Package 'act'

September 26, 2021

Type Package

Title Aligned Corpus Toolkit

Version 1.1.9

Date 2021-09-25

Maintainer Oliver Ehmer <oliver.ehmer@romanistik.uni-freiburg.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

RoxygenNote 7.1.2

Imports methods, stringr, stringi, tools, textutils, utils, progress, XML, xml2, openxlsx

Suggests knitr, rmarkdown

VignetteBuilder knitr

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

NeedsCompilation no

Author Oliver Ehmer [aut, cre]

Repository CRAN

Date/Publication 2021-09-26 18:00:02 UTC

R topics documented:

`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

...

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
# The act package comes with some example data.
# The data 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)

# The example files that come with the package do only contain annotation files.
# Media files are not included.
# The following lines will download the data and create a new folder called
# 'examplecorpus' in the folder 'path'.
# You will find the data there.
## Not run:
path <- "EXISTING_FOLDER_ON_YOUR_COMPUTER"
sourceurl <-
"http://www.romanistik.uni-freiburg.de/ehmer/files/digitalhumanities/act_examplecorpus.zip"
temp <- tempfile()
download.file(sourceurl, temp)
unzip(zipfile=temp, exdir=path)

## 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_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,
  trimBeforeCheck = FALSE,
  filterTranscriptNames = NULL,
  filterTierNames = NULL
)
```

Arguments

`x`	Corpus object.
`trimBeforeCheck`	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_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, path_replacementMatrixCSV, filterTranscriptNames = NULL)
```

Arguments

`x`	Corpus object.
`path_replacementMatrixCSV`	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_getPathToExistingFile

Examples

```
library(act)

# An example replacement matrix comes with the package.
# It replaces most of the GAT conventions.
path <- system.file("extdata", "normalization", "normalizationMatrix.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, path_replacementMatrixCSV=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,
  addDestTierIfMissing = 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).
`addDestTierIfMissing`	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 examplecorpus:
printtranscript <- act::export_printtranscript(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 corprus object test:
printtranscript <- act::export_printtranscript(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,
addMissingTiers=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_printtranscript(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_printtranscript(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 `@history` and . Other slots are settings and are used when performing functions on the corpus oibject. To change the normalization matrix use `x@normalization.matrix <-act::matrix_load(path="...")`

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.

`import.skip.double.files` Logical; if `TRUE` files with the same names will be skipped (only one of them will be loaded), if `FALSE` transcripts will be renamed to make the names unique.

`import.modify.transcript.names` 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.

`normalization.matrix` Data.frame; Replacement matrix used for normalizing the annotations.

`history` List; History of modifications made by any of the package functions to the corpus.

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,
  outputFolder,
  filterTranscriptNames = NULL,
  filterTierNames = NULL,
  formats = c("eaf", "exb", "srt", "textgrid", "printtranscript"),
  createMediaLinks = TRUE,
  createOutputfolder = TRUE,
  l = NULL
)
```

Arguments

`x`	Corpus object.
`outputFolder`	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', 'srt', 'textgrid', 'printtranscript'. If left unspecified, all supported formats will be exported.

createMediaLinks — Logical; if TRUE media links will be created (affects only 'eaf' and 'exb' files).

createOutputfolder — Logical; if TRUE the outputfolder will be created recursively in case that it does not exist.

l — Layout object. layout of print transcripts (affects only 'printtranscript' files).

See Also

export_eaf, export_textgrid, import_textgrid

Examples

```
library(act)

# Set destination folder
outputFolder <- tempdir()

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

## End(Not run)

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

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

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


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

```
# Exports in 'ELAN' .eaf and Praat' .TextGrid format
act::corpus_export(x=examplecorpus,
                                outputFolder=outputFolder,
                                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,
  filterFilesInclude = "",
  createFullText = TRUE,
  assignMedia = TRUE
)
```

Arguments

`x`	Corpus object.
`filterFilesInclude`	Character string; Regular expression which files should be loaded.
`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

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()$act.fileformats.audio` and `options()$act.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_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,
  pathsMediaFiles = NULL,
  name = "New Corpus",
  importFiles = TRUE,
  skipDoubleFiles = TRUE,
  createFullText = TRUE,
  assignMedia = TRUE,
  pathNormalizationMatrix = NULL,
  namesSearchPatterns = character(),
  namesSearchReplacements = character(),
```

```
  namesToUpperCase = FALSE,
  namesToLowerCase = FALSE,
  namesTrim = TRUE,
  namesDefaultForEmptyNames = "no_name"
)
```

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.
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.
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.
namesToUpperCase	Logical; Convert transcript names all to upper case.
namesToLowerCase	Logical; Convert transcript names all to lower case.
namesTrim	Logical; Remove leading and trailing spaces in names.
namesDefaultForEmptyNames	Character string; Default value for empty transcript names (e.g., resulting from search-replace operations)

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()$act.fileformats.audio` and `options()$act.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

You can download the corresponding media files from www.oliverehmer.de in the section "Digital Humanities". Alternatively you can use the download commands in the example section.

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/lingva overview.xml

See Also

, corpus_new, corpus_import

Examples

```
library(act)

# Summary of the data in the corpus
examplecorpus

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

## Not run:
# Download example corpus with media files
destinationpath <- "/EXISTING_FOLDERON_YOUR_COMPUTER/examplecorpus"
temp <- tempfile()
download.file(options()$act.examplecorpusURL, temp)
unzip(zipfile=temp, exdir=destinationpath)

## End(Not run)
```

export_eaf *Export a transcript object to a '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,
  outputPath = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  createMediaLinks = TRUE
)
```

Arguments

`t`	Transcript object; transcript to be exported.
`outputPath`	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.

Details

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

Value

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

See Also

`corpus_export`, `export_exb`, `export_printtranscript`, `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 = ".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, outputPath=path)

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

---

export_exb *Export a transcript object to a '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,
  outputPath = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  createMediaLinks = TRUE
)
```

Arguments

`t`	Transcript object; transcript to be exported.
`outputPath`	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 `outputPath`. If `outputPath` is left empty, the function will return the contents of the .exb itself.

Value

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

See Also

`corpus_export`, `export_eaf`, `export_printtranscript`, `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, outputPath=path)

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

`export_printtranscript` *Export a transcript object to a print transcript*

Description

If you want to modify the layout of the print transcripts, create a new layout object with `mylayout <-methods::new("layout")`, modify the settings and pass it as argument `l`. In the layout object you may also set additional filters to include/exclude tiers matching regular expressions.

Usage

```
export_printtranscript(
  t,
  l = NULL,
  outputPath = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  insert_arrow_annotationID = "",
  header_heading = "",
  header_firstinfo = "",
  collapse = TRUE
)
```

Arguments

`t`	Transcript object.
`l`	Layout object.
`outputPath`	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.
`insert_arrow_annotationID`	Integer; ID of the annotation in front of which the arrow will be placed.
`header_heading`	Character string; text that will be used as heading.
`header_firstinfo`	Character string; text that will used as first information in the header.
`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.

Value

Character string; transcript as text.

See Also

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

Examples

```
library(act)

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

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

# Display on screen
cat(printtranscript)
```

---

export_rpraat *Export a transcript object to a '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_printtranscript`, `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 a transcript object to a .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,
  outputPath = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  speaker.show = TRUE,
  speaker.width = 3,
  speaker.ending = ":"
)
```

Arguments

t	Transcript object; transcript to be saved.
outputPath	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.
speaker.show	Logical; if TRUE name of speaker will be shown before the content of the annotation.
speaker.width	Integer; width of speaker abbreviation, -1 for full name without shortening.
speaker.ending	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 outputPath. If outputPath is left empty, the function will return the contents of the .srt itself.

Value

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

See Also

corpus_export, export_eaf, export_exb, export_printtranscript, 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, outputPath=path)
```

`export_textgrid` *Export a transcript object to a '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,
  outputPath = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL
)
```

Arguments

`t`	Transcript object; transcript to be saved.
`outputPath`	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 `outputPath`. If `outputPath` is left empty, the function will return the contents of the .TextGrid itself.

Value

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

See Also

`corpus_export`, `export_eaf`, `export_exb`, `export_printtranscript`, `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, outputPath=path)
```

---

helper_format_time *Formats time as HH:MM:SS,mmm*

Description

Formats time as HH:MM:SS,mmm

Usage

```
helper_format_time(
  t,
  digits = 1,
  addHrsMinSec = FALSE,
  addTimeInSeconds = FALSE
)
```

Arguments

t	Double; time in seconds.
digits	Integer; number of digits.
addHrsMinSec	Logical; if TRUE 'hrs' 'min' 'sec' will be used instead of ':'.
addTimeInSeconds	Logical; if TRUE time value in seconds will be shown, too.

Value

Character string.

Examples

```
library(act)


helper_format_time(12734.2322345)
helper_format_time(2734.2322345)
helper_format_time(34.2322345)
helper_format_time(0.2322345)

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

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

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

helper_format_time(12734.2322345, addHrsMinSec=TRUE, addTimeInSeconds=TRUE)
helper_format_time(2734.2322345, addHrsMinSec=TRUE, addTimeInSeconds=TRUE)
helper_format_time(34.2322345, addHrsMinSec=TRUE, addTimeInSeconds=TRUE)
helper_format_time(0.2322345, addHrsMinSec=TRUE, addTimeInSeconds=TRUE)

helper_format_time(12734.2322345, addHrsMinSec=TRUE, digits=3, addTimeInSeconds=TRUE)
helper_format_time(2734.2322345, addHrsMinSec=TRUE, digits=3, addTimeInSeconds=TRUE)
helper_format_time(34.2322345, addHrsMinSec=TRUE, digits=3, addTimeInSeconds=TRUE)
helper_format_time(0.2322345, addHrsMinSec=TRUE, digits=3, addTimeInSeconds=TRUE)
```

---

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,
  addMissingTiers = TRUE,
  deleteTiersThatAreNotInTheSortVector = FALSE
)
```

Arguments

`tierTable`	Data frame; tiers as specified and necessary in `@tiers` of a transcript object.
`sortVector`	Vector of character strings; regular expressions to match the tier names. The order within the vector presents the new order of the tiers. Use "*" (=two backslashes and a star) to indicate where tiers that are not present in the sort vector but in the transcript should be inserted.
`addMissingTiers`	Logical; if `TRUE` all tiers that are given in 'the 'sortVector' but are missing in 'tierTable' will be added.
`deleteTiersThatAreNotInTheSortVector`	Logical; if `TRUE` tiers that are not matched by the regular expressions in 'sortVector' will be deleted. Otherwise the 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 regular expression strings. 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 a element '*' in 'sortVector' to define the position where they should be placed..

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 'deleteTiersThatAreNotInTheSortVector=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,
deleteTiersThatAreNotInTheSortVector=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_transcriptNames_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_transcriptNames_get(x)
```

Arguments

x	Corpus object

Value

List

Examples

```
library(act)

act::helper_transcriptNames_get(examplecorpus)
```

`helper_transcriptNames_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_transcriptNames_make(
  transcriptNames,
  searchPatterns = character(),
  searchReplacements = character(),
  toUpperCase = FALSE,
  toLowerCase = FALSE,
  trim = FALSE,
  defaultForEmptyNames = "no_name"
)
```

Arguments

`transcriptNames`	Vector of character strings; Names of the transcripts to validate.
`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.
`toUpperCase`	Logical; Convert transcript names all to upper case.
`toLowerCase`	Logical; Convert transcript names all to lower case.
`trim`	Logical; Remove leading and trailing spaces in names.
`defaultForEmptyNames`	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_transcriptNames_make(transcriptNames)
```

---

helper_transcriptNames_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_transcriptNames_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_transcriptNames_get(examplecorpus)

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

# create an error: empty name
```

```
## Not run:
names.test <- names.old
names.test[2] <- " "
test <-  act::helper_transcriptNames_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_transcriptNames_set(examplecorpus, names.test)

## End(Not run)
```

`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 and 'Praat' .TextGrid 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.

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

Value

Transcript object.

See Also

`corpus_import`, `corpus_new`, `import_eaf`, `import_exb`, `import_rpraat`, `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()`. 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. 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, import, import_textgrid, import_eaf

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)
```

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.

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()`. 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]])
```

layout-class *Layout object, defining the layout of print transcripts*

Description

You can create an new layout object with `methods::new("layout")`. This will give you a new layout object with the default settings uses by act. If you want to modify the layout of the print transcripts, create a new layout object with `mylayout <-methods::new("layout")`, modify the values in the `@slots` and pass it as argument `l` to the respective functions.

Slots

`name` Character string; Name of the layout.

`filter.tier.includeRegEx` Character string; as regular expression, tiers matching the expression will be included in the print transcript.

`filter.tier.excludeRegEx` Character string; as regular expression, tiers matching the expression will be excluded from the print transcript.

`transcript.width` Integer; width of transcript, -1 for no line wrapping.

`speaker.width` Integer; width of speaker abbreviation, -1 for full name without shortening.

`speaker.ending` Character string; string that is added at the end of the speaker name.

`spacesbefore` Integer; number of spaces inserted before line number.

`additionalline1.insert` Logical; if `TRUE` an additional dummy line will be inserted after each annotation line, the text is defined in `.additionalline1.text`.

`additionalline1.text` Character string; Content of additional dummy line 1.

`additionalline1.indent` Logical; if `TRUE` the content of the dummy line 1 will be indented to begin where the content of the annotations start.

`additionalline2.insert` Logical; if `TRUE` an additional dummy line will be inserted after each annotation line, the text is defined in `.additionalline2.text`.

`additionalline2.text` Character string; Content of additional dummy line 2.

`additionalline2.indent` Logical; if `TRUE` the content of the dummy line 2 will be indented to begin where the content of the annotations start.

`brackets.tryToAlign` Logical; if `TRUE` act will try to align brackets [] for parallel speaking (Attention: experimental function; results may not satisfy).

`pauseTierRegEx` Character string; regular expression to identify pause tier for auto formatting pauses.

`header.insert` Logical; if `TRUE` a transcript header is inserted.

`header.heading.fromColumnName` Character string; is only used when transcripts are made based on a search results; defines from which column of a search results table the heading is taken (if `object@.header.insert==TRUE`)

`header.firstInfo.fromColumnName` Character string; is only used when transcripts are made based on a search results; defines from which column of a search results table the first info is taken (if `object@.header.insert==TRUE`)

`arrow.insert` Logical; is only used when transcripts are made based on a search results; if `TRUE` an arrow will be inserted, highlighting the transcript line containing the search hit.

`arrow.shape` Character string; shape of the arrow.

`matrix_load` *Load replacement matrix*

Description

This function is only for checking how the normalization matrix will be loaded internally.

Usage

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

Arguments

`path`	Character string; path to the replacement matrix (a CSV file). If argument is left open, the default replacement matrix of the package will be returned.
`myFileEncoding`	Character string; encoding of the file.

Value

Data.frame

Examples

```
library(act)

# An example replacement matrix comes with the package.
path <- system.file("extdata", "normalization", "normalizationMatrix.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, myFileEncoding = "UTF-8")
```

Arguments

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

Value

nothing

Examples

```
library(act)

# An example replacement matrix comes with the package.
path <- system.file("extdata", "normalization", "normalizationMatrix.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,
  searchInSubfolders = TRUE,
  filterFile = "",
  transcriptNames = NULL,
  deleteExistingMedia = TRUE,
  onlyUniqueFiles = TRUE
)
```

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).
searchInSubfolders	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.
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.

Details

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

Value

Corpus object.

See Also

media_delete, media_getPathToExistingFile

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_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_getPathToExistingFile

Examples

```
library(act)

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

`media_getPathToExistingFile` *Gets the path of a media file for a transcript*

Description

Gets the path of a media file for a transcript

Usage

```
media_getPathToExistingFile(
  t,
  filterMediaFile = c(".*\\.(mp4|mov)", ".*\\.(aiff|aif|wav)", ".*\\.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.path

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

`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()
```

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.

The package uses the following options.

Program * `act.excamplecorpusURL` character strings; where to download example media files. * `act.updateX` Logical; If `TRUE` the original corpus object 'x' passed passed to the search functions `search_new` and `search_run` will also be updated, in case that during the search fulltexts are created or the normalization is performed. * `act.showprogress` logical; if `TRUE` a progress bar will be shown during (possibly) time consuming operations.

Paths * `act.path.praat` Character string; path to the 'Praat' executable on your computer. Only necessary if you use the functions to remote control Praat using Praat scripts. * `act.path.sendpraat` Character string; path to the 'sendpraat' executable on your computer. Only necessary if you use the functions to remote control Praat using Praat scripts. * `act.path.elan` Character string; path to the 'ELAN' executable on your computer. Only necessary if you want to open search results in ELAN.

File formats * `act.fileformats.video` Vector of character strings; Suffixes of video files that will be identified; default is 'c("mp4", "mov")'. * `act.fileformats.audio` Vector of character strings; Suffixes of audio files that will be identified; default is 'c("wav", "aif", "mp3")'.

FFMPEG commands and options * `act.ffmpeg.command` Character string; 'FFmpeg' command that is used for cutting video files. * `act.ffmpeg.command.fastVideoPostioning` Character string; 'FFmpeg' command that is used for cutting video files using the 'FFmpeg' option 'fast video positioning'. This is considerably faster when working with long video files. * `act.ffmpeg.command.audio` Character string; 'FFmpeg' command that is used for cutting/generating uncompressed audio files. * `act.ffmpeg.command.UsefastVideoPostioning` Logical; if `TRUE` the 'FFmpeg' option using fast video positioning (ant the respective commands as defined in the other options) will be used. * `act.ffmpeg.exportchannels.fromColumnName` Character string; Name of the column in the data frame `s@results` from information, which audio channel to export, will be taken.

Import annotation files * `act.import.readEmptyIntervals` Logical; if `TRUE` empty intervals in you annotation files will be read, if `FALSE` empty intervals will be skipped. * `act.import.scanSubfolders` Logical; if `TRUE` sub folders will also be scanned for annotation files; if `FALSE` only the main level of the folders specified in `paths.annotation.files` of your corpus object will be scanned. *

act.import.storeFileContentInTranscript if TRUE the contents of the original annotation file will be stored in transcript@file.content. Set to FALSE if you want to keep your corpus object small.

Export * act.export.foldergrouping1.fromColumnName Character string; Name of sub folders that will be created in the folder of the search result, level 1. * act.export.foldergrouping2.fromColumnName Character string; Name of sub folders that will be created in the folder of the search result, level 2. * act.export.filename.fromColumnName Character string; Name of the column from which the file names for exported files will be taken.

Miscellaneous * act.separator_between_intervals Character; Single character that is used for separating intervals when creating the full text. * act.separator_between_tiers Character; Single character that is used for separating tiers when creating the full text. * act.separator_between_words Character string; regular expression with alternatives that count as separators between words. Used for preparing the concordance. * act.wordCountRegEx Character string; regular expression that is used to count words.

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.

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

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 substitles 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.

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,
  l = NULL,
  outputFolder = 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.
l	Layout object.
outputFolder	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.

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,
  cutSpanBeforesec = NULL,
  cutSpanAftersec = NULL,
  outputFolder = NULL,
  filterMediaInclude = "",
  fastVideoPostioning = TRUE,
  videoCodecCopy = FALSE,
  audioCutsAsMP3 = FALSE,
  Panning = NULL
)
```

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.path` . You can use the respective media functions. .
`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.

`outputFolder`
: 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.path`.

`fastVideoPostioning`
: Logical; If `TRUE` the FFmpeg command will be using the parameter fast video positioning as specified in `options()$act.ffmpeg.command.fastVideoPostioning`.

`videoCodecCopy`
: Logical; if `TRUE` FFMPEG will use the option *codec copy* for videos.

`audioCutsAsMP3`
: Logical; If `TRUE` audio cuts will be exported as '.mp3' files, using `options()$act.ffmpeg.command.aud`

`Panning`
: 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.exportchannels.fromColumnName' .

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.path`. 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 format
The output format is predefined by in the options:

- `act.ffmpeg.command` defines the basic FFmpeg command
- `act.ffmpeg.command.fastVideoPostioning` defines the FFmpeg command to be used with large video files.

The default is to generate mp4 video cuts. You can also use the following commands to change the output format:

MP4 video cuts with original video quality:

- `options(act.ffmpeg.command = 'ffmpeg -i "INFILEPATH" -ss TIMESTART -t TIMEDURATION OPTIONS -y "OUTFILEPATH.mp4" -hide_banner')`

- options(act.ffmpeg.command.fastVideoPostioning = 'ffmpeg -ss TIMESTARTMINUS10SECONDS -i "INFILEPATH" -ss 10.000 -t TIMEDURATION OPTIONS -y "OUTFILEPATH.mp4" -hide_banner')

MP4 video cuts with reduced video quality:

- options(act.ffmpeg.command = 'ffmpeg -i "INFILEPATH" -ss TIMESTART -t TIMEDURATION OPTIONS -vf scale=1920:-1 -b:v 1M -b:a 192k -y "OUTFILEPATH.mp4" -hide_banner')
- options(act.ffmpeg.command.fastVideoPostioning = 'ffmpeg -ss TIMESTARTMINUS10SECONDS -i "INFILEPATH" -ss 10.000 -t TIMEDURATION OPTIONS -vf scale=1920:-1 -b:v 6M -b:a 192k -y "OUTFILEPATH.mp4" -hide_banner')

WAV audio cuts:

- options(act.ffmpeg.command = 'ffmpeg -i "INFILEPATH" -ss TIMESTART -t TIMEDURATION OPTIONS -y "OUTFILEPATH.wav" -hide_banner')
- options(act.ffmpeg.command = 'ffmpeg -i "INFILEPATH" -ss TIMESTART -t TIMEDURATION OPTIONS -y "OUTFILEPATH.mp3" -hide_banner')

Advanced options
You can adjust the FFmpeg commands according to your needs. The following options define the FFmpeg command that will be used by the package. The command needs to contain place holders which will be replaced by the actual values in the package. If you want to define your own ffmpeg command, please make sure to use the following placeholders:

- INFILEPATH path to the input media file.
- OUTFILEPATH path where the output media file will be saved
- OPTIONS FFmpeg options that will be applied additionally, in particular fast video positioning.
- TIMESTART time in seconds where to begin the cutting
- TIMESTARTMINUS10SECONDS time in seconds where to begin the cutting, in case that fast video positioning is being used.
- TIMEDURATION duration of cuts.

Value

Search object; cut lists will be stored in s@cuts.cutlist.mac and s@cuts.cutlist.win.

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 'outputFolder' 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,
  cutSpanBeforesec = NULL,
  cutSpanAftersec = NULL,
  l = NULL,
  outputFolder = 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.
l	Layout object.
outputFolder	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.

Details

If you want to modify the layout of the print transcripts, create a new layout object with `mylayout <-methods::new("layout")`, modify the settings and pass it as argument `l`.

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])
```

`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,
  outputFolder = NULL,
  speaker.show = TRUE,
  speaker.width = 3,
  speaker.ending = ":"
)
```

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.
`outputFolder`	Character string; if parameter is not set, the srt subtitles will only be inserted in `s@results`; if the path to a existing folder is given transcripts will be saved in '.srt' format.
`speaker.show`	Logical; if `TRUE` name of speaker will be shown before the content of the annotation.
`speaker.width`	Integer; width of speaker abbreviation, -1 for full name without shortening.
`speaker.ending`	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. If you want to modify the layout of the print transcripts, create a new layout object with `mylayout <-methods::new("layout")`, modify the settings and pass it as argument `l`.

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$transcript.names

# Search all transcripts that don't have "ARG" in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTranscriptExcludeRegEx="ARG")
myfilter$transcript.names

# Search all tiers that have an "A" or an "a" in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTierIncludeRegEx="(?i)A")
myfilter$tier.names

# Search all tiers that have a capital "A" in their name
myfilter <- act::search_makefilter(x=examplecorpus, filterTierIncludeRegEx="A")
myfilter$tier.names

# In which transcripts do these tiers occur?
myfilter$transcript.names

# Let's check the first of the transcripts, if this is really the case...
examplecorpus@transcripts[[myfilter$transcript.names[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",
  filterTranscriptNames = NULL,
  filterTranscriptIncludeRegEx = NULL,
  filterTranscriptExcludeRegEx = NULL,
  filterTierNames = NULL,
  filterTierIncludeRegEx = NULL,
  filterTierExcludeRegEx = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  concordanceMake = TRUE,
  concordanceWidth = NULL,
  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; prefix for the name of the consecutively numbered search results.
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.
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.
concordanceWidth	Integer; number of characters to the left and right of the search hit in the concordance , the default is `120`.
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,
  resultNr,
  openOriginalEafFileIfAvailable = FALSE
)
```

Arguments

`x`	Corpus object.
`s`	Search object.
`resultNr`	Integer; Number of the search result (row in the data frame `s@results`) to be opened.
`openOriginalEafFileIfAvailable`	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.

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, resultNr=1, TRUE)

## End(Not run)
```

search_openresult_inpraat *Open a search result in 'Praat'*

Description

The function remote controls 'Praat' by using 'sendpraat' and a 'Praat' script. It opens a search result in the 'Praat' TextGrid Editor.

Usage

```
search_openresult_inpraat(
  x,
  s,
  resultNr,
  play = TRUE,
  closeAfterPlaying = FALSE,
  filterMediaFile = c(".*\\.(aiff|aif|wav)", ".*\\.mp3"),
  delayBeforeOpen = 0.5
)
```

Arguments

x	Corpus object.
s	Search object.
resultNr	Integer; Number of the search result (row in the data frame s@results) to be played.
play	Logical; If TRUE selection will be played.
closeAfterPlaying	Logical; If TRUE TextGrid editor will be closed after playing (Currently non functional!)
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.
delayBeforeOpen	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

To make this function work you need to do two things first: - Install 'sendpraat' on your computer. To do so follow the instructions in the vignette 'installation-sendpraat'. Show the vignette with `vignette("installation-sendpraat")`. - Set the path to the 'sendpraat' executable correctly by using 'options(act.path.sendpraat = ...)'.

Examples

```
library(act)

mysearch <- act::search_new(x=examplecorpus, pattern = "pero")

# You can only use this functions if you have installed and
# located the 'sendpraat' executable properly in the package options.
## Not run:
act::search_openresult_inpraat(x=examplecorpus, s=mysearch, resultNr=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,
  resultNr,
  play = TRUE,
  closeAfterPlaying = FALSE,
  bringQuicktimeToFront = TRUE,
  filterFile = c(".*\\.(mp4|mov)", ".*\\.(aiff|aif|wav)", ".*\\.mp3")
)
```

Arguments

`x`	Corpus object.
`s`	Search object.
`resultNr`	Integer; Number of the search result (row in the data frame `s@results`) to be played.
`play`	Logical; If `TRUE` selection will be played.
`closeAfterPlaying`	Logical; if `TRUE` the Quicktime player will be closed after playing the cuts.
`bringQuicktimeToFront`	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,
resultNr = 1,
play=TRUE,
closeAfterPlaying=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,
resultNr = i,
play=TRUE,
closeAfterPlaying=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, bringQuicktimeToFront = FALSE)
```

Arguments

x	Corpus object.
s	Search object.
bringQuicktimeToFront	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,
  sheetNameXLSX = "data",
  saveAsCSV = FALSE,
  encodingCSV = "UTF-8",
  separatorCSV = ",",
  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.
`sheetNameXLSX`	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.
`encodingCSV`	Character string; text encoding for CSV files.
`separatorCSV`	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,
  sheetNameXLSX = "data",
  encodingCSV = "UTF-8",
  separatorCSV = ","
)
```

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.

sheetNameXLSX	Character string, set the name of the excel sheet containing the data.
encodingCSV	Character string; text encoding in the case of CVS files.
separatorCSV	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$transcript.name, mysearch@results$tier.name, 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$transcript.name, mysearch@results$tier.name, mysearch@results$hit)
```

---

search_searchandopen_inpraat

Search corpus and open first result in Praat

Description

The function remote controls 'Praat' by using 'sendpraat' and 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

To make this function work you need to do two things first: - Install 'sendpraat' on your computer. To do so follow the instructions in the vignette 'installation-sendpraat'. Show the vignette with `vignette("installation-sendpraat")`. - Set the path to the 'sendpraat' executable correctly by using 'options(act.path.sendpraat = ...)'.

Examples

```
library(act)

# You can only use this functions if you have installed
# and located the 'sendpraat' executable properly in the package options.
## Not run:
act::search_searchandopen_inpraat(x=examplecorpus, "pero")

## End(Not run)
```

---

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",
  deleteLinesWithNoResults = FALSE,
  excludeHitsWithinSameTier = TRUE
)
```

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
deleteLinesWithNoResults	Logical; if `TRUE` search results will be deleted for which the sub search does not give any results
excludeHitsWithinSameTier	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.

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.

@example inst/examples/search_transcript_content.R

`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.

@example inst/examples/search_transcript_fulltext.R

`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"),
  absolutePosition = 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.
absolutePosition	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 'absolutePosition' 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. 'absolutePosition=1') or in relation to a existing tier (e.g. destinationTier='speaker1'). To insert a tier at the end, leave 'absolutePosition' 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
x@history[length(x@history)]
#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",
absolutePosition=2)
#check results
x@history[length(x@history)]
#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
x@history[length(x@history)]
#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
x@history[length(x@history)]
#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 and returns 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)
x@history[length(x@history)]

# tiers 'A' and 'B' do not occur anymore
act::info(x)$tiers$tier.name
```

`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 regular expressions defined in 'sortVector'. 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,
  addMissingTiers = FALSE,
  deleteTiersThatAreNotInTheSortVector = FALSE
)
```

Arguments

`x`	Corpus object.
`sortVector`	Vector of character strings; regular expressions to match the tier names. The order within the vector presents the new order of the tiers. Use "*" (=two backslashes and a star) to indicate where tiers that are not present in the sort vector but in the transcript should be inserted.
`filterTranscriptNames`	Vector of character strings; names of the transcripts to be included.
`addMissingTiers`	Logical; if `TRUE` all tiers that are given in 'the 'sortVector' but are missing in the transcripts will be added.
`deleteTiersThatAreNotInTheSortVector`	Logical; if `TRUE` tiers that are not matched by the regular expressions in 'sortVector' will be deleted. Otherwise the 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,
addMissingTiers=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,
addMissingTiers=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.

`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.path` Character string; Path(s) to the media files that correspond to this transcript object.

`normalization.systime` POSIXct; Time of the last normalization.

`fulltext.systime` POSIXct; [READ ONLY] Time of the last creation of the full texts.

`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

`modification.systime` POSIXct; [READ ONLY] Time of the last modification of the transcript. Modifications after importing the annotation file by applying one/some of the packages function(s). Manual changes of the transcript by the user are not tracked!

`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,
  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.
`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.skip.double.files, 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. - 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,
  annotationsWithReversedTimes = TRUE,
  overlappingAnnotations = TRUE,
  annotationsWithTimesBelowZero = TRUE,
  missingTiers = TRUE,
  showWarning = FALSE
)
```

Arguments

x	Corpus object.
filterTranscriptNames	Vector of character strings; names of the transcripts to be included.
annotationsWithReversedTimes	Logical; If TRUE annotations with reversed times will be deleted
overlappingAnnotations	Logical; If TRUE overlapping annotations will be corrected.
annotationsWithTimesBelowZero	Logical; If TRUE annotations before 0 sec will be corrected.
missingTiers	Logical; If TRUE tiers missing in @tiers slot of the transcript object will be added.
showWarning	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)
x@history[[length(x@history)]]

# 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. - 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,
  annotationsWithReversedTimes = TRUE,
  overlappingAnnotations = TRUE,
  annotationsWithTimesBelowZero = TRUE,
  missingTiers = TRUE,
  showWarning = FALSE
)
```

Arguments

t	Transcript object.
annotationsWithReversedTimes	Logical; If TRUE annotations with reversed times will be deleted
overlappingAnnotations	Logical; If TRUE overlapping annotations will be corrected.
annotationsWithTimesBelowZero	Logical; If TRUE annotations before 0 sec will be corrected.
missingTiers	Logical; If TRUE tiers missing in @tiers slot of the transcript object will be added.
showWarning	Logical; If TRUE a warning notice will be shown upon correction.

Value

Transcript object;

See Also

transcripts_cure

Examples

```
library(act)

# --- annotationsWithReversedTimes: 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)


# --- annotationsWithTimesBelowZero: 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)


# --- overlappingAnnotations: will end where the next starts
t<- examplecorpus@transcripts[[1]]
t@annotations <- t@annotations[order(t@annotations$tier.name, t@annotations$startSec), ]
t@annotations$endSec[1] <- 8
t2 <- act::transcripts_cure_single(t)
tail(t2@history, n=1)


# --- missingTiers: will be added to @tiers in transcript object
t<- examplecorpus@transcripts[[1]]
t@annotations <- t@annotations[order(t@annotations$tier.name, t@annotations$startSec), ]
t@annotations$tier.name[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$tier.name, t@annotations$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$tier.name[8] <- "NEW"
t2 <- act::transcripts_cure_single(t, showWarning=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))
test@history[length(test@history)]

# 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$transcript.names)

# compare the the original and modified corpus object
length(examplecorpus@transcripts)
length(test@transcripts)
setdiff(names(examplecorpus@transcripts), names(test@transcripts))
```

`transcripts_filter` *Filter all transcripts in a corpus*

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,
  filterOnlyTheseTranscripts = NULL,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  preserveTimes = TRUE,
  sort = c("none", "tier>startSec", "startSec>tier")
)
```

Arguments

`x`	Corpus object;
`filterTranscriptNames`	Vector of character strings; names of transcripts to remain in the transcripts. If left unspecified, all transcripts will remain in the transcripts.
`filterOnlyTheseTranscripts`	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.
`preserveTimes`	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.tier.names <- unique(act::tiers_all(examplecorpus)$name)
some.tier.names <- all.tier.names[1:10]
x <- act::transcripts_filter(examplecorpus, filterTierNames=some.tier.names)
x@history[[length(x@history)]]
```

---

transcripts_filter_single

Filter a single transcript

Description

Filter a transcript object and return the filtered transcript 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_single(
  t,
  filterTierNames = NULL,
  filterSectionStartsec = NULL,
  filterSectionEndsec = NULL,
  preserveTimes = 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.
`preserveTimes`	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$tier.name)

# Filter transcript to only contain annotations of the FIRST tier
t2 <- act::transcripts_filter_single(t1, filterTierNames=t1@tiers$name[1])
t2@tiers
table(t2@annotations$tier.name)

# 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$tier.names)
t2@tiers
table(t2@annotations$tier.name)

# --- 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,
 preserveTime=FALSE)
cbind(t2@annotations$startSec,t2@annotations$endSec)
```

`transcripts_merge` *Merge several transcripts*

Description

Merges several transcript objects in a corpus object. One transcript is the destination transcript (the transcript that will be updated and receives the new data). The other transcripts are the update transcripts (they contain the data that will replace data in teh destination transcript). The update transcripts need to contain a tier in which the update sections are marked with a specific character string.

Usage

```
transcripts_merge(
  x,
  destinationTranscriptName,
  updateTranscriptNames,
  identifierTier = "update",
  identifierPattern = ".+",
  eraseUpdateSectionsCompletely = TRUE
)
```

Arguments

`x`	Corpus object;
`destinationTranscriptName`	Character strings; name of transcript hat willl be updated.
`updateTranscriptNames`	Vector of character strings; names of transcripts that contain the updates.
`identifierTier`	Character string; regular expression that identifies the tier in which the sections are marked, that will be inserted into transDestination.
`identifierPattern`	Character string; regular expression that identifies the sections that will be inserted into transDestination.
`eraseUpdateSectionsCompletely`	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 transUpdates be erased.

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

See Also

transcripts_merge2

Examples

```
library(act)

# We need three transcripts  to demonstrate the function \code{transcripts_merge}:
# - the destination transcript: "update_destination"
# - two transcripts that contain updates: "update_update1 and "update_update2"

#Have a look at the annotations in the destination transcript first.
#It contains 2 annotations:
examplecorpus@transcripts[["update_destination"]]@annotations
#Have a look at the annotations in the update_update1 transcript, too:
#It contains 3 annotations:
examplecorpus@transcripts[["update_update1"]]@annotations

# Run the function with only one update:
test <- act::transcripts_merge(x=examplecorpus,
   destinationTranscriptName="update_destination",
   updateTranscriptNames = "update_update1")

#Have a look at the annotations in the destination transcript again.
#It now contains 5 annotations:
test@transcripts[["update_destination"]]@annotations


# Run the function with two transcript objects for updates:
test <- act::transcripts_merge(x=examplecorpus,
destinationTranscriptName="update_destination",
updateTranscriptNames = c("update_update1","update_update2"))

#Have a look at the annotations in the destination transcript again.
#It now contains 8 annotations:
test@transcripts[["update_destination"]]@annotations

# Compare the transcript in the original and in the modified corpus object.
# The update transcript objects are gone:
act::info_summarized(examplecorpus)$transcript.names
act::info_summarized(test)$transcript.names

#Have a look at the history of the corpus object
test@history
```

`transcripts_merge2` *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_merge2(
  destinationTranscript,
  updateTranscripts,
  identifierTier = "update",
  identifierPattern = ".+",
  eraseUpdateSectionsCompletely = TRUE
)
```

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.

`eraseUpdateSectionsCompletely`
: 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.

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

See Also

transcripts_merge

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_merge2(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, outputPath=path)
```

`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,
  toUpperCase = FALSE,
  toLowerCase = FALSE,
  trim = FALSE,
  stopIfNotUnique = 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.
toUpperCase	Logical; Convert transcript names all to upper case.
toLowerCase	Logical; Convert transcript names all to lower case.
trim	Logical; Remove leading and trailing spaces in names.
stopIfNotUnique	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
test@history[length(test@history)]

# convert to lower case
test <- act::transcripts_rename(examplecorpus, toLowerCase=TRUE)
test@history[length(test@history)]

# search replace
test <- act::transcripts_rename(examplecorpus,
 searchPatterns=c("ARG", "BOL"),
 searchReplacements = c("ARGENTINA", "BOLIVIA")
)
test@history[length(test@history)]

# search replace ignoring upper and lower case
test <- act::transcripts_rename(examplecorpus,
 searchPatterns=c("(?i)arg", "(?i)bol"),
 searchReplacements = c("ARGENTINA", "BOLIVIA")
)
test@history[length(test@history)]


# 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	Character string; Which full text should be created; accepts the following values: `fulltext.bytier`,`fulltext.bytime`,`fulltext`.
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,
  path_replacementMatrixCSV = "",
  transcriptNames = NULL,
  forceUpdate = FALSE
)
```

Arguments

x	Corpus object.
path_replacementMatrixCSV	Character string; path to replacement matrix in CSV format. If empty, the default replacement matrix that comes with the package will be used.
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.

Examples

```
library(act)

examplecorpus <- act::transcripts_update_normalization(x=examplecorpus)
```

Index