3 <- act::helper_tiers_merge_tables(tierTable1,tierTable2)
tierTable3
```

helper_progress_tick *Helper: Advance progress bar by one tick*

Description

Helper: Advance progress bar by one tick

Usage

```
helper_progress_tick(len = 1)
```

Arguments

len	Integer; number of items finished with this tick (default 1).

Examples

```
library(act)

# --- Create two tier tables from scratch
tierTable1 <- act::helper_tiers_new_table(c("a","b","c","d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable2 <- act::helper_tiers_new_table(c("a","b","x","y"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable3 <- act::helper_tiers_merge_tables(tierTable1,tierTable2)
tierTable3
```

`helper_style_acronym` *Helper: Acronyms of tiers by a style profile*

Description

Builds the acronym of each tier name the way the print transcript does: case, search and replace, extract, width, ending - with the values of the tier style, and where it leaves one open, those of the profile.

Usage

```
helper_style_acronym(style, tierNames, ending = TRUE)
```

Arguments

`style`	Style profile; see `helper_style_read`.
`tierNames`	Vector of character strings; tier names.
`ending`	Logical; if `TRUE` the ending of the acronym is added.

Value

Vector of character strings; one acronym per tier name.

helper_style_appearance *Helper: Styles of a style profile with their appearance*

Description

Lists the styles of a profile with the appearance that finally holds for each: its own values, and for a value it does not set the one of the style it builds on (transcript lines on the default style, the default and the header styles on Normal). Character styles only have their own values.

Usage

```
helper_style_appearance(style)
```

Arguments

style	Style profile; see helper_style_read.

Value

Data.frame with one row per style: name, type, role, active, word, pattern, applies, and font, size, color, background, italic, bold (NA where nothing is set).

helper_style_docx *Helper: Word file of a style profile*

Description

Writes an empty Word file that carries the styles of a style profile: the file the profile names (or templatePath, or the template of the package) with the paragraph and character styles of the profile written into it. Use it as a template of your own, or to merge transcripts into.

Usage

```
helper_style_docx(
  style,
  pathOutput,
  templatePath = NULL,
  symbolChars = character(0)
)
```

Arguments

style	Style profile; see helper_style_read.
pathOutput	Character string; path of the .docx file to write.
templatePath	Character string; path of the Word file to start from. NULL takes the file named in the profile.
symbolChars	Vector of character strings; the multimodal symbols, needed only to return the character rules.

Value

Invisibly a list with the path and the names of the styles that were created (not in the Word file before).

---

helper_style_list *Helper: Style profiles that can be found*

Description

Helper: Style profiles that can be found

Usage

```
helper_style_list()
```

Value

Named vector of character strings; paths of the profile files, named by the profile names.

---

helper_style_read *Helper: Read a style profile*

Description

A style profile is a JSON file that holds the whole format of a print transcript: line structure, acronyms, tier styles, character styles and the appearance in Word. A profile can build on another profile (extends) and then only states what differs.

Usage

```
helper_style_read(style)
```

Arguments

style	Character string or list; the name of a profile (a file <name>.json in one of the profile folders), the path of a profile file, or a profile given as list.

Details

The profile folders are the folders named in the option `act.style.folders` and the folder `extdata/styles` of the package. The profile is checked; all problems found are reported together.

Value

The profile as list of class `act_style`, with all values filled in.

`helper_style_video` *Helper: Look of the subtitle rows by a style profile*

Description

The subtitles of a video show the annotations of tiers as rows; the profile holds their look in the block `video` of the default style (text and box) and in the block `video` of each tier style (the values it changes for its tiers). This helper resolves the look the way the print transcript resolves the appearance: the tier style of a tier, and for every value it leaves open the default.

Usage

```
helper_style_video(style, tierNames)
```

Arguments

`style`	Style profile; see `helper_style_read`.
`tierNames`	Vector of character strings; tier names.

Value

List with `text` (font, size in percent of the video height, color, effect, effect.color, effect.width), `box` (background, opacity, padding in font sizes, overflow, lines) and `rows`: one entry per tier name with `style` (name of the tier style), `row` (a subtitle row by default), `size` (percent of the text size), `color`, `effect`, `effect.color`, `effect.width`, `lines`, `bold`, `italic` and `acronym` (the row carries the acronym of the speaker).

helper_text_graphemes_split

Helper: Split a text into graphemes

Description

Splits a character string into its user-perceived characters (grapheme clusters): a base letter with combining accents counts as one element. The layout engine measures and aligns text with this split.

Usage

```
helper_text_graphemes_split(text)
```

Arguments

text	Character string.

Value

Character vector, one element per grapheme; empty for NA or an empty string.

Examples

```
act::helper_text_graphemes_split("ca\u0301fe")
```

helper_tiers_filter_create

Helper: Create filter for tier names

Description

Creates a tier name filter based on a vector of character strings and the values 'filterTierIncludeRegEx' and 'filterTierExcludeRegEx' (in a transcript profile: functions$tiers.keep and functions$tiers.exclude). First, tiers matching 'filterTierExcludeRegEx' are removed. Then, tiers matching 'filterTierIncludeRegEx' are added back from the original tier list, ensuring they are always included regardless of the exclude filter. The original tier order is preserved.

Usage

```
helper_tiers_filter_create(
  tierNames,
  filterTierIncludeRegEx = "",
  filterTierExcludeRegEx = ""
)
```

Arguments

tierNames	Vector of character strings; names of the tiers.
filterTierIncludeRegEx	Character string; as regular expression, tiers matching the expression will always be included, even if they match the exclude expression.
filterTierExcludeRegEx	Character string; as regular expression, tiers matching the expression will be excluded from the print transcript.

Value

Vector of character strings; names of tiers

Examples

```
library(act)

# --- Create a tier table from scratch
tierTable <- act::helper_tiers_new_table(c("a","b","c", "d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))
tierTable
```

---

helper_tiers_merge_tables

Helper: Merge tier tables

Description

Merges several the tier tables into one tier table.

Usage

```
helper_tiers_merge_tables(...)
```

Arguments

...	accepts different kinds of objects; transcript objects, lists of transcript objects (as in @transcripts of a corpus object) and tier tables (as in @tiers of a transcript object).

Details

NOTE: To actually modify the tiers in a transcript object or a corpus object corpus use the functions of the package, e.g. `act::transcripts_merge`. This function is only a helper function and for people that like experiments. If tiers with the same name are of different types ('IntervalTier', 'TextTier') an error will be raised. In that case can use, for example, 'act::tier_convert()' to change the tier types.

Value

Data.frame

See Also

helper_tiers_sort_table, helper_tiers_merge_tables, tiers_convert, tiers_rename, tiers_sort, transcripts_merge

Examples

```
library(act)

# --- Create two tier tables from scratch
tierTable1 <- act::helper_tiers_new_table(c("a","b","c","d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable2 <- act::helper_tiers_new_table(c("a","b","x","y"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

tierTable3 <- act::helper_tiers_merge_tables(tierTable1,tierTable2)
tierTable3
```

---

helper_tiers_new_table

Helper: Create a tier table

Description

Creates a new tier table as necessary in `@tiers` of a transcript object.

Usage

```
helper_tiers_new_table(tierNames, tierTypes = NULL, tierPositions = NULL)
```

Arguments

`tierNames`	Vector of character strings; names of the tiers.
`tierTypes`	Vector of character strings; types of the tiers. Allowed values: "IntervalTier","TextTier". Needs to have the same length as 'tierNames'.
`tierPositions`	Vector of integer values; Sort order of the tiers. Needs to have the same length as 'tierNames'.

Details

NOTE: To actually modify the tiers in a transcript object or a corpus object corpus use the functions of the package. This function is only a helper function and for people that like experiments.

Value

Data.frame

See Also

helper_tiers_sort_table, helper_tiers_merge_tables, tiers_convert, tiers_rename, tiers_sort

Examples

```
library(act)

# --- Create a tier table from scratch
tierTable <- act::helper_tiers_new_table(c("a","b","c", "d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))
tierTable
```

---

helper_tiers_sort_table

Helper: Sort a tier table

Description

NOTE: To actually reorder the tiers in a transcript object or a corpus object corpus use `act::tiers_sort`. This function is only a helper function and for people that like experiments.

Usage

```
helper_tiers_sort_table(
  tierTable,
  sortVector,
  tiersAddMissing = TRUE,
  tiersDelete = FALSE
)
```

Arguments

`tierTable`	Data frame; tiers as specified and necessary in `@tiers` of a transcript object.
`sortVector`	Vector of character strings; exact tier names (no regular expressions). The order within the vector presents the new order of the tiers. Use "*" to indicate where tiers that are not present in the sort vector but in the transcript should be inserted (the escaped form, a backslash followed by a star, is accepted as well).
`tiersAddMissing`	Logical; if `TRUE` all tiers that are given in 'the 'sortVector' but are missing in 'tierTable' will be added.
`tiersDelete`	Logical; if `TRUE` tiers that are not named in 'sortVector' will be deleted. Otherwise they will be inserted at the end of the table or at the position defined by "*" in 'sortVector'.

Details

Sort a tier table by a predefined vector of tier names. The names are compared exactly (no regular expressions). Tiers that are missing in the table but are present in the sort vector may be inserted. Tiers that are present in the table but not in the sort vector may be deleted or inserted. These tiers will be inserted by default at the end of the table. You may also use an element "*" in 'sortVector' to define the position where they should be placed (the escaped form, a backslash followed by a star, is accepted as well).

Value

Data.frame

See Also

tiers_sort, helper_tiers_new_table, helper_tiers_merge_tables

Examples

```
# This function applies to the tier tables that are necessary in \code{@tiers} of a transcript.
# object. For clarity, we will create such a table from scratch.

library(act)

# --- Create a tier table from scratch
tierTable <- helper_tiers_new_table(c("a","b","c", "d"),
c("IntervalTier", "TextTier","IntervalTier","TextTier"))

# --- Create a vector, defining the new order of the tiers.
sortVector <- c("c","a","d","b")

# Sort the table
tierTable.1 <- act::helper_tiers_sort_table(tierTable=tierTable, sortVector=sortVector)
tierTable.1

# --- Create a vector, in which the tier "c" is missing.
sortVector <- c("a","b","d")

# Sort the table, the missing tier will be inserted at the end.
tierTable.1 <- act::helper_tiers_sort_table(tierTable=tierTable, sortVector=sortVector)
tierTable.1

# --- Create a vector, in which the tier "c" is missing,
# but define the place, where missing tiers will be inserted by "*"
sortVector <- c("a","\\*", "b","d")

# Sort the table. The missing tier "c" will be inserted in second place.
tierTable.2 <- act::helper_tiers_sort_table(tierTable=tierTable, sortVector=sortVector)
tierTable.2

# Sort the table, but delete tiers that are missing in the sort vector
# Note: If 'tiersDelete=TRUE' tiers that are missing in the
```

```
# will be deleted, even if the 'sortVector' contains a "\*".
tierTable.3 <- act::helper_tiers_sort_table(tierTable=tierTable,
sortVector=sortVector,
tiersDelete=TRUE)
tierTable.3

# --- Create a vector, which contains tier names that are not present in 'tierTable'.
sortVector <- c("c","a","x", "y", "d","b")
tierTable.4 <- act::helper_tiers_sort_table(tierTable=tierTable, sortVector=sortVector)
tierTable.4
```

`helper_time_format` *Formats time as HH:MM:SS,mmm*

Description

Formats time as HH:MM:SS,mmm

Usage

```
helper_time_format(
  t,
  digits = 1,
  addHrsMinSec = FALSE,
  addSec = FALSE,
  format = NULL,
  fps = 25
)
```

Arguments

`t`	Double; time in seconds.
`digits`	Integer; number of digits.
`addHrsMinSec`	Logical; if `TRUE` 'hrs' 'min' 'sec' will be used instead of ':'.
`addSec`	Logical; if `TRUE` time value in seconds will be shown, too.
`format`	Character or `NULL`; a named time format. If set, `digits`, `addHrsMinSec` and `addSec` are ignored. One of `"h:mm:ss.s"`, `"h:mm:ss.ss"`, `"h:mm:ss:ff"` (frames), `"mm:ss.ss"` (hours are dropped), `"s.s"`, `"s.ss"`. Fractions are truncated, not rounded.
`fps`	Numeric; frames per second for `"h:mm:ss:ff"`.

Value

Character string.

Examples

```
library(act)


helper_time_format(12734.2322345)
helper_time_format(2734.2322345)
helper_time_format(34.2322345)
helper_time_format(0.2322345)

helper_time_format(12734.2322345, addHrsMinSec=TRUE)
helper_time_format(2734.2322345, addHrsMinSec=TRUE)
helper_time_format(34.2322345, addHrsMinSec=TRUE)
helper_time_format(0.2322345, addHrsMinSec=TRUE)

helper_time_format(12734.2322345, digits=3)
helper_time_format(2734.2322345, digits=3)
helper_time_format(34.2322345, digits=3)
helper_time_format(0.2322345, digits=3)

helper_time_format(12734.2322345, addHrsMinSec=TRUE, digits=3)
helper_time_format(2734.2322345, addHrsMinSec=TRUE, digits=3)
helper_time_format(34.2322345, addHrsMinSec=TRUE, digits=3)
helper_time_format(0.2322345, addHrsMinSec=TRUE, digits=3)

helper_time_format(12734.2322345, addHrsMinSec=TRUE, addSec=TRUE)
helper_time_format(2734.2322345, addHrsMinSec=TRUE, addSec=TRUE)
helper_time_format(34.2322345, addHrsMinSec=TRUE, addSec=TRUE)
helper_time_format(0.2322345, addHrsMinSec=TRUE, addSec=TRUE)

helper_time_format(12734.2322345, addHrsMinSec=TRUE, digits=3, addSec=TRUE)
helper_time_format(2734.2322345, addHrsMinSec=TRUE, digits=3, addSec=TRUE)
helper_time_format(34.2322345, addHrsMinSec=TRUE, digits=3, addSec=TRUE)
helper_time_format(0.2322345, addHrsMinSec=TRUE, digits=3, addSec=TRUE)

helper_time_format(83.45, format="h:mm:ss.ss")
helper_time_format(83.45, format="h:mm:ss:ff", fps=25)
```

---

helper_time_formats_list

Helper: Time formats

Description

Lists the time formats that helper_time_format accepts as `format`.

Usage

```
helper_time_formats_list()
```

Value

Character vector of format names.

Examples

```
act::helper_time_formats_list()
```

`helper_transcript_names_get` *Helper: Get names of all transcripts in a corpus*

Description

Gets the names of all transcript objects in a corpus object based from the `@name` attribute of each transcript.

Usage

```
helper_transcript_names_get(x)
```

Arguments

`x`	Corpus object

Value

List

Examples

```
library(act)

act::helper_transcript_names_get(examplecorpus)
```

helper_transcript_names_make *Helper: Makes valid names for all transcripts in a corpus*

Description

Makes valid names for all transcript objects in a corpus object based on the names passed in 'transcriptNames' parameter. In particular, the functions also corrects names, which have to be non-empty and unique. The following options are performed in the mentioned order.

Usage

```
helper_transcript_names_make(
  transcriptNames,
  extractPatterns = NULL,
  searchPatterns = NULL,
  searchReplacements = NULL,
  toUpper = FALSE,
  toLower = FALSE,
  trim = FALSE,
  defaultEmpty = "no_name"
)
```

Arguments

transcriptNames	Vector of character strings; Names of the transcripts to validate.
extractPatterns	Vector of character strings; Extract pattern as regular expression. Leave empty for no search-replace in the names.
searchPatterns	Vector of character strings; Search pattern as regular expression. Leave empty for no search-replace in the names.
searchReplacements	Vector of character strings; Replacements for search. Leave empty for no search-replace in the names.
toUpper	Logical; Convert transcript names all to upper case.
toLower	Logical; Convert transcript names all to lower case.
trim	Logical; Remove leading and trailing spaces in names.
defaultEmpty	Character string; Default value for empty transcript names (e.g., resulting from search-replace operations)

Value

List

Examples

```
library(act)

# make some names with an empty value "" and a duplivate "d"
transcriptNames <- c("a", "b", "", "d", "d")
act::helper_transcript_names_make(transcriptNames)
```

`helper_transcript_names_set`
Helper: Set names of all transcripts in a corpus

Description

Sets the names of all transcript objects in a corpus object both in the names of the list `x@transcripts` and in the slot `@name` of each transcript.

Usage

```
helper_transcript_names_set(x, transcriptNames)
```

Arguments

`x`	Corpus object
`transcriptNames`	Vector of character strings; new names.

Value

List

Examples

```
library(act)

# get current names of the transcripts
names.old <- act::helper_transcript_names_get(examplecorpus)

# rename giving numbers as names
names.test <- as.character(seq(1:length(names.old)))
test <-  act::helper_transcript_names_set(examplecorpus, names.test)
names(test@transcripts)

# create an error: empty name
## Not run:
names.test <- names.old
names.test[2] <- " "
test <-  act::helper_transcript_names_set(examplecorpus, names.test)
```

```
## End(Not run)

# create an error: double names
## Not run:
names.test <- names.old
names.test[2] <- names.test[1]
test <-  act::helper_transcript_names_set(examplecorpus, names.test)

## End(Not run)
```

`helper_transcript_patch_apply` *Helper: Apply a patch to a transcript*

Description

Applies a patch created by helper_transcript_patch_make to a transcript object. `"backward"` turns the version after the modification into the version before it (undo), `"forward"` does the opposite (redo).

Usage

```
helper_transcript_patch_apply(x, patch, direction = c("backward", "forward"))
```

Arguments

`x`	Transcript object.
`patch`	List; a patch created by helper_transcript_patch_make.
`direction`	Character string; `"backward"` (undo) or `"forward"` (redo).

Details

Before anything is changed, the function checks that the transcript is in the state the patch expects (the rows to be removed exist, the tier table matches). If not, it stops with an error and the transcript is left untouched. Changes of the media table are applied as added and removed files, so other changes of the media list (e.g. repaired paths) are kept.

Value

Transcript object

See Also

helper_transcript_patch_make

Examples

```
library(act)

before <- examplecorpus@transcripts[[1]]
after  <- before
after@annotations$content[1] <- "changed text"
after@annotations <- after@annotations[-2, ]

patch <- act::helper_transcript_patch_make(before = before, after = after, tierNames = NULL)
patch$annotations$removed
patch$annotations$added

undone <- act::helper_transcript_patch_apply(x = after, patch = patch, direction = "backward")
identical(undone@annotations$content, before@annotations$content)

redone <- act::helper_transcript_patch_apply(x = undone, patch = patch, direction = "forward")
identical(redone@annotations$content, after@annotations$content)
```

---

helper_transcript_patch_make

Helper: Create a patch between two versions of a transcript

Description

Compares two versions of the same transcript object and records what changed between them: the annotation rows that were removed and added, the tier table, the media table and the length. The result can be applied in both directions with helper_transcript_patch_apply, which makes it suitable for undo and redo of arbitrary modifications.

Usage

```
helper_transcript_patch_make(before, after, tierNames = NULL)
```

Arguments

`before`	Transcript object; the version before the modification.
`after`	Transcript object; the version after the modification.
`tierNames`	Vector of character strings; names of the tiers whose annotations are compared. If left unspecified, all annotations are compared.

Details

Annotation rows are identified by their content: tier, start, end and text, with times rounded to milliseconds. Other columns (identifiers, derived columns) travel with the rows but do not decide whether a row changed, so a patch still fits a transcript that was saved and read in again. Tiers are compared by name and type, media by path. If `tierNames` is given, only the annotations of these tiers are compared; the annotations of all other tiers are expected to be unchanged. Derived slots (the `fulltext.*` slots and `@history`) are not part of the patch.

Value

List with the elements `annotations`, `tiers`, `media`, `length.sec` (each `NULL` if unchanged) and `empty` (`TRUE` if nothing changed).

See Also

helper_transcript_patch_apply, helper_transcripts_compare

Examples

```
library(act)

before <- examplecorpus@transcripts[[1]]
after  <- before
after@annotations$content[1] <- "changed text"
after@annotations <- after@annotations[-2, ]

patch <- act::helper_transcript_patch_make(before = before, after = after, tierNames = NULL)
patch$annotations$removed
patch$annotations$added

undone <- act::helper_transcript_patch_apply(x = after, patch = patch, direction = "backward")
identical(undone@annotations$content, before@annotations$content)

redone <- act::helper_transcript_patch_apply(x = undone, patch = patch, direction = "forward")
identical(redone@annotations$content, after@annotations$content)
```

---

helper_transcripts_compare

Helper: Compare two versions of a transcript

Description

Compares the content of two versions of a transcript and reports where they differ: the time regions per tier in which annotations were added, removed or changed, the tiers that were added, removed, renamed or moved, and the media files that were added or removed.

Usage

```
helper_transcripts_compare(x, y, gap = 0, digits = 3, content = FALSE)
```

Arguments

`x`	Transcript object; the older version.
`y`	Transcript object; the newer version.
`gap`	Double; regions of the same tier that are less than or exactly `gap` seconds apart are joined into one region. `0` joins only touching and overlapping regions.

digits	Integer; number of decimals to which times are rounded.
content	Logical; if TRUE, regions gets the columns content.old and content.new: the text of the removed and of the added annotations in the region (several joined by " | ").

Details

The comparison works on the content, not on the file text: for every tier the annotations are compared by start, end and content. Formatting differences of the files (identifiers, dates, number formats) therefore do not count as changes. Times are rounded to digits decimals before they are compared.

A tier that is missing in y while a new tier in y holds exactly the same annotations is reported as renamed. Added, removed and renamed tiers are reported in tiers only; regions covers the tiers that exist in both versions.

Value

List with three data frames: regions (columns tierName, startsec, endsec, change with the values "added", "removed", "changed"; with content also content.old, content.new), tiers (columns change with the values "added", "removed", "renamed", "moved", tierName, tierName.new) and media (columns change, path).

See Also

helper_transcript_patch_make

Examples

```
library(act)

x <- examplecorpus@transcripts[[1]]
y <- x
y@annotations$content[1] <- "changed text"
y@annotations$content[5] <- "also changed"

result <- act::helper_transcripts_compare(x = x, y = y, gap = 1, digits = 3)
result$regions
result$tiers
result$media
```

---

import *Import a single annotation file*

Description

Advice: In most situations it is more convenient to use act::corpus_new, act::corpus_import for importing annotation files.

Usage

```
import(..., transcriptName = NULL)
```

Arguments

...	file path, contents of an annotation file or rPraat object; see description above.
transcriptName	Character string; name of the transcript, if this parameter is set, the default name of the transcript will be changed.

Details

Imports the contents of an annotation file and returns a transcript object.

The input to this function in the parameter '...' may either be (1) the path to an annotation file (Currently 'ELAN' .eaf, 'EXMARaLDA .exb, 'Praat' .TextGrid and 'Subrib title' .srt files), (2) the contents of an annotation file obtained from the `@file.content` or by reading the contents of the files directly with `read.lines()` or (3) a `rPraat` TextGrid object. The slot `@file.content` is only filled when the option `act.import.storefileContentInTranscript` is `TRUE` (default `FALSE`).

Only the first input to '...' will be processed

Value

Transcript object.

See Also

corpus_import, corpus_new, import_eaf, import_exb, import_rpraat, import_srt, import_textgrid

Examples

```
library(act)

# To import an annotation file of your choice:
## Not run:
 path <- "PATH_TO_AN_EXISTING_FILE_ON_YOUR_COMPUTER"

## End(Not run)

# Path to a .TextGrid file that you want to read
filePath <- system.file("extdata", "examplecorpus", "GAT",
      "ARG_I_PAR_Beto.TextGrid", package="act")
t <- act::import(filePath=filePath)
t

# Path to an .eaf file that you want to read
filePath <- system.file("extdata", "examplecorpus", "SYNC",
      "SYNC_rotar_y_flexionar.eaf", package="act")
t <- act::import(filePath=filePath)
t

# Content of a .TextGrid file, e.g. as stored in \code{@file.content}
```

```
# of a transcript object.
fileContent <- examplecorpus@transcripts[['ARG_I_CHI_Santi']]@file.content
t <- act::import(fileContent=fileContent)
t

# Content of an .eaf file, e.g. as stored in \code{@file.content}
# of a transcript object.
fileContent <- examplecorpus@transcripts[['SYNC_rotar_y_flexionar']]@file.content
t <- act::import(fileContent=fileContent)
t
```

`import_eaf` *Import a single 'ELAN' '*.eaf' file*

Description

Advice: In most situations it is more convenient to use `act::corpus_new`, `act::corpus_import` for importing annotation files.

Imports the contents of a 'ELAN' .eaf file and returns a transcript object. The input to this function is either the path to an .eaf file or the contents of a .eaf file obtained from the `@file.content` of an existing transcript object by `readLines()`. The slot `@file.content` is only filled when the option `act.import.storefileContentInTranscript` is `TRUE` (default `FALSE`). If you pass 'fileContent' you need to pass 'transcriptName' as parameter, too.

Usage

```
import_eaf(filePath = NULL, fileContent = NULL, transcriptName = NULL)
```

Arguments

`filePath`	Character string; input path of a single 'ELAN' .eaf file.
`fileContent`	Vector of character strings; contents of an 'ELAN' .eaf file read by `readLines()`.
`transcriptName`	Character string; name of the transcript.

Details

Please note:

- 'ELAN' offers a variety of tier types, some including dependencies from other tiers. Therefore not all annotations do actually have a time value. Missing values will be detected in the superordinate tier or will be interpolated. You will not be able to recognize interpolated values in the annotations.
- Please also note that dependencies between tiers in you .eaf file are not reflected in the transcript object within the act package.

Value

Transcript object.

See Also

corpus_import, corpus_new, import, import_exb, import_rpraat, import_textgrid

Examples

```
library(act)

# Path to an .eaf file that you want to read
path <- system.file("extdata", "examplecorpus", "SYNC",
     "SYNC_rotar_y_flexionar.eaf", package="act")


# To import a .eaf file of your choice:
## Not run:
path <- "PATH_TO_AN_EXISTING_EAF_ON_YOUR_COMPUTER"

## End(Not run)


t <- act::import_eaf(filePath=path)
t

# Content of an .eaf file (already read by \code{readLines}),
# e.g. from an existing transcript object:
mycontent <- examplecorpus@transcripts[['SYNC_rotar_y_flexionar']]@file.content
t <- act::import_eaf(fileContent=mycontent, transcriptName="test")
t
```

---

import_exb *Import a single 'EXMARaLDA' .exb file*

Description

Advice: In most situations it is more convenient to use `act::corpus_new`, `act::corpus_import` for importing annotation files.

Usage

```
import_exb(filePath = NULL, fileContent = NULL, transcriptName = NULL)
```

Arguments

`filePath`	Character string; input path of a single 'EXMARaLDA' .exb file.
`fileContent`	Vector of character strings; contents of a 'EXMARaLDA' .exb file .
`transcriptName`	Character string; name of the transcript.

Details

Imports the contents of a 'EXMARaLDA' .exb file and returns a transcript object. The source is either the path to a .exb file or the contents of a .exb file obtained from the `@file.content` of an existing transcript object. The slot `@file.content` is only filled when the option `act.import.storefileContentInTranscript` is `TRUE` (default `FALSE`). If you pass 'fileContent' you need to pass 'transcriptName' as parameter, too.

Please note:

- 'EXMARaLDA' allows for empty time slots without a time values. Missing values will be interpolated during the import. You will not be able to recognize interpolated values in the data.
- Meta data for tiers (such as the display name etc.) will not be imported.
- Media files are referenced not by their path but only as file names in .exb files. The names will be imported but will not work as paths in act.

Value

Transcript object.

See Also

corpus_import, corpus_new, import, import_eaf, import_rpraat, import_textgrid

Examples

```
library(act)

## Not run:
 # To import a .TextGrid file of your choice:
 filePath <- "PATH_TO_AN_EXISTING_TEXTGRID_ON_YOUR_COMPUTER"

 t <- act::import_exb(filePath=filePath)
 t

## End(Not run)
```

---

import_rpraat *Import a single 'rPraat' TextGrid object*

Description

This function is to create compatibility with the `rPraat` package. It converts a 'rPraat' TextGrid object into an act transcript object.

Usage

```
import_rpraat(rpraatTextgrid, transcriptName = NULL)
```

Arguments

rpraatTextgrid	List; rPraat TextGrid object.
transcriptName	Character string; name of the transcript.

Details

Please note:

- Time values of annotations in TextGrids may be below 0 seconds. Negative time values will be recognized corretly in the first place. When exporting transcript object to other formats like 'ELAN' .eaf, 'EXMARaLDA' .exb ect. annotations that are completely before 0 sec will be deleted, annotations that start before but end after 0 sec will be truncated. Please see also the function `act::transcripts_cure_single`.
- TextGrids and contained tiers may start and end at different times. These times do not need to match each other. The act package does not support start and end times of TextGrids and tiers and will. The default start of a TextGrid will be 0 seconds or the lowest value in case that annotations start below 0 seconds.

Credits: Thanks to Tomáš Bořil, the author of the rPraat package, for commenting on the exchange functions.

Value

Transcript object.

See Also

corpus_import, corpus_new, import, import_eaf, import_exb, import_textgrid, export_rpraat

Examples

```
library(act)

# Path to the .TextGrid file that you want to read
path <- system.file("extdata", "examplecorpus", "GAT",
     "ARG_I_PAR_Beto.TextGrid", package="act")

# To import a .TextGrid file of your choice:
## Not run:
 path <- "PATH_TO_AN_EXISTING_TEXTGRID_ON_YOUR_COMPUTER"

## End(Not run)

# Make sure to have rPraat installed before you try the following
## Not run:
 # Read TextGrid file with rPraat
```

```
rPraat.tg <- rPraat::tg.read(path)

# Convert to an act transcript
t <- act::import_rpraat(rPraat.tg)

# Change the name and add it to the examplecorpus
t@name <-"rpraat"
newcorpus <- act::transcripts_add(examplecorpus, t)

# Have a look
newcorpus@transcripts[["rpraat"]]

# Alternatively, you can use the general import function
t <- act::import(rPraat.tg)

## End(Not run)
```

`import_srt` *Import a single .srt 'subrib title' object*

Description

Advice: In most situations it is more convenient to use `act::corpus_new`, `act::corpus_import` for importing annotation files.

Usage

```
import_srt(filePath, transcriptName = NULL, tierName = "subtitle")
```

Arguments

`filePath`	Character string; input path of a single 'subribtitle' .srt file.
`transcriptName`	Character string; name of the transcript.
`tierName`	Character string; name of the imported tier

Value

Transcript object.

See Also

corpus_import, corpus_new, import, import_eaf, import_exb, `import_textgrid`, import_rpraat

Examples

```
library(act)

# Path to the .TextGrid file that you want to read
path <- system.file("extdata", "examplecorpus", "GAT",
     "ARG_I_PAR_Beto.TextGrid", package="act")

# To import a .TextGrid file of your choice:
## Not run:
 path <- "PATH_TO_AN_EXISTING_TEXTGRID_ON_YOUR_COMPUTER"

## End(Not run)

# Make sure to have rPraat installed before you try the following
## Not run:
 # Read TextGrid file with rPraat
 rPraat.tg <- rPraat::tg.read(path)

 # Convert to an act transcript
 t <- act::import_rpraat(rPraat.tg)

 # Change the name and add it to the examplecorpus
 t@name <-"rpraat"
 newcorpus <- act::transcripts_add(examplecorpus, t)

 # Have a look
 newcorpus@transcripts[["rpraat"]]

 # Alternatively, you can use the general import function
 t <- act::import(rPraat.tg)

## End(Not run)
```

`import_textgrid` *Import a single 'Praat' .TextGrid file*

Description

Advice: In most situations it is more convenient to use `act::corpus_new`, `act::corpus_import` for importing annotation files.

Usage

```
import_textgrid(
  filePath = NULL,
  fileContent = NULL,
  transcriptName = NULL,
  verbose = TRUE
)
```

Arguments

filePath	Character string; input path of a single 'Praat' .TextGrid file.
fileContent	Vector of character strings; contents of a 'Praat' .TextGrid file read with readLines().
transcriptName	Character string; name of the transcript.
verbose	Logical; if TRUE warnings about encoding issues are printed to the console. If FALSE messages are only stored in t@load.message. Default is TRUE.

Details

Imports the contents of a 'Praat' .TextGrid file and returns a transcript object. The source is either the path to a .TextGrid file or the contents of a .TextGrid file obtained from the @file.content of an existing transcript object by readLines(). The slot @file.content is only filled when the option act.import.storefileContentInTranscript is TRUE (default FALSE). If you pass 'fileContent' you need to pass 'transcriptName' as parameter, too.

Please note:

- Time values of annotations in TextGrids may be below 0 seconds. Negative time values will be recognized corretly in the first place. When exporting transcript object to other formats like 'ELAN' .eaf, 'EXMARaLDA' .exb ect. annotations that are completely before 0 sec will be deleted, annotations that start before but end after 0 sec will be truncated. Please see also the function act::transcripts_cure_single.
- TextGrids and contained tiers may start and end at different times. These times do not need to match each other. The act package does not support start and end times of TextGrids and tiers and will. The default start of a TextGrid will be 0 seconds or the lowest value in case that annotations start below 0 seconds.

Value

Transcript object.

See Also

corpus_import, corpus_new, import, import_eaf, import_exb, import_rpraat

Examples

```
library(act)

# Path to the .TextGrid file that you want to read
path <- system.file("extdata", "examplecorpus", "GAT",
     "ARG_I_PAR_Beto.TextGrid", package="act")

# To import a .TextGrid file of your choice:
## Not run:
path <- "PATH_TO_AN_EXISTING_TEXTGRID_ON_YOUR_COMPUTER"

## End(Not run)
```

```
t <- act::import_textgrid(filePath=path)
t


# Content of a .TextGrid (already read by \code{readLines}),
# e.g. from an existing transcript object:
mycontent <- examplecorpus@transcripts[[1]]@file.content
t <- act::import_textgrid(fileContent=mycontent, transcriptName="test")
t
```

`info` *Information about corpus and transcript objects*

Description

Gives detailed information about the contents of a corpus object or a transcript object that is passed as parameter to the function. In the case that you want to pass a transcript object form a corpus object, make sure that you access the transcript using double [[]] brackets.

Usage

```
info(...)
```

Arguments

`...`	object; either a corpus or a transcript object.

Details

To get summarized information about the transcript and corpus objects use `act::info_summarized`.

Value

List.

See Also

info_summarized

Examples

```
library(act)

act::info(examplecorpus)

act::info(examplecorpus@transcripts[[1]])
```

`info_summarized` *Summarized information about corpus and transcript objects*

Description

Gives summarized information about the contents of a corpus object or a transcript object that is passed as parameter to the function. In the case that you want to pass a transcript object form a corpus object, make sure that you access the transcript using double [[]] brackets.

Usage

```
info_summarized(...)
```

Arguments

`...`	object; either a corpus or a transcript object.

Details

To get more detailed information about the tiers in a corpus object use `act::info`.

Value

List.

See Also

info

Examples

```
library(act)

act::info_summarized(examplecorpus)

act::info_summarized(examplecorpus@transcripts[[1]])
```

layers_convert_column_to_tier
Convert a column layer to tiers

Description

Converts a column in `t@annotations` into proper tiers. For each annotation that has content in the column, a new tier is created with the name `{mainTierName}{tierSeparator}{columnName}`.

Usage

```
layers_convert_column_to_tier(
  t,
  columnName,
  tierSeparator,
  insertAfter = NULL,
  mainTierNameCase = "lower"
)
```

Arguments

`t`	Transcript object.
`columnName`	Character string; name of the column in `t@annotations` to convert.
`tierSeparator`	Character string; separator between main tier name and layer name in the new tier name.
`insertAfter`	Character string or NULL. Controls where the new tier is inserted relative to other tiers of the same speaker: `NULL` (default) appends as last layer, `"main"` inserts directly after the main tier, or a layer name (e.g., `"trans-en"`) to insert after that layer.
`mainTierNameCase`	Character string. Case conversion for the main tier name part: `"lower"` (default), `"upper"`, or `"none"`. `NULL` and `NA` are treated as `"none"`.

Value

The modified transcript object with new tiers and the column removed.

`matrix_load` *Load replacement matrix*

Description

Loads a normalization/replacement matrix from a .csv or .xlsx file.

Usage

```
matrix_load(path = NULL, encoding = "UTF-8")
```

Arguments

`path`	Character string; path to the replacement matrix (.csv or .xlsx file). If argument is left open, the default replacement matrix of the package will be returned.
`encoding`	Character string; encoding of the file (only used for .csv files).

Details

The following placeholders are supported in the `replace` column:

- `\{SPACE\}` is replaced by a single space character `" "`
- `\{NOTHING\}` is replaced by an empty string `""`

Placeholders can be part of a longer string, e.g. `abc\{SPACE\}def` becomes `abc def`.

Value

Data.frame

Examples

```
library(act)

# An example replacement matrix comes with the package.
path <- system.file("extdata", "normalization", "normalization_matrix.csv", package="act")

# Load the matrix
mymatrix <- act::matrix_load(path)

# Have a look at the matrix
colnames(mymatrix)
mymatrix

#the original path of the matrix is stored in the attributes
attr(mymatrix, 'path')
```

matrix_save *Save replacement matrix*

Description

Save replacement matrix

Usage

```
matrix_save(replacementMatrix, path, encoding = "UTF-8")
```

Arguments

replacementMatrix	Data frame; replacement matrix.
path	Character string; path where the matrix will be saved.
encoding	Character string; encoding of the file.

Value

nothing

Examples

```
library(act)

# An example replacement matrix comes with the package.
path <- system.file("extdata", "normalization", "normalization_matrix.csv", package="act")

# Load the matrix
mymatrix <- act::matrix_load(path)

# ' # Create temporary file path
path <- tempfile(pattern = "mymatrix", tmpdir=tempdir(),  fileext = ".csv")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                  "mymatrix.csv")

## End(Not run)

# Save the matrix
act::matrix_save(mymatrix, path=path)
```

media_assign *Assign media file links to transcript objects*

Description

Searches for media files in folders and assigns the links to transcript objects in a corpus. The function uses the name of the transcript to find the media files, e.g. the function assumes that the annotation files have the same name as the media files, except from the suffix/the file type.

Usage

```
media_assign(
  x,
  searchPaths = NULL,
  searchSubfolders = TRUE,
  filterFile = "",
  namesExtractPattern = "",
  transcriptNames = NULL,
  deleteExistingMedia = TRUE,
  onlyUniqueFiles = TRUE,
  readMetadata = FALSE
)
```

Arguments

x	Corpus object.
searchPaths	Vector of character strings; paths where media files should be searched; if path is not defined, the paths given in x@paths.media.files will be used).
searchSubfolders	Logical; if FALSE only the main level of the directory will be scanned for media, if TRUE sub folders will be scanned for media, too.
filterFile	Character string; Regular expression of files to look for.
namesExtractPattern	Character string; Regular Expression to match a part of the transcript name to seach for media files.
transcriptNames	Vector of character strings; Names of the transcripts for which you want to search media files; leave empty if you want to search media for all transcripts in the corpus object.
deleteExistingMedia	Logical; if TRUE existing media links will be deleted, if FALSE existing media links will be preserved and new links will be added.
onlyUniqueFiles	Logical; if TRUE media files with the same name (in different locations) will only be added once; if FALSE all media files found will be added, irrespective of possible doublets.

readMetadata — Logical; if TRUE the read-derived media columns (length.sec, video.*, startsec etc.) are filled via media_metadata_read after assignment. Default FALSE (fast; no file reading).

Details

Only the the file types set in options()$act.media.fileformats.audio and options()$act.media.fileformats.video will be recognized. You can modify these options to recognize other media types.

Value

Corpus object.

See Also

media_select, media_delete, media_path_to_existing_file

Examples

```
library(act)

# Set the folder(s) where your media files are located in the corpus object
# Please be aware that that the example corpus that comes with the package
# does NOT contain media files. Please download the entire example corpus
# with media files if you want to use this function reasonably.
examplecorpus@paths.media.files <- c("", "")

examplecorpus <- act::media_assign(examplecorpus)
```

---

media_build — *Build a media table from file paths*

Description

Builds the core media table (as stored in transcript@media) from a vector of file paths. The generic act columns are set: path, startsec (0), kind ("full"), type ("video"/"audio"/ NA from the file extension). The remaining columns (source.path, clip.id, startsec.source, length.sec, video.width, video.height, video.fps) are set to NA and filled later (e.g. via media_metadata_read). Higher level packages may add further columns afterwards.

Usage

```
media_build(paths)
```

Arguments

paths — Vector of character strings; media file paths.

Value

Data.frame; the core media table with one row per path.

See Also

media_assign, media_select

`media_delete` *Delete media files links from transcript objects*

Description

Delete media files links from transcript objects

Usage

```
media_delete(x, transcriptNames = NULL)
```

Arguments

`x`	Corpus object.
`transcriptNames`	Vector of character strings; Names of the transcripts for which you want to search media files; leave empty if you want to search media for all transcripts in the corpus object.

Value

Corpus object.

See Also

media_assign, media_path_to_existing_file

Examples

```
library(act)

examplecorpus <- act::media_delete(examplecorpus)
```

media_metadata_fill *Fill the read-derived columns of a media table from the files*

Description

For each row whose metadata has not been read yet (`length.sec` is `NA`, unless `force = TRUE`), reads the file via `media_metadata_read` and fills `length.sec`, `source.path`, `clip.id` and the `video.*` columns. `startsec`/ `startsec.source` are only overwritten when the file actually carries an offset (tag or timecode), so the `0` of full recordings is preserved.

Usage

```
media_metadata_fill(media, force = FALSE)
```

Arguments

`media`	Data.frame; a media table as built by `media_build`.
`force`	Logical; if `TRUE` re-reads every row.

Value

Data.frame; the media table with read columns filled.

See Also

media_build, media_metadata_read, media_assign

media_metadata_read *Read act.* metadata and container properties from media files*

Description

Universal reader for media files. Supported formats: mp4, mov, wav, mp3 (via ffprobe) and jpg, jpeg (via exifr/exiftool). Returns a data.frame with one row per file. All columns are always present; columns that do not apply to the file format are `NA`.

Usage

```
media_metadata_read(file, tolerance_sec = 0.05)
```

Arguments

`file`	Character. Single file path or a vector of paths.
`tolerance_sec`	Numeric. Absolute tolerance in seconds for the consistency check between `act.startsec` tag and timecode track. Default is `0.05`.

Details

Column groups:

- act tag columns: `source`, `source.exists`, `startsec`, `startsec.source`, `clip.id`, `timecode`, `comment`
- container columns: `container.format`, `length.sec`, `file.size`, `mtime`
- video columns: `video.width`, `video.height`, `video.fps`, `video.codec`, `video.bitrate`, `video.editlist.offset` (offset in seconds of the video track's edit list, read from the MP4/MOV header: raw time = time shown by players that honour the edit list + offset; 0 without edit list)
- audio columns: `audio.sample.rate`, `audio.channels`, `audio.codec`, `audio.bitrate`
- a file that was read but has no sound track gets `audio.codec = "none"` and `audio.channels = 0`, one without picture track `video.codec = "none"`; `NA` means not read (or reading failed)
- image columns: `image.width`, `image.height`, `bit.depth`

The `comment` column carries consistency notes joined by `;` . Examples: `"ok"`, `"no tags, using timecode"`, `"no tags, no timecode"`, `"source missing on disk"`, `"INCONSISTENT: tag=X.XXX timecode=Y.YYY"`, `"INCONSISTENT: clipID tag=X file=Y"`, `"clipID derived from filename"`, `"unsupported format"`.

Priority for `startsec`: `act.startsec` tag wins. If absent, the timecode track value is used (MP4 tmcd stream). If neither, `NA`.

Value

A data.frame with one row per input file.

See Also

`helper_metadata_ffmpeg_args()`, `helper_metadata_exif_write()`

`media_path_to_existing_file`

Gets the path of a media file for a transcript

Description

Gets the path of a media file for a transcript

Usage

```
media_path_to_existing_file(
  t,
  filterMediaFile = c("(?i).*\\.(mp4|mov)", "(?i).*\\.(aiff|aif|wav)",
    "(?i).*\\.mp3")
)
```

Arguments

t	transcript object; transcript for which you want to get the media path.
filterMediaFile	Vector of character strings; Each element of the vector is a regular expression. Expressions will be checked consecutively. The first match with an existing media file will be used for playing. The default checking order is video > uncompressed audio > compressed audio.

Value

Character string; path to a media file, or NULL if no existing media file has been found.

See Also

media_assign, media_delete

Examples

```
library(act)

# Please be aware that that the example corpus that comes with the package
# does NOT contain media files. Please download the entire example corpus
# with media files if you want to use this function reasonably.

# You can access the media files linked to a transcript directly using
# the object properties.
examplecorpus@transcripts[["SYNC_rotar_y_flexionar"]]@media

# Get only media files of a certain type, e.g. a wav file, and return only the first match:
act::media_path_to_existing_file(examplecorpus@transcripts[["SYNC_rotar_y_flexionar"]],
 filterMediaFile=".*\\.wav")
```

---

media_select *Select media files from a media table*

Description

Filters the rows of a media table (as stored in transcript@media) down to the files that should be used, e.g. for export or playback. The selection is controlled by the act.media.* options and can be overridden per call.

Usage

```
media_select(
  media,
  include = NA,
  exclude = NA,
  apply = TRUE,
```

```
  maxFiles = NULL,
  audioAsFallback = getOption("act.media.audio_as_fallback", FALSE),
  videoMaxFiles = getOption("act.media.video_max"),
  audioMaxFiles = getOption("act.media.audio_max"),
  videoPriority = getOption("act.media.video_priority"),
  audioPriority = getOption("act.media.audio_priority")
)
```

Arguments

media	Data.frame; media table with at least the columns path and type (as in transcript@media).
include	Character string or NA; regular expression, only rows whose path matches are kept. NA keeps all.
exclude	Character string or NA; regular expression, rows whose path matches are removed. NA removes none.
apply	Logical; if TRUE the priority/maximum/audio-fallback selection is applied, if FALSE only include/exclude/maxFiles are applied.
maxFiles	Integer or NULL; overall maximum number of rows returned (NULL or NA = no limit), applied last.
audioAsFallback	Logical; if TRUE audio files are only returned when no video file is present.
videoMaxFiles	Integer, NA or NULL; maximum number of video files (NA/NULL = no limit).
audioMaxFiles	Integer, NA or NULL; maximum number of audio files (NA/NULL = no limit).
videoPriority	Character vector or NULL; regex patterns matched against video file names for prioritization. First matching pattern wins.
audioPriority	Character vector or NULL; file extensions for audio prioritization. First matching extension wins.

Details

The function does not access the file system; it only filters the given table. Existence on disk is not checked (see media_path_to_existing_file for that).

Value

Data.frame; the selected rows of the media table.

See Also

media_assign, media_path_to_existing_file

merge_docx *Merge .docx files in a folder*

Description

Loads all .docx files in a folder and merges them to a singe .docx file.

Usage

```
merge_docx(
  folderInput,
  pathTemplateInput,
  pathOutput = NULL,
  recursive = TRUE,
  filterRegex = NULL
)
```

Arguments

folderInput	Character string; Path to a existing folder containing the .docx files
pathTemplateInput	Character string; Path to .docx file used as a template, where the other files will be inserted.
pathOutput	Character string; Optional. Output path were to save result. If parameter is not set, the print transcripts will only be returned.
recursive	Logical; if TRUE folder input will be searched recursively.
filterRegex	Character string; optional regular expression applied to the matched .docx file paths. Only files matching the expression are included in the merge. Default NULL (no filtering).

Value

merged .docx in officer format,

See Also

export_docx

options_delete *Delete all options set by the package from R options*

Description

Delete all options set by the package from R options

Usage

```
options_delete()
```

Examples

```
library(act)
act::options_delete()
```

options_reset *Reset options to default values*

Description

Reset options to default values

Usage

```
options_reset()
```

Examples

```
library(act)
act::options_reset()
```

options_show *Options of the package*

Description

The package has numerous options that change the internal workings of the package.

Usage

```
options_show(group = NULL)
```

Arguments

group	Character string; optional name of an option group to show. One of "program", "path", "media", "ffmpeg", "import", "export", "parsing", "layout", "tier". If NULL (the default), all options are shown.

Details

There are several options that change the way the package works. They are set globally.

- Use options(name.of.option = value) to set an option.
- Use options()$name.of.option to get the current value of an option.
- Use act::options_reset() to set all options to the default value.
- Use act::options_delete() to clean up and delete all option settings.

Options are organized in groups:

Program: General behavior settings (progress bars, corpus updates).

Paths: Paths to external programs (Praat, ELAN).

Media: Recognized audio and video file extensions and the media selection settings (priority, maximum number, audio fallback).

FFmpeg: Commands and options for media cutting and still extraction.

Import: Settings for reading annotation files.

Export: Settings for file naming and folder structure.

Parsing: Separators, word counting, and pause identification patterns.

Layout: Settings of the alignment engine for print transcripts.

Tier: Tier name patterns (stills tiers, catch-all multimodal tiers).

Value

Nothing.

Examples

```
library(act)
## Not run:
act::options_show()

## End(Not run)
```

search-class *Search object*

Description

This object defines the properties of a search in act. It also contains the results of this search in a specific corpus, if the search has already been run. (Note that you can also create a search without running it immediately). A search object can be run on different corpora.

Some of the slots are defined by the user. Other slots are [READ ONLY], which means that they can be accessed by the user but should not be changed. They contain values that are filled when you execute functions on the object.

Slots

`name` Character string; name of the search. Will be used, for example, as name of the sub folder when creating media cuts

`pattern` Character string; search pattern as a regular expression.

`search.mode` Character string; defines if the original contents of the annotations should be searched or if the full texts should be searched. Slot takes the following values: `content`, `fulltext` (=default, includes both full text modes), `fulltext.byTime`, `fulltext.byTier`.

`search.normalized` logical. if `TRUE` the normalized annotations will be used for searching.

`resultid.prefix` Character string; search results will be numbered consecutively; This character string will be placed before the consecutive numbers.

`resultid.start` Integer; search results will be numbered consecutively; This is the start number of the identifiers.

`filter.transcript.names` Vector of character strings; names of transcripts to include in the search. If the value is `character()` or `""` filter will be ignored.

`filter.transcript.includeRegEx` Character string; Regular expression that defines which transcripts should be INcluded in the search (matching the name of the transcript).

`filter.transcript.excludeRegEx` Character string; Regular expression that defines which transcripts should be EXcluded in the search (matching the name of the transcript).

`filter.tier.names` Vector of character strings; names of tiers to include in the search. If the value is `character()` or `""` filter will be ignored.

`filter.tier.includeRegEx` Character string; Regular expression that defines which tiers should be INcluded in the search (matching the name of the tier).

`filter.tier.excludeRegEx` Character string; Regular expression that defines which tiers should be EXcluded in the search (matching the name of the tier).

`filter.section.startsec` Double; Time value in seconds, limiting the search to a certain time span in each transcript, defining the start of the search window.

`filter.section.endsec` Double; Time value in seconds, limiting the search to a certain time span in each transcript, defining the end of the search window.

`concordance.make` Logical; If a concordance should be created when the search is run.

`concordance.width` Integer; number of characters to include in the concordance.

`cuts.span.beforesec` Double; Seconds how much the cuts (media and print transcripts) should start before the start of the search hit.

`cuts.span.aftersec` Double; Seconds how much the cuts (media and print transcripts) should end after the end of the search hit.

`cuts.column.srt` Character string; name of destination column in the search results data frame where the .srt subtitles will be inserted; column will be created if not present in data frame; set to "" for no insertion.

`cuts.column.printtranscript` Character string; name of destination column in the search results data frame where the print transcripts will be inserted; column will be created if not present in data frame; set to "" for no insertion.

`cuts.printtranscripts` Character string; `[READ ONLY]` All print transcripts for the search results (if generated previously)

`cuts.cutlist.mac` Character string; `[READ ONLY]` 'FFmpeg' cut list for use on a Mac, to cut the media files for the search results.

`cuts.cutlist.win` Character string; `[READ ONLY]` 'FFmpeg' cut list for use on Windows, to cut the media files for the search results.

`results` Data.frame; Results of the search.1

`results.nr` Integer; `[READ ONLY]` Number of search results.

`results.tiers.nr` Integer; `[READ ONLY]` Number of tiers over which the search results are distrubuted.

`results.transcripts.nr` Integer; `[READ ONLY]` Number of transcripts over which the search results are distrubuted.

`x.name` Character string; `[READ ONLY]` name of the corpus object on which the search has been run.

See Also

export_docx

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish.
mysearch <- act::search_new(examplecorpus, pattern= "yo")
mysearch
# Search in normalized content vs. original content
mysearch.norm  <- act::search_new(examplecorpus, pattern="yo", searchNormalized=TRUE)
mysearch.org   <- act::search_new(examplecorpus, pattern="yo", searchNormalized=FALSE)
mysearch.norm@results.nr
mysearch.org@results.nr

# The difference is because during normalization capital letters will be converted
# to small letters. One annotation in the example corpus contains a "yo" with a
# capital letter:
mysearch <- act::search_new(examplecorpus, pattern="yO", searchNormalized=FALSE)
```

```
mysearch@results$hit

# Search in full text vs. original content.
# Full text search will find matches across annotations.
# Let's define a regular expression with a certain span.
# Search for the word "no" 'no' followed by a "pero" 'but'
# in a distance ranging from 1 to 20 characters.
myRegEx <- "\\bno\\b.{1,20}pero"
mysearch <- act::search_new(examplecorpus, pattern=myRegEx, searchMode="fulltext")
mysearch
mysearch@results$hit
```

---

search_concordance *Make concordance for search results*

Description

Make concordance for search results

Usage

```
search_concordance(x, s, searchNormalized = TRUE)
```

Arguments

x	Corpus object.
s	Search object.
searchNormalized	Logical; if TRUE function will search in the normalized content, if FALSE function will search in the original content.

Value

Search object.

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish
# Search without creating the concordance immediately.
# This is for example useful if you are working with a large corpus, since
# making the concordance may take a while.
mysearch <- act::search_new(examplecorpus, pattern="yo", concordanceMake=FALSE)
mysearch@results[1,]


# The results do not contain the concordance, it is only 15 columns
```

```
ncol(mysearch@results)

# Make the concordance
mysearch.new <- act::search_concordance(x=examplecorpus, s=mysearch)
ncol(mysearch.new@results)
```

---

search_cuts *Create print transcripts, media cutlists and srt subtitles for all search results*

Description

This function will call the following functions:

- `act_cuts_printtranscript` to create print transcripts,
- `act::cuts_media` to create FFmpeg cutlist to make media snippets,
- `act::search_cuts_srt()` to create .srt subtitles, for all search results.

For a detailed description including examples please refer to the documentation of the indidival functions. They also offer some more parameters than this functions. If you want to use those, call the functions individually.

Usage

```
search_cuts(
  x,
  s,
  cutSpanBeforesec = NULL,
  cutSpanAftersec = NULL,
  folderOutput = NULL,
  style = NULL
)
```

Arguments

`x`	Corpus object.
`s`	Search object.
`cutSpanBeforesec`	Double; Start the cut some seconds before the hit to include some context; the default NULL will take the value as set in @cuts.span.beforesec of the search object.
`cutSpanAftersec`	Double; End the cut some seconds before the hit to include some context; the default NULL will take the value as set in @cuts.span.beforesec of the search object.

folderOutput	Character string; if parameter is not set, the print transcripts will only be inserted in s@results; if the path to a existing folder is given transcripts will be saved in '.txt' format.
style	Transcript profile: the name of a profile, the path of a profile file or a profile read with helper_style_read. NULL uses the profile "act".

Value

Search object;

Examples

```
library(act)

# IMPORTANT: In the example corpus all transcripts are assigned media links.
# The actual media files are, however, not included in when installing the package
# due to size limitations of CRAN.
# But you may download the media files separately.
# Please see the section 'examplecorpus' for instructions.
# --> You will need the media files to execute the following example code.

## Not run:
 # Search
 mysearch <- act::search_new(examplecorpus, pattern="yo")

 # Create print transcripts, media cutlists and .srt subtitles
 # for all search results
 test <- act::search_cuts(x=examplecorpus, s=mysearch)

 # Display all print transcripts on screen from @cuts.printtranscripts
 cat(test@cuts.printtranscripts)

 # Display cutlist on screen from @cuts.cutlist.mac
 cat(test@cuts.cutlist.mac)

 # Display .srt subtitles
 cat(test@results[, mysearch@cuts.column.srt])

## End(Not run)
```

---

search_cuts_media *Create cut lists for 'FFmpeg'*

Description

This function creates FFmpeg commands to cut media files for each search results. If you want to execute the commands (and cut the media files) you need to have FFmpeg installed on you computer. To install FFmpeg you can follow the instructions given in the vignette 'installation-ffmpeg'. Show the vignette with vignette("installation-ffmpeg").

Usage

```
search_cuts_media(
  x,
  s,
  exportMedia = TRUE,
  exportStills = TRUE,
  exportThumbnail = TRUE,
  cutSpanBeforesec = NULL,
  cutSpanAftersec = NULL,
  folderOutput = NULL,
  filterMediaInclude = "",
  filterMediaExclude = "",
  videoFastPositioning = TRUE,
  videoCodecCopy = FALSE,
  audioAsMP3 = FALSE,
  audioPanning = NULL,
  audioNormalize = FALSE,
  outputOS = c("mac", "win"),
  outputFileName = "FFMPEG_cutlist",
  videoCodecCopyFail = "encode"
)
```

Arguments

x	Corpus object; Please note: all media paths for a transcript need to be given as a list in the corpus object in `corpus@transcripts[[ ]]@media` . You can use the respective media functions. .
s	Search object.
exportMedia	Logical; If `TRUE` media cuts (audio/video) will be included in the cut list.
exportStills	Logical; If `TRUE` stills (image) will be included in the cut list. For creating the list of stills to be exported see search_cuts_media.
exportThumbnail	Logical; If `TRUE` a thumbnail image (.jpg, at most `act.ffmpeg.thumbnail.max_height` pixels high) will be exported to the main folder. The name will be the same as the media cut. By default the time value is the start of the search hit. To change this set time values in seconds in a column called `thumbnails` in `s@results`.
cutSpanBeforesec	Double; Start the media cut some seconds before the hit to include some context; the default `NULL` will take the value as set in @cuts.span.beforesec of the search object.
cutSpanAftersec	Double; End the media cut some seconds before the hit to include some context; the default `NULL` will take the value as set in @cuts.span.beforesec of the search object.
folderOutput	Character string; path to folder where files will be written.

`filterMediaInclude`
: Character string; regular expression to match only some of the media files in `corpus@transcripts[[ ]]@media`.

`filterMediaExclude`
: Character string; regular expression to exclude some of the media files in `corpus@transcripts[[ ]]@media`. Empty string (default) excludes none.

`videoFastPositioning`
: Logical; no longer used. Video cuts, stills and thumbnails always seek exactly (two-stage seek, see helper_ffmpeg_args). Kept for compatibility.

`videoCodecCopy`
: Logical; If `TRUE` the video is copied instead of encoded (fast mode, e.g. for many search hits): a copy can only start at a keyframe, so each video cut starts at the keyframe before the cut (up to the keyframe distance of the camera earlier, often a few seconds) and keeps the full resolution; sound and picture stay in sync, the act.* metadata carry the real start. See helper_ffmpeg_args_clip and helper_ffmpeg_keyframe.

`audioAsMP3`
: Logical; If `TRUE` audio cuts will be converted to .mp3 files (bit rate `act.ffmpeg.audio.bitrate`). If `FALSE` (default) the audio format is preserved.

`audioPanning`
: Integer; 0=leave audio as is (ch1&ch2) , 1=only channel 1 (ch1), 2=only channel 2 (ch2), 3=both channels separated (ch1&ch2), 4=all three versions (ch1&ch2, ch1, ch2). This setting will override the option made in 'act.ffmpeg.channels_from_column' .

`audioNormalize`
: Logical; If `TRUE` and option `act.ffmpeg.audio.normalize` is `TRUE`, a loudnorm filter is appended to the audio filter chain. Default is `FALSE`.

`outputOS`
: Vector of character Strings; Saves FFMpeg cut list in format for `"win"`=windows, `"mac"`=apple ox/linux.

`outputFileName`
: Character String; Name of the cut list.

`videoCodecCopyFail`
: Character; what happens when `videoCodecCopy` is `TRUE` and no keyframe can be found for a cut (ffprobe missing or failed, source not readable): `"abort"` stops with an error, `"encode"` encodes this cut instead and shows a warning. Default is `"encode"`: a cut list is run later and should not fail as a whole because of one source.

Details

Cut lists
The commands are collected in cut lists. The cut lists will be stored in different ways:

- A cut list for for ALL search results will be stored in `s@cuts.cutlist.mac` to be used on MacOS and `s@cuts.cutlist.win` to be used on Windows.
- Individual cut lists for EACH search result will be stored in additional columns in the data frame `s@results`. The cut lists that can be executed in the Terminal (Apple) or the Command Line Interface (Windows).

Input media files
The function will use all files in `corpus@transcripts[[ ]]@media`. Therefore you will need to set the options `filterMediaInclude` filtering for which input media files you want to create the cuts.

The filter is a regular expression, e.g. `'\.(wav|aif)'` for '.wav' and '.aif' audio files or `'\.mp4'` for '.mp4' video files.

Output file names
The file names of the cuts are taken from the `resultID` column in the search results table. To change this, modify the contents of this column or define a different column using `options(act.export.filename.fromColumnNam`
By default, the device descriptor of the original media will be appended to the file name. It is taken from the `devices` column of `corpus@transcripts[[ ]]@media`. To change this, modify that column.

Output format
The output format is predefined by in the options:

- Video cuts are built by helper_ffmpeg_args_clip: picture and sound start at the time ELAN shows (edit list alignment), H.264 yuv420p with the options `act.ffmpeg.video.bitrate`, `.crf`, `.keyframe_interval`, `.max_height`, `.max_width` and AAC with `act.ffmpeg.audio.bitrate`.
- Audio cuts are built by helper_ffmpeg_args_audio: copied without filter, otherwise written with the bit depth of the source; mp3 with `act.ffmpeg.audio.bitrate`.
- The channel versions (`_ch1`, `_ch2`) are stereo files with the left resp. right channel on both sides; the filters come from helper_audio_filter_parts.
- Stills and thumbnails are built by helper_ffmpeg_args (exact seek, JPG quality from `act.ffmpeg.image.quality`; thumbnails at most `act.ffmpeg.thumbnail.max_height` pixels high); each still gets the act.* EXIF comment through an exiftool line in the cut list (skipped when exiftool is not installed).

Extract stills
To extract stills the time information and the file names need to be stored in the search object in the results data frame, e.g. in `s@results`. The information needs to be contained in a vector. The elements of the vector are the time values in seconds, the name attribute will be used as file name, e.g. `myStills <- c(still1=1.0, still2=2.0)`. The vector needs stored for each search result in the column called `stills.values` as a list, e.g code `s@results$stills.values[[1]] <- myStills`. Stills will be stored in a sub folder called `stills` by default, if not otherwise defined in the column `stills.folder`. Please see the section `options_show` to customize the behavior of the function.

Cut list lines
The lines of the cut lists are written by helper_cutlist_lines. They call ffmpeg as set in the option `act.cutlist.ffmpeg` (default `"ffmpeg"`, found via the PATH of the machine that runs the cut list).

Value

Search object; cut lists will be stored in `s@cuts.cutlist.mac` and `s@cuts.cutlist.win`.

See Also

search_cuts_media, `vignette("install_ffmpeg", package = "act")`

Examples

```
library(act)
```

```
# IMPORTANT: In the example corpus all transcripts are assigned media links.
# The actual media files are, however, not included in when installing the package
# due to size limitations of CRAN.
# But you may download the media files separately.
# Please see the section 'examplecorpus' for instructions.
# --> You will need the media files to execute the following example code.

## Not run:
 # Search
 mysearch <- act::search_new(examplecorpus, pattern="yo")

 # Create cut lists
 mysearch <- act::search_cuts_media (x=examplecorpus, s=mysearch)

 # Check results for Mac:
 # Get entire cut list for Mac and display on screen,
 # so you can copy&paste this into the Terminal
 mycutlist <- mysearch@cuts.cutlist.mac
 cat(mycutlist)
 # Cut list for first search result
 mycutlist <- mysearch@results$cuts.cutlist.mac[[1]]
 cat(mycutlist)

 # Check results for Windows:
 # Get entire cut list for Mac and display on screen,
 # so you can copy&paste this into the CLI
 mycutlist <- mysearch@cuts.cutlist.win
 cat(mycutlist)
 # Cut list for first search result
 mycutlist <- mysearch@results$cuts.cutlist.win[[1]]
 cat(mycutlist)

 # It is, however, more convenient to specify the argument 'folderOutput' in order to get
 # the cut list as a (executable) file/batch list.

## End(Not run)
```

`search_cuts_printtranscript` *Create print transcripts for all search results*

Description

Print transcripts in the style of conversation analysis will be created for each search result. The transcripts will be inserted into the column defined in `s@cuts.column.printtranscript`. All transcripts will be stored in `s@cuts.printtranscripts`.

Usage

```
search_cuts_printtranscript(
```

```
  x,
  s,
  exportTxt = TRUE,
  exportDocx = TRUE,
  headerInsertSource = TRUE,
  cutSpanBeforesec = 0,
  cutSpanAftersec = 0,
  folderOutput = NULL,
  style = NULL
)
```

Arguments

x	Corpus object.
s	Search object.
exportTxt	Logical; If TRUE and folderOutput is set, .txt transcripts will be written to disk. The print transcripts are always inserted into the column s@cuts.column.printtranscript of s@results, independent of this parameter.
exportDocx	Logical; If TRUE and folderOutput is set, .docx transcripts will be written to disk.
headerInsertSource	Logical; if TRUE standard information about the source and location of the sequence will be inserted after the heading.
cutSpanBeforesec	Double; Start the cut some seconds before the hit to include some context; the default NULL will take the value as set in @cuts.span.beforesec of the search object.
cutSpanAftersec	Double; End the cut some seconds before the hit to include some context; the default NULL will take the value as set in @cuts.span.beforesec of the search object.
folderOutput	Character string; path to an existing folder for file export. If NULL (default), no files will be written to disk; the print transcripts are inserted only into s@results. If set, .txt and/or .docx files will be written depending on exportTxt and exportDocx.
style	Transcript profile: the name of a profile, the path of a profile file or a profile read with helper_style_read. NULL uses the profile "act".

Details

The layout of the print transcripts comes from a transcript profile (parameter style); see export_docx.

Value

Search object;

Examples

```
library(act)

# Search
mysearch <- act::search_new(examplecorpus, pattern="yo")

# Create print transcripts for all search results
test <- act::search_cuts_printtranscript (x=examplecorpus, s=mysearch)

# Display all print transcripts on screen from @cuts.printtranscripts
cat(test@cuts.printtranscripts)

# Display all print transcripts from results data frame
cat(test@results[,mysearch@cuts.column.printtranscript])
cat(test@results[,mysearch@cuts.column.printtranscript])

# Only single print transcript from results data frame
cat(test@results[1,mysearch@cuts.column.printtranscript])

# Create print transcript snippets including 1 sec before and 5 sec after
mysearch@cuts.span.beforesec =1
mysearch@cuts.span.aftersec = 5
test <- act::search_cuts_printtranscript (x=examplecorpus,
s=mysearch)

# Display all transcript snippets on screen
cat(test@results[,mysearch@cuts.column.printtranscript])

# Only insert print transcripts into the column, no file export.
# The print transcripts are always inserted into the column,
# independent of exportTxt and exportDocx.
test <- act::search_cuts_printtranscript (x           = examplecorpus,
                                          s           = mysearch,
                                          exportTxt   = FALSE,
                                          exportDocx  = FALSE,
                                          folderOutput= NULL)
cat(test@results[1, mysearch@cuts.column.printtranscript])
```

`search_cuts_srt` *Create .srt subtitles for all search results*

Description

Subtitles in 'Subrib Title' .srt format will be created for each search result. The subtitles will be inserted into the column defined in `s@cuts.column.srt`.

Usage

```
search_cuts_srt(
```

```
  x,
  s,
  cutSpanBeforesec = NULL,
  cutSpanAftersec = NULL,
  folderOutput = NULL,
  speakerShow = TRUE,
  speakerWidth = 3,
  speakerEnding = ":"
)
```

Arguments

x	Corpus object.
s	Search object.
cutSpanBeforesec	Double; Start the cut some seconds before the hit to include some context; the default NULL will take the value as set in @cuts.span.beforesec of the search object.
cutSpanAftersec	Double; End the cut some seconds before the hit to include some context; the default NULL will take the value as set in @cuts.span.beforesec of the search object.
folderOutput	Character string; path to an existing folder for file export. If NULL (default), no files will be written to disk; the srt subtitles are inserted only into s@results. If set, .srt files will be written to disk in addition.
speakerShow	Logical; if TRUE name of speaker will be shown before the content of the annotation.
speakerWidth	Integer; width of speaker abbreviation, -1 for full name without shortening.
speakerEnding	Character string; string that is added at the end of the speaker name.

Details

Span
If you want to extend the cut before or after each search result, you can modify `@cuts.span.beforesec` and `@cuts.span.aftersec` in your search object.

Value

Search object;

Examples

```
library(act)

# Search
mysearch <- act::search_new(examplecorpus, pattern="yo")

# Create srt subtitles for all search results
```

```
test <- act::search_cuts_srt (x=examplecorpus, s=mysearch)

# Display srt subtitle of first three results
cat(test@results[1:3, mysearch@cuts.column.srt])

# Create srt subtitle including 1 sec before and 5 sec after
mysearch@cuts.span.beforesec = 1
mysearch@cuts.span.aftersec = 5
test <- act::search_cuts_srt (x=examplecorpus,
        s=mysearch)

# Display srt subtitle of first  results
cat(test@results[1,mysearch@cuts.column.srt])
```

`search_makefilter` *Makes a filter for transcript and tier names*

Description

Search a corpus object and return the names of all transcripts and tiers that match the given parameters. You can define parameters to include and/or exclude transcripts and tiers based on their names. All parameters passed to the function will be combined.

Usage

```
search_makefilter(
  x,
  filterTranscriptNames = NULL,
  filterTranscriptIncludeRegex = NULL,
  filterTranscriptExcludeRegex = NULL,
  filterTierNames = NULL,
  filterTierIncludeRegex = NULL,
  filterTierExcludeRegex = NULL
)
```

Arguments

`x`	Corpus object.
`filterTranscriptNames`	Vector of character strings; Names of the transcripts that you want to include; to include all transcripts in the corpus object leave parameter empty or set to `character()` or `""`.
`filterTranscriptIncludeRegex`	Character string; as regular expression, include transcripts matching the expression.
`filterTranscriptExcludeRegex`	Character string; as regular expression, exclude transcripts matching the expression.

`filterTierNames`
: Vector of character strings; Names of the tiers that you want to include; to include all tiers in the corpus object leave parameter empty or set to `character()` or `""`.

`filterTierIncludeRegex`
: Character string; as regular expression, include tiers matching the expression.

`filterTierExcludeRegex`
: Character string; as regular expression, exclude tiers matching the expression.

Details

This functions is useful if you want to use functions of the package such as `transcripts_update_normalization`, `transcripts_update_fulltexts`, `corpus_export` and limit them to only some of the transcripts.

Value

List of character vectors. `$filterTranscriptNames` contains all transcript names in the corpus matching the expressions, `$filterTierNames` contains all tier names in the corpus matching the expressions.

See Also

search_new, search_run, search_sub

Examples

```
library(act)

# Search all transcripts that have "ARG" (ignoring case sensitivity) in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTranscriptIncludeRegex="(?i)arg")
myfilter$transcriptNames

# Search all transcripts that don't have "ARG" in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTranscriptExcludeRegex="ARG")
myfilter$transcriptNames

# Search all tiers that have an "A" or an "a" in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTierIncludeRegex="(?i)A")
myfilter$tierNames

# Search all tiers that have a capital "A" in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTierIncludeRegex="A")
myfilter$tierNames

# In which transcripts do these tiers occur?
myfilter$transcriptNames

# Let's check the first of the transcripts, if this is really the case...
examplecorpus@transcripts[[myfilter$transcriptNames[1]]]@tiers
```

search_new *Create a new search*

Description

Creates a new search object and runs the search in a corpus object. Only 'x' and 'pattern' are obligatory. The other arguments can be left to their default values.

Usage

```
search_new(
  x,
  pattern,
  searchMode = c("content", "fulltext", "fulltext.byTime", "fulltext.byTier"),
  searchNormalized = TRUE,
  name = "mysearch",
  resultidPrefix = "result",
  resultidStart = 1,
  filterTranscriptNames = NULL,
  filterTranscriptIncludeRegex = NULL,
  filterTranscriptExcludeRegex = NULL,
  filterTierNames = NULL,
  filterTierIncludeRegex = NULL,
  filterTierExcludeRegex = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  concordanceMake = TRUE,
  cutSpanBeforesec = 0,
  cutSpanAftersec = 0,
  runSearch = TRUE
)
```

Arguments

x	Corpus object; basis in which will be searched.
pattern	Character string; search pattern as regular expression.
searchMode	Character string; takes the following values: `content`, `fulltext` (=default, includes both full text modes), `fulltext.byTime`, `fulltext.byTier`.
searchNormalized	Logical; if `TRUE` function will search in the normalized content, if `FALSE` function will search in the original content.
name	Character string; name of the search. Will be used, for example, as name of the sub folder when creating media cuts.
resultidPrefix	Character string; search results will be numbered consecutively; This character string will be placed before the consecutive numbers.

`resultidStart` Integer; search results will be numbered consecutively; This is the start number of the identifiers.

`filterTranscriptNames`
Vector of character strings; names of transcripts to be included.

`filterTranscriptIncludeRegex`
Character string; as regular expression, limit search to certain transcripts matching the expression.

`filterTranscriptExcludeRegex`
Character string; as regular expression, exclude certain transcripts matching the expression.

`filterTierNames`
Vector of character strings; names of tiers to be included in the search.

`filterTierIncludeRegex`
Character string; as regular expression, limit search to certain tiers matching the expression.

`filterTierExcludeRegex`
Character string; as regular expression, exclude certain tiers matching the expression.

`filterSectionStartsec`
Double; start time of region for search.

`filterSectionEndsec`
Double; end time of region for search.

`concordanceMake`
Logical; if `TRUE` concordance will be added to search results. The width of the concordance is controlled by the option `act.concordanceWidth` (default `NULL` uses the search class default of 120 characters).

`cutSpanBeforesec`
Double; Start the media and transcript cut some seconds before the hit to include some context, the default is `0`.

`cutSpanAftersec`
Double; End the media and transcript cut some seconds before the hit to include some context, the default is `0`.

`runSearch` Logical; if `TRUE` search will be run in corpus object, if `FALSE` only the search object will be created.

Value

Search object.

See Also

search_run, search_makefilter, search_sub

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish.
mysearch <- act::search_new(examplecorpus, pattern= "yo")
mysearch
# Search in normalized content vs. original content
mysearch.norm  <- act::search_new(examplecorpus, pattern="yo", searchNormalized=TRUE)
mysearch.org   <- act::search_new(examplecorpus, pattern="yo", searchNormalized=FALSE)
mysearch.norm@results.nr
mysearch.org@results.nr

# The difference is because during normalization capital letters will be converted
# to small letters. One annotation in the example corpus contains a "yo" with a
# capital letter:
mysearch <- act::search_new(examplecorpus, pattern="yO", searchNormalized=FALSE)
mysearch@results$hit

# Search in full text vs. original content.
# Full text search will find matches across annotations.
# Let's define a regular expression with a certain span.
# Search for the word "no" 'no' followed by a "pero" 'but'
# in a distance ranging from 1 to 20 characters.
myRegEx <- "\\bno\\b.{1,20}pero"
mysearch <- act::search_new(examplecorpus, pattern=myRegEx, searchMode="fulltext")
mysearch
mysearch@results$hit
```

`search_openresult_inelan` *Open a search result in 'ELAN'*

Description

The function creates an temporary .eaf file and a .psfx file that locates the search hit. These files will then be opened in ELAN. To make this function work you need to have 'ELAN' installed on your computer and tell the act package where ELAN is located. Therefore you need to set the path to the ELAN executable in the option 'act.path.elan' using `options(act.path.elan='PATHTOYOURELANEXECUTABLE')`.

Usage

```
search_openresult_inelan(
  x,
  s,
  resultid,
  openOriginal = FALSE,
  overwrite = TRUE
)
```

Arguments

x	Corpus object.
s	Search object.
resultid	Integer; Number of the search result (row in the data frame s@results) to be opened.
openOriginal	Logical; if TRUE the function will check if the original annotation file was an .eaf file and if it still exists in the original location. If so, the function will not create a temporary .eaf file but open the original file. Warning: The original .pfsx file (if it exists) will be overwritten.
overwrite	Logical; if TRUE (default) an existing .pfsx file will be overwritten. If FALSE an existing .pfsx file is kept.

Details

WARNING: This function will overwrite existing .psfx files.

Credits: Thanks to Han Sloetjes for feedback on the structure of the temporary .pfsx files. He actually made the code work.

Examples

```
library(act)

mysearch <- act::search_new(x=examplecorpus, pattern = "yo")

# You can only use this function if you have installed ELAN on our computer.
## Not run:
options(act.path.elan='PATHTOYOURELANEXECUTABLE')
act::search_openresult_inelan(x=examplecorpus, s=mysearch, resultid=1, TRUE)

## End(Not run)
```

---

search_openresult_inpraat

Open a search result in 'Praat'

Description

The function remote controls 'Praat' with a 'Praat' script. It opens a search result in the 'Praat' TextGrid Editor.

Usage

```
search_openresult_inpraat(
  x,
  s,
  resultid,
  play = TRUE,
  close = FALSE,
  filterMediaFile = c("(?i).*\\.(aiff|aif|wav)", "(?i).*\\.mp3"),
  delay = 0.5
)
```

Arguments

x	Corpus object.
s	Search object.
resultid	Integer; Number of the search result (row in the data frame s@results) to be played.
play	Logical; If TRUE selection will be played.
close	Logical; If TRUE TextGrid editor will be closed after playing.
filterMediaFile	Vector of character strings; Each element of the vector is a regular expression. Expressions will be checked consecutively. The first match with an existing media file will be used for playing. The default checking order is uncompressed audio > compressed audio.
delay	Double; Time in seconds before the section will be opened in Praat. This is useful if Praat opens but the section does not. In that case increase the delay.

Details

'Praat' receives the script through its own command line option --send (from 'Praat' 6.1 on): set the path to 'Praat' with 'options(act.path.praat = ...)'. If 'Praat' is not running, it is started (see vignette("praat_remote_control", package = "act")).

See Also

vignette("praat_remote_control", package = "act")

Examples

```
library(act)

mysearch <- act::search_new(x=examplecorpus, pattern = "pero")

# You can only use this functions if you have installed 'Praat'
# and located it properly in the package options.
## Not run:
act::search_openresult_inpraat(x=examplecorpus, s=mysearch, resultid=1, TRUE, TRUE)

## End(Not run)
```

search_openresult_inquicktime
Open a search result in 'Quicktime' (and play it)

Description

The function remote controls 'Quicktime' by using an Apple Script. It opens a search result in 'Quicktime' and plays it.

Usage

```
search_openresult_inquicktime(
  x,
  s,
  resultid,
  play = TRUE,
  close = FALSE,
  bringToFront = TRUE,
 filterFile = c("(?i).*\\.(mp4|mov)", "(?i).*\\.(aiff|aif|wav)", "(?i).*\\.mp3")
)
```

Arguments

x	Corpus object.
s	Search object.
resultid	Integer; Number of the search result (row in the data frame s@results) to be played.
play	Logical; If TRUE selection will be played.
close	Logical; if TRUE the Quicktime player will be closed after playing the cuts.
bringToFront	Logical; if TRUE the Quicktime player will be activated and placed before all other windows.
filterFile	Vector of character strings; Each element of the vector is a regular expression. Expressions will be checked consecutively. The first match with an existing media file will be used for playing. The default checking order is video > uncompressed audio > compressed audio.

Details

Note: You need to be on a Mac to use this function.

Span
If you want to extend the cut before or after each search result, you can modify @cuts.span.beforesec and @cuts.span.aftersec in your search object.

Value

Logical; TRUE if media file has been played, or FALSE if not.

Examples

```
library(act)

mysearch <- act::search_new(x=examplecorpus, pattern = "pero")

# You can only use this function if you are on a Mac.
# In addition, you need to have downloaded the example media.
## Not run:

# Assign media files
examplecorpus@paths.media.files <- c("FOLDERWHEREMEDIAFILESARELOCATED")
examplecorpus <- act::media_assign(examplecorpus)

# Play the media for the first search result
act::search_openresult_inquicktime(x=examplecorpus,
s=mysearch,
resultid = 1,
play=TRUE,
close=TRUE)

# Play all search results after one another.
 for (i in 1:nrow(mysearch@results)) {
  print(mysearch@results$content[i])
  act::search_openresult_inquicktime(x=examplecorpus,
s=mysearch,
resultid = i,
play=TRUE,
close=TRUE)
 }

## End(Not run)
```

`search_playresults_inquicktime` *Open all search results in 'Quicktime' and play them*

Description

The function remote controls 'Quicktime' by using an Apple Script. It opens consecutively all search results in 'Quicktime' and plays them.

Usage

```
search_playresults_inquicktime(x, s, bringToFront = FALSE)
```

Arguments

x	Corpus object.
s	Search object.

bringToFront Logical; if TRUE the Quicktime player will be activated and placed before all other windows.

Details

Note: You need to be on a Mac to use this function.

Value

No return value.

Examples

```
library(act)

mysearch <- act::search_new(x=examplecorpus, pattern = "pero")

# You can only use this function if you are on a Mac.
# In addition, you need to have downloaded the example media files.
## Not run:
 # Assign media files
 examplecorpus@paths.media.files <- c("FOLDERWHEREMEDIAFILESARELOCATED")
 examplecorpus <- act::media_assign(examplecorpus)

 # Create print transcripts. This is not necessary.
 # But its nice to see them when playing all results.
 mysearch <- act::search_cuts_printtranscript (x=examplecorpus, s=mysearch)

 # Play all search results
 act::search_playresults_inquicktime(x=examplecorpus, s=mysearch)

## End(Not run)
```

---

search_results_export *Exports search results*

Description

Search results from a search object will be saved to a Excel-XLSX or a CSV (comma separated values) file. By default a XLSX file will be saved. If you want to save a CSV file, use saveAsCSV=TRUE. Please note:

- The function will '=' signs at the beginning of annotation by ".=". This is because the content would be interpreted as the beginning of a formula (leading to an error).
- In the case of writing to an excel file, line breaks will be replaced by "|". This is because line breaks will lead to an error.

Usage

```
search_results_export(
  s,
  path,
  sheetName = "data",
  saveAsCSV = FALSE,
  encoding = "UTF-8",
  separator = ";",
  overwrite = TRUE
)
```

Arguments

s	Search object. Search object containing the results you wish to export.
path	Character string; path where file will be saved. Please add the suffix '.csv' or '.xlsx' to the file name.
sheetName	Character string, set the name of the excel sheet.
saveAsCSV	Logical; if TRUE results will be saved as CSV file; Logical; if FALSE a XLS file will be saved.
encoding	Character string; text encoding for CSV files.
separator	Character; single character that is used to separate the columns.
overwrite	Logical; if TRUE existing files will be overwritten

Examples

```
library(act)

# Search
mysearch <- act::search_new(examplecorpus, pattern="yo")
nrow(mysearch@results)

# Create temporary file path
path <- tempfile(pattern = "searchresults", tmpdir = tempdir(),
        fileext = ".xlsx")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
 path <- tempfile(pattern = "searchresults",
       tmpdir = "PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
       fileext = ".xlsx")

## End(Not run)

# Save search results
act::search_results_export(s=mysearch, path=path)

# Do your coding of the search results somewhere outside of act
# ...
```

```
# Load search results
mysearch.import <- act::search_results_import(path=path)
nrow(mysearch.import@results)
```

`search_results_import` *Import search results*

Description

Search results will be imported from an Excel '.xlsx' file or a comma separated values '.csv' file into a search object.

Usage

```
search_results_import(
  path,
  revertReplacements = TRUE,
  sheetName = "data",
  encoding = "UTF-8",
  separator = ";"
)
```

Arguments

`path`	Character string; path to file from where data will be loaded.
`revertReplacements`	Logical, when exporting search results from act, '=' at the beginning of lines are replaced by '.=", and in numbers the decimal separator '.' is replaced by a ",". If `TRUE`, this replacement will be reverted when importing search results.
`sheetName`	Character string, set the name of the excel sheet containing the data.
`encoding`	Character string; text encoding in the case of CVS files.
`separator`	Character; single character that is used to separate the columns in CSV files.

Value

Search object.

Examples

```
library(act)

# Search
mysearch <- act::search_new(examplecorpus, pattern="yo")
nrow(mysearch@results)

# Create temporary file path
```

```
path <- tempfile(pattern = "searchresults", tmpdir = tempdir(),
        fileext = ".xlsx")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
 path <- tempfile(pattern = "searchresults",
       tmpdir = "PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
       fileext = ".xlsx")

## End(Not run)

# Save search results
act::search_results_export(s=mysearch, path=path)

# Do your coding of the search results somewhere outside of act
# ...

# Load search results
mysearch.import <- act::search_results_import(path=path)
nrow(mysearch.import@results)
```

`search_run` *Run a search*

Description

Runs a search, based on an existing search object s, in a corpus object x.

Usage

```
search_run(x, s)
```

Arguments

x	Corpus object.
s	Search object.

Value

Search object.

See Also

search_new, search_makefilter, search_sub

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish.
# Only create the search object without running the search.
mysearch <- act::search_new(x=examplecorpus, pattern= "yo", runSearch=FALSE)

# Run the search
mysearch <- act::search_run(x=examplecorpus, s=mysearch)
mysearch
mysearch@results$hit

# Search Only in tiers called "A", in any transcript
mysearch@filter.tier.names <-"A"
mysearch@filter.transcript.names <-""
mysearch <- act::search_run(x=examplecorpus, s=mysearch)
cbind(mysearch@results$transcriptName, mysearch@results$tierName, mysearch@results$hit)

# Search Only in tiers called "A", only in transcript "ARG_I_PER_Alejo"
mysearch@filter.tier.names <-"A"
mysearch@filter.transcript.names <-"ARG_I_PER_Alejo"
mysearch <- act::search_run(x=examplecorpus, s=mysearch)
cbind(mysearch@results$transcriptName, mysearch@results$tierName, mysearch@results$hit)
```

---

search_searchandopen_inpraat

Search corpus and open first result in Praat

Description

The function remote controls 'Praat' with a 'Praat' script. It first searches your corpus object and uses the first search hit. The corresponding TextGrid will be opened in the 'Praat' TextGrid Editor and the search hit will be displayed.

Usage

```
search_searchandopen_inpraat(x, pattern)
```

Arguments

x	Corpus object.
pattern	Character string; search pattern as regular expression.

Details

'Praat' receives the script through its own command line option `--send` (from 'Praat' 6.1 on): set the path to 'Praat' with 'options(act.path.praat = ...)'. If 'Praat' is not running, it is started (see `vignette("praat_remote_control", package = "act")`).

See Also

```
vignette("praat_remote_control", package = "act")
```

Examples

```
library(act)

# You can only use this functions if you have installed 'Praat'
# and located it properly in the package options.
## Not run:
act::search_searchandopen_inpraat(x=examplecorpus, "pero")

## End(Not run)
```

---

search_stills *Search stills for a search*

Description

Only `x`, `s` and `pattern` are obligatory. The other arguments can be left to their default values.

Usage

```
search_stills(
  x,
  s,
  pattern,
  searchMode = c("content", "fulltext", "fulltext.byTime", "fulltext.byTier"),
  searchNormalized = FALSE,
  filterTierNames = NULL,
  filterTierIncludeRegex = "still",
  filterTierExcludeRegex = NULL,
  resultids = NULL,
  stillsFolder = "stills"
)
```

Arguments

`x`	Corpus object; basis in which will be searched.
`s`	Search object.
`pattern`	Character string; search pattern as regular expression.
`searchMode`	Character string; takes the following values: `content`, `fulltext` (=default, includes both full text modes), `fulltext.byTime`, `fulltext.byTier`.

`searchNormalized`
: Logical; if `TRUE` function will search in the normalized content, if `FALSE` function will search in the original content.

`filterTierNames`
: Vector of character strings; names of tiers to be included.

`filterTierIncludeRegex`
: Character string; as regular expression, limit search to certain tiers matching the expression.

`filterTierExcludeRegex`
: Character string; as regular expression, exclude some tiers from search matching the expression.

`resultids`
: Vector of Integer; By default all results in the search object will be processed. if `resultids` is set, still will only be searched for the results specified.

`stillsFolder`
: Character string; name of the sub folder where to store the stills. folder will be created recursively

Value

Search object.

See Also

search_cuts_media, search_new

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish.
mysearch <- act::search_new(examplecorpus, pattern= "yo")
mysearch
# Search in normalized content vs. original content
mysearch.norm  <- act::search_new(examplecorpus, pattern="yo", searchNormalized=TRUE)
mysearch.org   <- act::search_new(examplecorpus, pattern="yo", searchNormalized=FALSE)
mysearch.norm@results.nr
mysearch.org@results.nr

# The difference is because during normalization capital letters will be converted
# to small letters. One annotation in the example corpus contains a "yo" with a
# capital letter:
mysearch <- act::search_new(examplecorpus, pattern="yO", searchNormalized=FALSE)
mysearch@results$hit

# Search in full text vs. original content.
# Full text search will find matches across annotations.
# Let's define a regular expression with a certain span.
# Search for the word "no" 'no' followed by a "pero" 'but'
# in a distance ranging from 1 to 20 characters.
myRegEx <- "\\bno\\b.{1,20}pero"
mysearch <- act::search_new(examplecorpus, pattern=myRegEx, searchMode="fulltext")
```

```
mysearch
mysearch@results$hit
```

---

search_sub *Add a sub search to a prior search*

Description

This function starts from the results of a prior search and performs a sub search for a temporal co-occurence. In the sub search all results from the prior search will be checked. The sub search will check annotations in other tiers that temporally overlap with the original search result. Those annotation will be checked if they match a search pattern. If so, the search hit of the sub search will be added to a new column in the original search results data frame.

Usage

```
search_sub(
  x,
  s,
  pattern,
  searchMode = c("content", "fulltext", "fulltext.byTime", "fulltext.byTier"),
  searchNormalized = TRUE,
  filterTierIncludeRegex = "",
  filterTierExcludeRegex = "",
  destinationColumn = "subsearch",
  deleteEmptyLines = FALSE,
  excludeHitsInSameTier = TRUE,
  collapseString = " | "
)
```

Arguments

x	Corpus object.
s	Search object.
pattern	Character string; search pattern as regular expression
searchMode	Character string; takes the following values: `content`, `fulltext` (=default, includes both full text modes), `fulltext.byTime`, `fulltext.byTier`.
searchNormalized	Logical; if `TRUE` function will search in the normalized content, if `FALSE` function will search in the original content.
filterTierIncludeRegex	Character string; limit search to tiers that match the regular expression
filterTierExcludeRegex	Character string; limit search to tiers that match the regular expression

`destinationColumn`
: Character string; name of column where results of sub search will be stored

`deleteEmptyLines`
: Logical; if `TRUE` search results will be deleted for which the sub search does not give any results

`excludeHitsInSameTier`
: Logical; if `TRUE` the function will not add hits from the same tier as the original search result; if `FALSE` hits from the same tier as the original search result will be included.

`collapseString`
: Character string; Characters that will be used to separate multiple search hits

Value

Search object.

See Also

search_new, search_run, search_makefilter

Examples

```
library(act)

# Lets search for instances where participants laugh together
# First search for annotations that contain laughter (in original content)
myRegEx <- "(\\brie\\b|\\briendo\\b)"
mysearch <- act::search_new(x=examplecorpus,
       pattern=myRegEx,
       searchNormalized = FALSE)
mysearch@results.nr

# Now perform sub search, also on laughs/laughing
test <- act::search_sub(x=examplecorpus,
      s=mysearch,
      pattern=myRegEx)

# Check the co-occurring search hits
test@results$subsearch
```

---

search_transcript_content

Search in original content of a single transcript

Description

Search in original content of a single transcript

Usage

```
search_transcript_content(t, s)
```

Arguments

t	Transcript object; transcript to search in.
s	Search object.

Value

Data.frame data frame with search results.

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish.
# Only create the search object without running the search.
mysearch <- act::search_new(x=examplecorpus, pattern= "yo", runSearch=FALSE)

# Run the search

df <- act::search_transcript_content(t=examplecorpus@transcripts[[3]],
s=mysearch)
nrow(df)
```

---

search_transcript_fulltext

Search in full text of a single transcript

Description

Search in full text of a single transcript

Usage

```
search_transcript_fulltext(t, s)
```

Arguments

t	Transcript object; transcript to search in.
s	Search object.

Value

Data.frame data frame with search results.

Examples

```
library(act)

# Search for the 1. Person Singular Pronoun in Spanish.
# Only create the search object without running the search.
mysearch <- act::search_new(x=examplecorpus, pattern= "yo", runSearch=FALSE)

# Run the search
df <- act::search_transcript_fulltext(t=examplecorpus@transcripts[[3]],
                                      s=mysearch)
nrow(df)
```

`tiers_add` *Add tiers*

Description

Adds a tiers in all transcript objects of a corpus. If tiers should be added only in certain transcripts, set the parameter `filterTranscriptNames`. In case that you want to select transcripts by using regular expressions use the function `act::search_makefilter` first.

Usage

```
tiers_add(
  x,
  tierName,
  tierType = c("IntervalTier", "TextTier"),
  positionAbsolute = NULL,
  destinationTier = NULL,
  relativePositionToDestinationTier = 0,
  insertOnlyIfDestinationExists = FALSE,
  filterTranscriptNames = NULL,
  skipIfTierAlreadyExists = TRUE
)
```

Arguments

`x`	Corpus object.
`tierName`	Character string; names of the tier to be added.
`tierType`	Character string; type of the tier to be added.
`positionAbsolute`	Integer; Absolute position where the tier will be inserted. Value 1 and values beloe 1 will insert the tier in the first position; To insert the tier at the end, leave 'positionAbsolute' and 'destinationTier' open.
`destinationTier`	Character string; insert the tier relative to this tier.

`relativePositionToDestinationTier`
: Integer; position relative to the destination tier; 1=immediately after; 0 and -1=immediately before; bigger numbers are also allowed.

`insertOnlyIfDestinationExists`
: Logical; if `TRUE` the new tier will only be added if the destination tier 'destinationTier' exists in the transcript object. If `FALSE` the new tier will only be added in any case. If the destination tier 'destinationTier' does not exist in the transcript object, the tier will be inserted at the end.

`filterTranscriptNames`
: Vector of character strings; names of the transcripts to be modified. If left open, the tier will be added to all transcripts in the corpus.

`skipIfTierAlreadyExists`
: Logical; if `TRUE` the new tier will be skipped if a tier with this name already exists in the transcript; if `FALSE` an error will be raised.

Details

You can either insert the new tier at a specific position (e.g. 'positionAbsolute=1') or in relation to a existing tier (e.g. destinationTier='speaker1'). To insert a tier at the end, leave 'positionAbsolute' and 'destinationTier' open.

Results will be reported in `@history` of the transcript objects.

Value

Corpus object.

See Also

tiers_delete, tiers_rename, tiers_convert, tiers_sort

Examples

```
library(act)

# --- Add new interval tier.
# Since not position is set it will be inserted in the end, by default.
x <- act::tiers_add(examplecorpus,
     tierName="TEST")
#check results
tail(x@transcripts[[1]]@history, 1)
#have a look at the first transcript
x@transcripts[[1]]@tiers
#--> New tier is inserted in the end.

# --- Add new interval tier in position 2
x <- act::tiers_add(examplecorpus,
     tierName="TEST",
     positionAbsolute=2)
#check results
tail(x@transcripts[[1]]@history, 1)
```

```
#have a look at the first transcript
x@transcripts[[1]]@tiers
#--> New tier is inserted as second tier.


# --- Add new interval tier at the position of "Entrevistador", only if this tier exists,
# If the destination tier does not exist, the new tier will NOT be inserted.

#Have a look at the first and the second transcript.
examplecorpus@transcripts[[1]]@tiers
#Transcript 1 does contain a tier "Entrevistador" in the first position.
examplecorpus@transcripts[[2]]@tiers
#Transcript 2 does contain a tier "Entrevistador" in the first position.

#Insert new tier
x <- act::tiers_add(examplecorpus,
     tierName="TEST",
     destinationTier="Entrevistador",
     relativePositionToDestinationTier=0,
     insertOnlyIfDestinationExists=TRUE)

#Check results
tail(x@transcripts[[1]]@history, 1)
#Have a look at the transcript 1:
# Tier 'TEST' was in first position (e.g. where 'Entrevistador' was before).
x@transcripts[[1]]@tiers
#Have a look at the transcript 2:
#Tier 'TEST' was not inserted, since there was no destination tier 'Entrevistador'.
x@transcripts[[2]]@tiers

# --- Add new interval tier AFTER tier="Entrevistador"
# If the destination tier does not exist, the new tier will be inserted at the end in any case.
x <- act::tiers_add(examplecorpus,
     tierName="TEST",
     destinationTier="Entrevistador",
     relativePositionToDestinationTier=1,
     insertOnlyIfDestinationExists=FALSE)
#check results
tail(x@transcripts[[1]]@history, 1)
#Have a look at the transcript 1:
# Tier 'TEST' was inserted after the tier 'Entrevistador'.
x@transcripts[[1]]@tiers
#Have a look at the transcript 2:
#Tier 'TEST' was insertedat the end.
x@transcripts[[2]]@tiers
```

---

tiers_all *All tiers in a corpus*

Description

Merges tiers from all transcripts in a corpus object into a data frame.

Usage

```
tiers_all(x, compact = TRUE)
```

Arguments

x	Corpus object.
compact	Logical; if TRUE a condensed overview will be returned, if FALSE a detailed overview will be returned.

Value

Data frame

Examples

```
library(act)

#Get data frame with all tiers
alltiers <- act::tiers_all(examplecorpus)
alltiers

#Get data frame with a simplified version
alltiers <- act::tiers_all(examplecorpus, compact=TRUE)
alltiers
```

---

tiers_convert *Convert tiers*

Description

Converts tier types between 'interval' and 'point' tier. Applies to all tiers in all transcript objects of a corpus. If only certain transcripts or tiers should be affected set the parameter `filterTranscriptNames`.
In case that you want to select transcripts by using regular expressions use the function `act::search_makefilter` first.

Usage

```
tiers_convert(
  x,
  intervalToPoint = FALSE,
  pointToInterval = FALSE,
  filterTierNames = NULL,
  filterTranscriptNames = NULL
)
```

Arguments

x	Corpus object.
intervalToPoint	Logical; if TRUE interval tiers will be converted to point/text tiers.
pointToInterval	Logical; if TRUE point/text tiers will be converted to interval tiers.
filterTierNames	Vector of character strings; names of the tiers to be included.
filterTranscriptNames	Vector of character strings; names of the transcripts to be checked. If left open, all transcripts will be checked

Details

Note: When converting from interval > point tier, the original end times of the annotations will be lost definitely.

Value

Corpus object.

See Also

tiers_add, tiers_delete, tiers_rename, tiers_sort, helper_tiers_new_table, helper_tiers_sort_table

Examples

```
library(act)

# Check the names and types of the existing tiers in the first two transcripts
examplecorpus@transcripts[[1]]@tiers
examplecorpus@transcripts[[2]]@tiers

# Convert interval tiers to point tiers
newcorpus <- act::tiers_convert(examplecorpus, intervalToPoint=TRUE)

# the names and types of the existing tiers
newcorpus@transcripts[[1]]@tiers
newcorpus@transcripts[[2]]@tiers

# Convert point tiers to interval tiers
newcorpus <- act::tiers_convert(newcorpus, pointToInterval=TRUE)

# Note: In this round trip conversion from 'interval > point > interval tier'
# the original end times of the annotations get lost (when converting from interval > point).
```

`tiers_delete` *Delete tiers*

Description

Deletes tiers in all transcript objects of a corpus. If only tiers in certain transcripts should be affected set the parameter `filterTranscriptNames`. In case that you want to select tiers and/or transcripts by using regular expressions use the function `act::search_makefilter` first. Results will be reported in `@history` of the transcript objects.

Usage

```
tiers_delete(x, tierNames, filterTranscriptNames = NULL)
```

Arguments

`x`	Corpus object.
`tierNames`	Character string; names of the tiers to be deleted.
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be modified. If left open, all transcripts will be checked.

Value

Corpus object.

See Also

tiers_add, tiers_rename, tiers_convert, tiers_sort, helper_tiers_new_table, helper_tiers_sort_table

Examples

```
library(act)

# get info about all tiers
all.tiers <- act::info(examplecorpus)$tiers

# tiers 'A' and 'B' occur 6 times in 6 transcripts
all.tiers["A", "tier.count"]
all.tiers["B", "tier.count"]

# delete tiers
tierNames <- c("A", "B")
x<- examplecorpus
x <- act::tiers_delete(examplecorpus, tierNames=tierNames)
tail(x@transcripts[[1]]@history, 1)

# tiers 'A' and 'B' do not occur anymore
act::info(x)$tiers$tierName
```

`tiers_rename` *Rename tiers*

Description

Renames all tiers in all transcript objects of a corpus. If only certain transcripts should be affected set the parameter `filterTranscriptNames`. In case that you want to select transcripts by using regular expressions use the function `act::search_makefilter` first.

Usage

```
tiers_rename(x, searchPattern, searchReplacement, filterTranscriptNames = NULL)
```

Arguments

`x`	Corpus object.
`searchPattern`	Character string; search pattern as regular expression.
`searchReplacement`	Character string; replacement string.
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be included.

Details

The tiers will only be renamed if the resulting names preserve the uniqueness of the tier names. Results will be reported in `@history` of the transcript objects. Please be aware that this function is not optimized for speed and may take quite a while to run, depending on the size of your corpus object.

Value

Corpus object.

See Also

tiers_add, tiers_convert, tiers_rename, tiers_sort, helper_tiers_new_table, helper_tiers_sort_table

Examples

```
library(act)

# Check the names of the existing tiers in the first two transcripts
examplecorpus@transcripts[[1]]@tiers$name
examplecorpus@transcripts[[2]]@tiers$name

x <- act::tiers_rename(examplecorpus, "Entrevistador", "E")

x@transcripts[[1]]@tiers$name
```

```
x@transcripts[[2]]@tiers$name
```

`tiers_sort` *Reorder tiers in all transcripts of a corpus*

Description

Reorder the positions of tiers in all transcripts of a corpus object. The ordering of the tiers will be done according to a vector of exact tier names defined in 'sortVector' (no regular expressions). If only certain transcripts or tiers should be affected set the parameter `filterTranscriptNames`. In case that you want to select transcripts by using regular expressions use the function `act::search_makefilter` first.

Usage

```
tiers_sort(
  x,
  sortVector,
  filterTranscriptNames = NULL,
  tiersAddMissing = FALSE,
  tiersDelete = FALSE
)
```

Arguments

`x`	Corpus object.
`sortVector`	Vector of character strings; exact tier names (no regular expressions). The order within the vector presents the new order of the tiers. Use "*" to indicate where tiers that are not present in the sort vector but in the transcript should be inserted (the escaped form, a backslash followed by a star, is accepted as well).
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be included.
`tiersAddMissing`	Logical; if `TRUE` all tiers that are given in 'the 'sortVector' but are missing in the transcripts will be added.
`tiersDelete`	Logical; if `TRUE` tiers that are not named in 'sortVector' will be deleted. Otherwise they will be inserted at the end of the table or at the position defined by "*" in 'sortVector'.

Value

Corpus object.

See Also

tiers_add, tiers_convert, tiers_delete, tiers_rename, helper_tiers_new_table, helper_tiers_sort_table

Examples

```
library(act)

# Check the order of the existing tiers in the first two transcripts
examplecorpus@transcripts[[1]]@tiers$name[order(examplecorpus@transcripts[[1]]@tiers$position)]
examplecorpus@transcripts[[2]]@tiers$name[order(examplecorpus@transcripts[[2]]@tiers$position)]

# Get tier names to create the sort vector
sortVector <- c(examplecorpus@transcripts[[1]]@tiers$name,
                examplecorpus@transcripts[[2]]@tiers$name)

# Revert the vector for demonstration.
sortVector <- sortVector[length(sortVector):1]

# This will only reorder the tiers.
examplecorpus <- act::tiers_sort(x=examplecorpus,
sortVector=sortVector)

# Check again the order of the tiers
examplecorpus@transcripts[[1]]@tiers$name[order(examplecorpus@transcripts[[1]]@tiers$position)]
examplecorpus@transcripts[[2]]@tiers$name[order(examplecorpus@transcripts[[2]]@tiers$position)]

# This will reorder the tiers and additionally add tiers that are given
# in the sort vector but not present in the transcript.
examplecorpus <- act::tiers_sort(x=examplecorpus,
sortVector=sortVector,
tiersAddMissing=TRUE)
# Check again the order of the tiers
examplecorpus@transcripts[[1]]@tiers$name[order(examplecorpus@transcripts[[1]]@tiers$position)]
examplecorpus@transcripts[[2]]@tiers$name[order(examplecorpus@transcripts[[2]]@tiers$position)]

# Insert a tier called "newTier" into all transcripts in the corpus:
for (t in examplecorpus@transcripts) {
 sortVector <- c(t@tiers$name, "newTier")
 examplecorpus <- act::tiers_sort(x=examplecorpus,
sortVector=sortVector,
filterTranscriptNames=t@name,
tiersAddMissing=TRUE)
}
# Check for example the first transcript: it now contains a tier called "newTier"
examplecorpus@transcripts[[1]]@tiers


# To get more examples and information about sorting see 'helper_tiers_sort_table()'.
```

`transcript-class` *transcript object*

Description

A transcript object contains the annotations of a loaded annotation file and some meta data . In addition, it contains information that is auto generated by the act package, which is necessary for some functions (e.g. the full text search)

Details

Some of the slots are defined by the user. Other slots are [READ ONLY], which means that they can be accessed by the user but should not be changed. They contain values that are filled when you execute functions on the object.

Slots

`name` Character string; [READ ONLY] Name of the transcript, generated from the annotation file name.

`file.path` Character string; [READ ONLY] Original location of the annotation file.

`file.encoding` Character string; [READ ONLY] Encoding applied to the file when reading.

`file.type` Character string; [READ ONLY] Type of the original annotation file/object, e.g. 'eaf' or 'textgrid' for files and 'rpraat' for a rPraat .TextGrid object.

`file.content` Character string; [READ ONLY] Content of the original annotation file/object.

`file.state` List; [READ ONLY] State of the file the transcript was read from or last written to: `path`, `hash` (fingerprint, see helper_file_hash_compute), `size`, `mtime`, `time` (see helper_file_state). Compared with the file on disk before it is overwritten (helper_file_state_changed). Empty for a transcript without a file of its own.

`import.result` Character string; [READ ONLY] Information about the success of the import of the annotation file.

`load.message` Character string; [READ ONLY] Possibly messages about errors that occurred on importing the annotation file.

`length.sec` Double; [READ ONLY] Duration of the transcript in seconds.

`tiers` Data.frame; [READ ONLY] Table with the tiers. To modify the tiers it is highly recommended to use functions of the package to ensure for consistency of the data.

`annotations` Data.frame; Table with the annotations.

`media` Data.frame; Table of the media files that correspond to this transcript object. Core columns: `path` (absolute file path), `source.path` (path of the source recording a clip was cut from), `clip.id` (clip identifier), `startsec` (start of the file in the original time reference, 0 for full recordings), `startsec.source` (where startsec came from: "tag"/"timecode"/"corpus"), `length.sec` (file duration), `video.width`, `video.height`, `video.fps`, `video.codec`, `video.bitrate`, `audio.sample.rate`, `audio.channels`, `audio.codec`, `audio.bitrate`, `container.format`, `file.size` (bytes), `kind` ("full" or "clip"), `type` ("video" or "audio"). Columns that are not yet read may be NA. Additional columns may be present.

`normalization.signature` Character string; [READ ONLY] Hash signature of the inputs that produced the current normalization (annotations content + search/replace rows of the normalization matrix + layout marker options). Used to detect when re-normalization is needed.

`fulltext.signature` Character string; [READ ONLY] Hash signature of the inputs that produced the current fulltexts (annotations + separators + tier filter). Used to detect when re-creation of the fulltexts is needed.

`fulltext.filter.tier.names` Vector of character strings; names of tiers that were included in the full text..

`fulltext.bytime.orig` Character string; [READ ONLY] full text of the transcript based on the ORIGINAL content of the annotations, sorting the annotations by TIME

`fulltext.bytime.norm` Character string; [READ ONLY] full text of the transcript based on the NORMALIZED content of the annotations, sorting the annotations by TIME

`fulltext.bytier.orig` Character string; [READ ONLY] full text of the transcript based on the ORIGINAL content of the annotations, sorting the annotations first by TIERS and then by time

`fulltext.bytier.norm` Character string; [READ ONLY] full text of the transcript based on the NORMALIZED content of the annotations, sorting the annotations first by TIERS and then by time

`history` List; [READ ONLY] History of the modifications made to the transcript object.

Examples

```
library(act)

examplecorpus@transcripts[[1]]
```

`transcripts_add` *Add transcripts to a corpus*

Description

Add a single or multiple transcript objects to a corpus object.

Usage

```
transcripts_add(
  x,
  ...,
  skipDuplicates = FALSE,
  createNormalization = TRUE,
  createFulltext = TRUE,
  assignMedia = TRUE
)
```

Arguments

x	Corpus object
...	transcript object, list of transcript objects, corpus object.
skipDuplicates	Logical; If FALSE double transcripts will be renamed to make the names unique, if TRUE double transcripts will not be added.
createNormalization	Logical; if TRUE the normalized content will be created.
createFulltext	Logical; if TRUE full text will be created.
assignMedia	Logical; if TRUE the folder(s) specified in @paths.media.files of your corpus object will be scanned for media.

Details

The name of the transcript objects have to be unique in the act package. The @name attribute of each transcript object will be set as identifier in the list of transcripts in the corpus object. By default, transcripts with non unique names will be renamed. If you prefer to import.skipDoubleFiles, set the parameter skipDuplicates=TRUE. Skipped/renamed transcripts will be reported in

Value

Corpus object

Examples

```
library(act)

# get one of the already existing transcript in the examplecorpus
newtrans <- examplecorpus@transcripts[[1]]

# add this transcript to the examplecorpus
newcorpus <- act::transcripts_add(examplecorpus, newtrans)

# compare the two corpus objects
length(examplecorpus@transcripts)
length(newcorpus@transcripts)

names(examplecorpus@transcripts)
names(newcorpus@transcripts)
```

---

transcripts_cure *Cure all transcript objects in a corpus*

Description

Transcript object may contain errors, e.g. because of defect annotation input files or user modifications. This function may cure some of these errors in all transcript objects of a corpus.

- Annotations with reversed times: annotations with `endsec` lower than `startsec` will be deleted.
- Overlapping annotations: earlier annotations will end where the next annotation starts.
- Annotations below 0 sec: Annotations that are starting and ending before 0 sec will be deleted; Annotations starting before but ending after 0 sec will be truncated.
- Transcript length zero: If `@length.sec` is missing or `<= 0` it will be set to the largest annotation endsec.
- Zero-duration annotations: In IntervalTiers, annotations with `startsec == endsec` will be merged (if multiple at same time), then extended to the next annotation, or deleted if no room is available.
- Missing tiers: Tiers that are present in the annotations but missing in the list of tiers in `@tiers` of the transcript object will be added.

Usage

```
transcripts_cure(
  x,
  filterTranscriptNames = NULL,
  annotationsTimesReversed = TRUE,
  annotationsOverlap = TRUE,
  annotationsTimesBelowZero = TRUE,
  transcriptLengthZero = TRUE,
  annotationsZeroDuration = TRUE,
  tiersMissing = TRUE,
  warning = FALSE
)
```

Arguments

`x`	Corpus object.
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be included.
`annotationsTimesReversed`	Logical; If `TRUE` annotations with reversed times will be deleted
`annotationsOverlap`	Logical; If `TRUE` overlapping annotations will be corrected.
`annotationsTimesBelowZero`	Logical; If `TRUE` annotations before 0 sec will be corrected.
`transcriptLengthZero`	Logical; If `TRUE` a transcript with `@length.sec <= 0` will be corrected.
`annotationsZeroDuration`	Logical; If `TRUE` zero-duration annotations in IntervalTiers will be cured.

tiersMissing	Logical; If TRUE tiers missing in @tiers slot of the transcript object will be added.
warning	Logical; If TRUE a warning notice will be shown upon correction.

Value

Corpus object;

See Also

transcripts_cure_single

Examples

```
library(act)

# The example corpus does not contain any errors.
# But let's use the function anyway.
x<- act::transcripts_cure(examplecorpus)
tail(x@transcripts[[1]]@history, 1)

# See \code{act::cure_transcript} for actual examples.
```

---

transcripts_cure_single

Cure a single transcript

Description

Transcript object may contain errors, e.g. because of defect annotation input files or user modifications. This function may cure some of these errors.

- Annotations with reversed times: annotations with endsec lower than startsec will be deleted.
- Overlapping annotations: earlier annotations will end where the next annotation starts.
- Annotations below 0 sec: Annotations that are starting and ending before 0 sec will be deleted; Annotations starting before but ending after 0 sec will be truncated.
- Transcript length zero: If @length.sec is missing or <= 0 it will be set to the largest endsec of any annotation (or a small default).
- Zero-duration annotations: In IntervalTiers, annotations with startsec == endsec will be cured. Multiple zero-duration annotations at the same time point are merged by concatenating their content with a single space. The resulting annotation is then extended to the start of the next annotation (or to @length.sec if it is the last one). If no room is available (next annotation starts at the same time), the annotation is deleted.
- Missing tiers: Tiers that are present in the annotations but missing in the list of tiers in @tiers of the transcript object will be added.

Usage

```
transcripts_cure_single(
  t,
  annotationsTimesReversed = TRUE,
  annotationsOverlap = TRUE,
  annotationsTimesBelowZero = TRUE,
  transcriptLengthZero = TRUE,
  annotationsZeroDuration = TRUE,
  tiersMissing = TRUE,
  warning = FALSE
)
```

Arguments

t	Transcript object.
annotationsTimesReversed	Logical; If TRUE annotations with reversed times will be deleted
annotationsOverlap	Logical; If TRUE overlapping annotations will be corrected.
annotationsTimesBelowZero	Logical; If TRUE annotations before 0 sec will be corrected.
transcriptLengthZero	Logical; If TRUE a transcript with `@length.sec <= 0` will be corrected to the largest annotation endsec.
annotationsZeroDuration	Logical; If TRUE zero-duration annotations in IntervalTiers will be merged/extended/deleted.
tiersMissing	Logical; If TRUE tiers missing in `@tiers` slot of the transcript object will be added.
warning	Logical; If TRUE a warning notice will be shown upon correction.

Value

Transcript object;

See Also

transcripts_cure

Examples

```
library(act)

# --- annotationsTimesReversed: will be deleted
# get example transcript and reverse the times of an annotation
t <- examplecorpus@transcripts[[1]]
t@annotations$startsec[1] <- 20
t@annotations$endsec[1] <- 10
t2 <- act::transcripts_cure_single(t)
```

```
tail(t2@history, n=1)


# --- annotationsTimesBelowZero: will be deleted or start at 0 sec
t <- examplecorpus@transcripts[[1]]
t@annotations$startsec[1] <- -2
t@annotations$endsec[1]   <- -1
t2 <- act::transcripts_cure_single(t)
tail(t2@history, n=1)

t <- examplecorpus@transcripts[[1]]
t@annotations$startsec[2] <- -5
t2 <- act::transcripts_cure_single(t)
tail(t2@history, n=1)


# --- annotationsOverlap: will end where the next starts
t<- examplecorpus@transcripts[[1]]
t@annotations <- t@annotations[order(t@annotations$tierName, t@annotations$startsec), ]
t@annotations$endsec[1] <- 8
t2 <- act::transcripts_cure_single(t)
tail(t2@history, n=1)


# --- tiersMissing: will be added to @tiers in transcript object
t<- examplecorpus@transcripts[[1]]
t@annotations <- t@annotations[order(t@annotations$tierName, t@annotations$startsec), ]
t@annotations$tierName[1] <- "NEW"
t2 <- act::transcripts_cure_single(t)
tail(t2@history, n=1)
t2@tiers
# compare with original tiers
t@tiers


# --- several things at once
t<- examplecorpus@transcripts[[1]]
t@annotations <- t@annotations[order(t@annotations$tierName, t@annotations$startsec), ]
# annotation completely below 0 sec
t@annotations$startsec[1] <- -6
t@annotations$endsec[1]   <- -5
# annotation starts before but ends after 0 sec
t@annotations$startsec[2] <- -3
# annotation with reversed times
t@annotations$startsec[3] <- 6.9
t@annotations$endsec[3]   <- -6.8
# annotation overlaps with next annotation
t@annotations$endsec[6]   <- 9
# new tier, missing tier list
t@annotations$tierName[8] <- "NEW"
t2 <- act::transcripts_cure_single(t, warning=TRUE)
tail(t2@history, n=1)
```

```
examplecorpus@transcripts[[1]]@history
```

`transcripts_delete` *Delete transcripts from a corpus*

Description

Delete transcript objects from a corpus object. You need to name the transcripts to delete directly in the parameter 'transcriptNames'. If you want to delete transcripts based on a search pattern (regular expression) use `act::search_sub` first.

Usage

```
transcripts_delete(x, transcriptNames)
```

Arguments

`x`	Corpus object
`transcriptNames`	Vector of character strings; names of the transcript object to be deleted.

Value

Corpus object

Examples

```
library(act)

# delete two transcripts by their name
test <- act::transcripts_delete(examplecorpus,
c("BOL_CCBA_SP_MeryGaby1",
"BOL_CCBA_SP_MeryGaby2"))

# compare the the original and modified corpus object
length(examplecorpus@transcripts)
length(test@transcripts)
setdiff(names(examplecorpus@transcripts), names(test@transcripts))

# delete transcripts that match a filter, e.g. all transcripts from Bolivia "BOL_"
myfilter <- act::search_makefilter(examplecorpus, filterTranscriptIncludeRegex = "BOL_")
test <- act::transcripts_delete(examplecorpus,
myfilter$transcriptNames)

# compare the the original and modified corpus object
length(examplecorpus@transcripts)
length(test@transcripts)
setdiff(names(examplecorpus@transcripts), names(test@transcripts))
```

`transcripts_filter` *Filter a single transcript: Extract*

Description

Filter all transcript objects in a corpus and return the filtered corpus object. It is possible to filter out temporal sections and tiers. In case that you want to select tiers by using regular expressions use the function `act::search_makefilter` first.

Usage

```
transcripts_filter(
  x,
  filterTranscriptNames = NULL,
  filterTranscriptsRestict = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  timesPreserve = TRUE,
  sort = c("none", "tier>startsec", "startsec>tier")
)
```

Arguments

`x`	Corpus object;
`filterTranscriptNames`	Vector of character strings; names of transcripts to remain. If left unspecified, all transcripts will remain in the transcripts.
`filterTranscriptsRestict`	Vector of character strings; names of transcripts to which filters will be applied. If left unspecified, all transcripts will be filtered.
`filterTierNames`	Vector of character strings; names of tiers to remain in the transcripts. If left unspecified, all tiers will remain in the transcripts.
`filterSectionStartsec`	Double, start of selection in seconds.
`filterSectionEndsec`	Double, end of selection in seconds.
`timesPreserve`	Logical; Parameter is used if `filterSectionStartsec` it set. If `TRUE` start times will be preserved, if `FALSE` the selection will start from 0.
`sort`	Logical; Annotations will be sorted: 'none' (=no sorting), 'tier>startsec' (=sort first by tier, then by startsec), 'startsec>tier' (=sort first by startsec, then by tier)

Value

Corpus object;

Examples

```
library(act)

# Filter corpus to only contain some tiers
all.tierNames <- unique(act::tiers_all(examplecorpus)$name)
some.tierNames <- all.tierNames[1:10]
x <- act::transcripts_filter(examplecorpus, filterTierNames=some.tierNames)
tail(x@transcripts[[1]]@history, 1)
```

---

transcripts_filter_remove_single

Filter a single transcript: Remove

Description

Filter a transcript object and return the filtered transcript object. It is possible to REMOVE temporal sections and tiers. In case that you want to select tiers by using regular expressions use the function `act::search_makefilter` first.

Usage

```
transcripts_filter_remove_single(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  sort = c("none", "tier>startsec", "startsec>tier")
)
```

Arguments

`t`	Transcript object.
`filterTierNames`	Vector of character strings; names of tiers to be remain in the transcripts. If left unspecified, all tiers will remain in the transcript exported.
`filterSectionStartsec`	Double, start of selection in seconds.
`filterSectionEndsec`	Double, end of selection in seconds.
`sort`	Logical; Annotations will be sorted: 'none' (=no sorting), 'tier>startsec' (=sort first by tier, then by startsec), 'startsec>tier' (=sort first by startsec, then by tier)

Value

Transcript object;

Examples

```
library(act)

# get an example transcript
t1 <- examplecorpus@transcripts[[1]]

# --- Filter by tiers
# The example transcript contains two tiers that contain four annotations each.
t1@tiers
table(t1@annotations$tierName)

# Filter transcript to only contain annotations of the FIRST tier
t2 <- act::transcripts_filter_single(t1, filterTierNames=t1@tiers$name[1])
t2@tiers
table(t2@annotations$tierName)

# Use act::search_makefilter() first to get the tier names,
# in this case search for tiers with a capital 'I',
# which is the second tier, called 'ISanti'
myfilter <- act::search_makefilter(examplecorpus,
    filterTranscriptNames=t2@name,
    filterTierIncludeRegex="I"
    )
t2 <- act::transcripts_filter_single(t1, filterTierNames=myfilter$tierNames)
t2@tiers
table(t2@annotations$tierName)

# --- Filter by time section
# only set start of section (until the end of the transcript)
t2 <- act::transcripts_filter_single(t1, filterSectionStartsec=6)
cbind(t2@annotations$startsec,t2@annotations$endsec)

# only set end of section (from the beginning of the transcript)
t2 <- act::transcripts_filter_single(t1, filterSectionEndsec=8)
cbind(t2@annotations$startsec,t2@annotations$endsec)

# set start and end of section
t2 <- act::transcripts_filter_single(t1,
  filterSectionStartsec=6,
  filterSectionEndsec=8)
cbind(t2@annotations$startsec,t2@annotations$endsec)

# set start and end of section, start new times from 0
t2 <- act::transcripts_filter_single(t1,
 filterSectionStartsec=6,
 filterSectionEndsec=8)
cbind(t2@annotations$startsec,t2@annotations$endsec)
```

`transcripts_filter_single` *Filter a single transcript: Extract*

Description

Filter a transcript object and return the filtered transcript object. It is possible to EXTRACT temporal sections and tiers. In case that you want to select tiers by using regular expressions use the function `act::search_makefilter` first.

Usage

```
transcripts_filter_single(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  timesPreserve = TRUE,
  sort = c("none", "tier>startsec", "startsec>tier")
)
```

Arguments

`t`	Transcript object.
`filterTierNames`	Vector of character strings; names of tiers to be remain in the transcripts. If left unspecified, all tiers will remain in the transcript exported.
`filterSectionStartsec`	Double, start of selection in seconds.
`filterSectionEndsec`	Double, end of selection in seconds.
`timesPreserve`	Logical; Parameter is used if `filterSectionStartsec` it set. If `TRUE` start times will be preserved, if `FALSE` the selection will start from 0.
`sort`	Logical; Annotations will be sorted: 'none' (=no sorting), 'tier>startsec' (=sort first by tier, then by startsec), 'startsec>tier' (=sort first by startsec, then by tier)

Value

Transcript object;

Examples

```
library(act)

# get an example transcript
t1 <- examplecorpus@transcripts[[1]]

# --- Filter by tiers
```

```
# The example transcript contains two tiers that contain four annotations each.
t1@tiers
table(t1@annotations$tierName)

# Filter transcript to only contain annotations of the FIRST tier
t2 <- act::transcripts_filter_single(t1, filterTierNames=t1@tiers$name[1])
t2@tiers
table(t2@annotations$tierName)

# Use act::search_makefilter() first to get the tier names,
# in this case search for tiers with a capital 'I',
# which is the second tier, called 'ISanti'
myfilter <- act::search_makefilter(examplecorpus,
    filterTranscriptNames=t2@name,
    filterTierIncludeRegex="I"
    )
t2 <- act::transcripts_filter_single(t1, filterTierNames=myfilter$tierNames)
t2@tiers
table(t2@annotations$tierName)

# --- Filter by time section
# only set start of section (until the end of the transcript)
t2 <- act::transcripts_filter_single(t1, filterSectionStartsec=6)
cbind(t2@annotations$startsec,t2@annotations$endsec)

# only set end of section (from the beginning of the transcript)
t2 <- act::transcripts_filter_single(t1, filterSectionEndsec=8)
cbind(t2@annotations$startsec,t2@annotations$endsec)

# set start and end of section
t2 <- act::transcripts_filter_single(t1,
  filterSectionStartsec=6,
  filterSectionEndsec=8)
cbind(t2@annotations$startsec,t2@annotations$endsec)

# set start and end of section, start new times from 0
t2 <- act::transcripts_filter_single(t1,
 filterSectionStartsec=6,
 filterSectionEndsec=8)
cbind(t2@annotations$startsec,t2@annotations$endsec)
```

`transcripts_merge` *Merge several transcripts (works with transcript objects directly)*

Description

Merges several transcripts. One transcript is the destination transcript (the transcript that will be updated). The other transcripts are the update transcripts and contain the updates. The update transcripts need to contain a tier in which the update sections are marked with a specific character string.

Usage

```
transcripts_merge(
  destinationTranscript,
  updateTranscripts,
  identifierTier = "update",
  identifierPattern = ".+",
  eraseCompletely = TRUE,
  keepOld = FALSE,
  insertCheckTier = FALSE,
  healCuts = FALSE
)
```

Arguments

destinationTranscript	Transcript object; transcript that serves as destination (and will receive the updates).
updateTranscripts	List of transcript objects; transcript objects that will be inserted into the destination transcripts (entirely or in part).
identifierTier	Character string; regular expression that identifies the tier in which the sections are marked, that will be inserted into destinationTranscript.
identifierPattern	Character string; regular expression that identifies the sections that will be inserted into destinationTranscript.
eraseCompletely	Logical; if TRUE update sections in destination transcript will be erased completely, if FALSE update sections in the destination tier will not be erased completely but only the tiers that are present in the updateTranscripts be erased.
keepOld	Logical; if TRUE annotations that are replaced during the merge are preserved in OLD_ prefixed tiers (e.g. OLD_TeaF). Default is FALSE.
insertCheckTier	Logical; if TRUE a CHECK-update tier is inserted with marker annotations at positions where annotations were truncated. Default is FALSE. Cut annotations are always reported via CLI regardless of this setting.
healCuts	Logical; if TRUE truncated annotations at the boundaries of update regions are automatically joined with adjacent annotations that have the same content. Default is FALSE.

Details

You may chose between the following two options:

- The update sections in the destination transcript will first be erased completely and then the updates will be filled in.

- The update sections in the destination transcript will NOT be erased completely. Rater only the contents of tiers will be erased that are also present in the update tiers. e.g. if your destination transcript contains more tiers than the update transcripts, the contents of those tiers will be preserved in the destination tier during the update.

Value

Transcript object

Examples

```
library(act)

# We need three transcripts  to demonstrate the function \code{transcripts_merge}:
# - the destination transcript
destinationTranscript <-  examplecorpus@transcripts[["update_destination"]]
# - two transcripts that contain updates
updateTranscripts <-   c(examplecorpus@transcripts[["update_update1" ]],
                         examplecorpus@transcripts[["update_update2" ]])

# Run the function
test <- transcripts_merge(destinationTranscript, updateTranscripts)

# Save the transcript to a TextGrid file.
# Set the destination file path
path <- tempfile(pattern = "merge_test", tmpdir = tempdir(),
                 fileext = ".TextGrid")

# It makes more sense, however, to you define a destination folder
# that is easier to access on your computer:
## Not run:
path <- file.path("PATH_TO_AN_EXISTING_FOLDER_ON_YOUR_COMPUTER",
                    paste(t@name, ".TextGrid", sep=""))

## End(Not run)

# Export
act::export_textgrid( t=test, pathOutput=path)
```

---

transcripts_openin_elan

Open transcript in 'ELAN'

Description

The function opens a transcript in ELAN- It can open the an original .eaf file (if it exists) or create an .eaf file. A .psfx file will be created id startSec and endSec are defined.

Usage

```
transcripts_openin_elan(
  t,
  openOriginal = FALSE,
  filePathOut = NULL,
  startSec = NULL,
  endSec = NULL,
  tierName = NULL,
  overwrite = TRUE,
  mediaWriteOffset = FALSE
)
```

Arguments

t	Transcript object.
openOriginal	Logical; if TRUE the function will check if the original annotation file was an .eaf file and if it still exists in the original location. If so, the function will not create a temporary .eaf file but open the original file. Warning: The original .pfsx file (if it exists) will be overwritten.
filePathOut	Character string, optional. If original .eaf is not found, a .eaf file will be created at this location. If NULL a temporary .eaf file will be created.
startSec	Double, Optional, start of selection en ELAN
endSec	Double, Optional, end of selection en ELAN
tierName	Character string, Optional, Name of tier to select in ELAN grid view.
overwrite	Logical; if TRUE (default) an existing .pfsx file will be overwritten. If FALSE an existing .pfsx file is kept.
mediaWriteOffset	Logical; passed to export_eaf when a temporary .eaf is created. If TRUE, clip media get a TIME_ORIGIN offset. Default FALSE.

Details

To make this function work you need to have 'ELAN' installed on your computer and tell the act package where ELAN is located. Therefore you need to set the path to the ELAN executable in the option 'act.path.elan' using `options(act.path.elan='PATHTOYOURELANEXECUTABLE')`.

WARNING: This function will overwrite existing .psfx files.

Credits: Thanks to Han Sloetjes for feedback on the structure of the temporary .pfsx files. He actually made the code work.

Examples

```
library(act)

mysearch <- act::search_new(x=examplecorpus, pattern = "yo")

# You can only use this function if you have installed ELAN on our computer.
```

```
## Not run:
options(act.path.elan='PATHTOYOURELANEXECUTABLE')
act::search_openresult_inelan(x=examplecorpus, s=mysearch, resultid=1, TRUE)

## End(Not run)
```

`transcripts_openin_praat` *Open transcript in 'Praat'*

Description

The function opens a transcript in the 'Praat' TextGrid editor together with its sound and selects a time range. 'Praat' is remote controlled with a 'Praat' script.

Usage

```
transcripts_openin_praat(
  t,
  openOriginal = FALSE,
  filePathOut = NULL,
  startSec = NULL,
  endSec = NULL,
  play = FALSE,
  filterMediaFile = c("(?i).*\\.(aiff|aif|wav)$", "(?i).*\\.mp3$"),
  delay = 0.5,
  zoomStartSec = NULL,
  zoomEndSec = NULL,
  close = FALSE
)
```

Arguments

`t`	Transcript object.
`openOriginal`	Logical; if `TRUE` the original .TextGrid file is opened if the transcript was read from one and it still exists. Otherwise a .TextGrid file is written.
`filePathOut`	Character string, optional. Where the .TextGrid file is written if the original is not used. If `NULL` a temporary file is created.
`startSec`	Double, optional; start of the selection in 'Praat'. If `NULL` the selection starts at 0. If `startSec` equals `endSec`, only the cursor is set (no selection).
`endSec`	Double, optional; end of the selection in 'Praat'. If `NULL` the selection ends at the end of the transcript.
`play`	Logical; if `TRUE` the selection is played.

filterMediaFile
: Vector of character strings; each element is a regular expression. They are checked one after the other; the first existing media file that matches is opened as sound. The default order is uncompressed audio > compressed audio.

delay
: Double; time in seconds to wait before the script is handed to 'Praat'.

zoomStartSec
: Double, optional; start of the range shown in the editor. If NULL (or zoomEndSec is NULL) the editor shows the selection with 0.5 s on either side (2.5 s around a cursor without selection).

zoomEndSec
: Double, optional; end of the range shown in the editor.

close
: Logical; if TRUE the editor is closed after playing (only with play = TRUE).

Details

It can open the original .TextGrid file (if the transcript was read from one and it still exists) or write a .TextGrid file. A TextGrid written by this function replaces an object of the same name that is already open in 'Praat', so the editor always shows the current transcript. An original file is never replaced in 'Praat': the open object may hold edits that are not saved yet.

'Praat' receives the script through its own command line option --send (from 'Praat' 6.1 on): set the path to 'Praat' with 'options(act.path.praat = ...)'. If 'Praat' is not running, it is started (see vignette("praat_remote_control", package = "act")).

Value

Logical; TRUE if the script was handed to 'Praat' (invisibly).

See Also

transcripts_openin_elan, search_openresult_inpraat

Examples

```
library(act)

# You can only use this function if you have installed 'Praat'
# and located it properly in the package options.
## Not run:
act::transcripts_openin_praat(t = examplecorpus@transcripts[[1]], startSec = 1, endSec = 3)

## End(Not run)
```

---

transcripts_rename *Rename transcripts in a corpus*

Description

Rename transcript objects in a corpus object. This function changes both the names of the transcripts in the list x@transcripts and in the @name slot of the transcript. The function ensures that each transcript object preserves a unique name.

Usage

```
transcripts_rename(
  x,
  newTranscriptNames = NULL,
  searchPatterns = NULL,
  searchReplacements = NULL,
  toUpper = FALSE,
  toLower = FALSE,
  trim = FALSE,
  stopNonUnique = TRUE
)
```

Arguments

x	Corpus object
newTranscriptNames	Vector of character strings; new names for the transcripts. If left open, the current names in the corpus object will be taken as basis.
searchPatterns	Character string; Search pattern as regular expression applied to the names of the transcripts.
searchReplacements	Character string; String to replace the hits of the search.
toUpper	Logical; Convert transcript names all to upper case.
toLower	Logical; Convert transcript names all to lower case.
trim	Logical; Remove leading and trailing spaces in names.
stopNonUnique	Logical; If TRUE the function will stop if replacement would lead to non-unique names; If FALSE names will be automatically changed to be unique.

Value

Corpus object

Examples

```
library(act)

# get current names
old.names <- names(examplecorpus@transcripts)

# make vector of names with the same length
new.names <- paste("transcript", 1:length(old.names), sep="")

# rename the transcripts
test <- act::transcripts_rename(examplecorpus, newTranscriptNames=new.names)

# check
names(test@transcripts)
test@transcripts[[1]]@name
```

```
names(test@transcripts)

# convert to lower case
test <- act::transcripts_rename(examplecorpus, toLower=TRUE)
names(test@transcripts)

# search replace
test <- act::transcripts_rename(examplecorpus,
 searchPatterns=c("ARG", "BOL"),
 searchReplacements = c("ARGENTINA", "BOLIVIA")
)
names(test@transcripts)

# search replace ignoring upper and lower case
test <- act::transcripts_rename(examplecorpus,
 searchPatterns=c("(?i)arg", "(?i)bol"),
 searchReplacements = c("ARGENTINA", "BOLIVIA")
)
names(test@transcripts)


# search replace too much
test <- act::transcripts_rename(x=examplecorpus,
 searchPatterns="ARG_I_CHI_Santi",
 searchReplacements = "")
names(test@transcripts)[1]
```

`transcripts_update_fulltexts` *Update full texts*

Description

Creates/updates the full texts of the transcripts in a corpus. The full text may be created in two different ways:

- The contents of a transcription will be joined consecutively based on the time information.
- The contents of each tier will be joined consecutively, and then the next tier will be joined.

Usage

```
transcripts_update_fulltexts(
  x,
  searchMode = c("fulltext", "fulltext.bytier", "fulltext.bytime"),
  transcriptNames = NULL,
  tierNames = NULL,
  forceUpdate = FALSE
)
```

Arguments

x	Corpus object.
searchMode	Deprecated. Both fulltext modes (bytime and bytier) are always created together; this parameter has no effect and is kept only for backward compatibility.
transcriptNames	Vector of character strings; Names of the transcripts you want to update; leave empty if you want to process all transcripts that need an update.
tierNames	Vector of character strings; Names of the tiers to include in the fulltext.
forceUpdate	Logical; If TRUE fulltexts will be created in any case, if FALSE fulltexts will be only be created if there was a modification to the transcript since the last creation of the fulltexts.

Value

Corpus object.

Examples

```
library(act)

examplecorpus <- act::transcripts_update_fulltexts(x=examplecorpus)
```

---

transcripts_update_normalization

Normalize transcriptions

Description

Normalizes the contents of transcriptions in a corpus object using a normalization matrix. Function returns a corpus object with normalized transcription and updates the original corpus object passed as argument to x.

Usage

```
transcripts_update_normalization(
  x,
  transcriptNames = NULL,
  forceUpdate = FALSE
)
```

Arguments

x	Corpus object.
transcriptNames	Vector of character strings; Names of the transcripts for which you want to search media files; leave empty if you want to search media for all transcripts in the corpus object.
forceUpdate	Logical; If TRUE transcripts will be normalized in any case, if FALSE transcripts will be only normalized if there was a modification to the transcript since the last normalization.

Details

If the normalization matrix is missing or empty (0 rows), a warning is issued and the corpus object is returned unchanged.

Examples

```
library(act)

examplecorpus <- act::transcripts_update_normalization(x=examplecorpus)
```

Index

∗ **datasets**
examplecorpus, 28

act, 5
annotations_add, 9
annotations_all, 11
annotations_delete, 11
annotations_delete_empty, 12
annotations_fill_tier_section, 14
annotations_insert_from_search_to_tier, 15
annotations_matrix, 17
annotations_replace_copy, 18

corpus-class, 20
corpus_export, 21, *33*, *37*, *39*, *42*, *44*, *49*, *70*, *71*
corpus_import, 23, *26*, *28*, *29*, *103*, *105–108*, *110*
corpus_merge, 25
corpus_new, *24*, 26, *29*, *103*, *105–108*, *110*

examplecorpus, *24*, 28, *28*, *30*
examplecorpus_media, *29*, 30
export_create_pfsx, 30
export_docx, *22*, 31, *34*, *36*, *37*, *39*, *40*, *42*, *44*, *123*, *127*, *135*
export_eaf, *22*, 33, *33*, *36*, *37*, *39*, *40*, *42*, *44*, *181*
export_edl, 35
export_exb, *22*, *33*, *34*, *36*, 37, *39*, *40*, *42*, *44*
export_printtranscript *(*export_txt*)*, 42
export_rpraat, *22*, *33*, *34*, *36*, *37*, 38, *40*, *42*, *44*, *107*
export_srt, *22*, *33*, *34*, *36*, *37*, 39, *39*, *42*, *44*
export_textgrid, *22*, *33*, *34*, *36*, *37*, *39*, *40*, 41, *44*
export_txt, *22*, *34*, *36*, *37*, *39*, *40*, 42, *42*, *44*

helper_annotationfile_read, 45, *52*
helper_annotations_columns_order, 46
helper_audio_filter_build, 46
helper_audio_filter_parts, *46*, 47, *56*, *57*, *133*
helper_corpus_export_cured, 49
helper_cutlist_lines, 49, *59*, *133*
helper_cutlist_save, *49*, *50*, 51
helper_encoding_file_detect, *45*, *46*, 52
helper_encoding_name_normalize, 53, *53*, *54*
helper_encoding_names_equivalent, *46*, 53, *53*
helper_ffmpeg_args, *49*, 54, *57–59*, *61*, *63*, *132*, *133*
helper_ffmpeg_args_audio, *48*, *49*, 56, *133*
helper_ffmpeg_args_clip, *48*, *49*, 57, *57*, *61*, *62*, *132*, *133*
helper_ffmpeg_check, 60, *62*, *64*
helper_ffmpeg_input_flags, 61
helper_ffmpeg_keyframe, *59*, 61, *132*
helper_ffmpeg_path, *55*, *60*, 62, *63*, *64*
helper_ffmpeg_run, *55*, *57*, *59*, 63
helper_ffprobe_path, *60*, *62*, 64
helper_file_app_available, 64
helper_file_date_key_make, 65, *66*, *67*
helper_file_hash_compute, *46*, 66, *69*, *166*
helper_file_newest_select, *65*, 66
helper_file_open_in, 67
helper_file_reveal, 68
helper_file_state, *46*, 68, *69*, *166*
helper_file_state_changed, *68*, 69, *166*
helper_history_changed, 70, *70*, *71*
helper_history_mark_export, 70, *70*
helper_layout_anchors, 71
helper_layout_bracket_pairs, 72
helper_layout_render, 73
helper_layout_symbol_matches, 75
helper_media_audio_track_exists, 76
helper_media_type, 77

helper_media_video_timing_read, 77
helper_metadata_exif_argv, *50*, 78
helper_metadata_exif_argv(), *79*
helper_metadata_exif_write, 78
helper_metadata_exif_write(), *78*, *80*, *120*
helper_metadata_ffmpeg_args, 79
helper_metadata_ffmpeg_args(), *79–81*, *120*
helper_metadata_ffmpeg_argv, *58*, 80
helper_metadata_ffmpeg_argv(), *81*
helper_metadata_probe_fps, 81
helper_program_paths_detect, 81
helper_progress_callback_get, 82
helper_progress_callback_set, 82, *82*
helper_progress_eta_format, 83
helper_progress_set, *82*, 84
helper_progress_tick, *82*, 84
helper_style_acronym, 85
helper_style_appearance, 86
helper_style_docx, 86
helper_style_list, 87
helper_style_read, 87
helper_style_video, 88
helper_text_graphemes_split, 89
helper_tiers_filter_create, 89
helper_tiers_merge_tables, 90, *91–93*
helper_tiers_new_table, 91, *93*, *161–164*
helper_tiers_sort_table, *91*, 92, *92*, *161–164*
helper_time_format, 94, *95*
helper_time_formats_list, 95
helper_transcript_names_get, 96
helper_transcript_names_make, 97
helper_transcript_names_set, 98
helper_transcript_patch_apply, 99, *100*, *101*
helper_transcript_patch_make, *99*, 100, *102*
helper_transcripts_compare, 101, *101*

import, 102, *105–108*, *110*
import_eaf, *103*, 104, *106–108*, *110*
import_exb, *103*, 105, *105*, *107*, *108*, *110*
import_rpraat, *39*, *103*, *105*, 106, *106*, *108*, *110*
import_srt, *103*, 108
import_textgrid, *103*, *105–107*, 109
info, 111, *112*
info_summarized, *111*, 112

layers_convert_column_to_tier, 113

matrix_load, 114
matrix_load(), *18*
matrix_save, 115
matrix_save(), *18*
media_assign, 116, *118*, *119*, *121*, *122*
media_build, 117, *119*
media_delete, *18*, *117*, 118, *121*
media_metadata_fill, 119
media_metadata_read, *55*, 119, *119*
media_metadata_read(), *79–81*
media_path_to_existing_file, *18*, *117*, *118*, 120, *122*
media_select, *117*, *118*, 121
merge_docx, 123

options_delete, 124
options_reset, 124
options_show, 124

search-class, 126
search_concordance, 128
search_cuts, 129
search_cuts_media, 130, *131*, *133*, *153*
search_cuts_printtranscript, 134
search_cuts_srt, 136
search_makefilter, 138, *141*, *150*, *155*
search_new, *139*, 140, *150*, *153*, *155*
search_openresult_inelan, 142
search_openresult_inpraat, 143, *183*
search_openresult_inquicktime, 145
search_playresults_inquicktime, 146
search_results_export, 147
search_results_import, 149
search_run, *139*, *141*, 150, *155*
search_searchandopen_inpraat, 151
search_stills, 152
search_sub, *139*, *141*, *150*, 154
search_transcript_content, 155
search_transcript_fulltext, 156

tiers_add, 157, *161–164*
tiers_all, 159
tiers_convert, *91*, *92*, *158*, 160, *162–164*
tiers_delete, *158*, *161*, 162, *164*
tiers_rename, *91*, *92*, *158*, *161*, *162*, 163, *163*, *164*

tiers_sort, *91–93*, *158*, *161–163*, 164
transcript-class, 165
transcripts_add, 167
transcripts_cure, 168, *171*
transcripts_cure_single, *27*, *49*, 170, *170*
transcripts_delete, 173
transcripts_filter, 174
transcripts_filter_remove_single, 175
transcripts_filter_single, 176
transcripts_merge, *91*, 178
transcripts_openin_elan, 180, *183*
transcripts_openin_praat, 182
transcripts_rename, 183
transcripts_update_fulltexts, 185
transcripts_update_normalization, 186