Package 'nzilbb.labbcat'

May 9, 2026

Encoding UTF-8

Version 1.5-1

Date 2026-03-31

Title Accessing Data Stored in 'LaBB-CAT' Instances

Imports jsonlite, httr, stringr, utils, rstudioapi, xml2

Description 'LaBB-CAT' is a web-based language corpus management system developed by the New Zealand Institute of Language, Brain and Behaviour (NZILBB) - see <https://labbcat.canterbury.ac.nz>. This package defines functions for accessing corpus data in a 'LaBB-CAT' instance. You must have at least version 20230818.1400 of 'LaBB-CAT' to use this package.
For more information about 'LaBB-CAT', see
Robert Fromont and Jennifer Hay (2008) <doi:10.3366/E1749503208000142>
or
Robert Fromont (2017) <doi:10.1016/j.csl.2017.01.004>.

License GPL (>= 3)

Copyright New Zealand Institute of Language, Brain and Behaviour, University of Canterbury

URL https://nzilbb.github.io/labbcat-R/, https://labbcat.canterbury.ac.nz

BugReports https://github.com/nzilbb/labbcat-R/issues

RoxygenNote 7.3.2

Suggests testthat (>= 2.1.0)

NeedsCompilation no

Author Robert Fromont [aut, cre] (ORCID: <https://orcid.org/0000-0001-5271-5487>)

Maintainer Robert Fromont <robert.fromont@canterbury.ac.nz>

Repository CRAN

Date/Publication 2026-03-31 20:30:02 UTC

Contents

`addDictionaryEntry` *Adds an entry to a dictionary*

Description

This function creates adds a new entry to the given dictionary.

Usage

```
addDictionaryEntry(labbcat.url, manager.id, dictionary.id, key, entry)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`manager.id`	The layer manager ID of the dictionary, as returned by getDictionaries
`dictionary.id`	The ID of the dictionary, as returned by getDictionaries
`key`	The key (word) in the dictionary to add an entry for.
`entry`	The value (definition) for the given key.

Details

You must have edit privileges in LaBB-CAT in order to be able to use this function.

Value

NULL if the entry was added, or a list of error messages if not.

See Also

Other dictionary functions: `addLayerDictionaryEntry()`, `deleteLexicon()`, `getDictionaries()`, `getDictionaryEntries()`, `loadLexicon()`, `removeDictionaryEntry()`, `removeLayerDictionaryEntry()`

Examples

```
## Not run:
## Add the word "robert" to the CELEX wordform pronunciation dictionary
addDictionaryEntry(labbcat.url, "CELEX-EN", "Phonology (wordform)", "robert", "'rQ-b@t")

## End(Not run)
```

addLayerDictionaryEntry *Adds an entry to a layer dictionary*

Description

This function adds a new entry to the dictionary that manages a given layer, and updates all affected tokens in the corpus. Words can have multiple entries.

Usage

```
addLayerDictionaryEntry(labbcat.url, layer.id, key, entry)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
layer.id	The ID of the layer with a dictionary configured to manage it.
key	The key (word) in the dictionary to add an entry for.
entry	The value (definition) for the given key.

Details

You must have edit privileges in LaBB-CAT in order to be able to use this function.

Value

NULL if the entry was added, or a list of error messages if not.

See Also

Other dictionary functions: addDictionaryEntry(), deleteLexicon(), getDictionaries(), getDictionaryEntries(), loadLexicon(), removeDictionaryEntry(), removeLayerDictionaryEntry()

Examples

```
## Not run:
## Add a pronunciation for the word "robert" to the phonemes layer dictionary
addLayerDictionaryEntry(labbcat.url, "phonemes", "robert", "'rQ-b@t")

## End(Not run)
```

annotatorExt *Retrieve annotator's "ext" resource*

Description

Retrieve a given resource from an annotator's "ext" web app. Annotators are modules that perform different annotation tasks, and can optionally implement functionality for providing extra data or extending functionality in an annotator-specific way. If the annotator implements an "ext" web app, it can provide resources and implement a mechanism for iterrogating the annotator. This function provides a mechanism for accessing these resources via R.

Usage

```
annotatorExt(labbcat.url, annotator.id, resource, parameters = NULL)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance.
annotator.id	ID of the annotator to interrogate.
resource	The name of the file to retrieve or instance method (function) to invoke. Possible values for this depend on the specific annotator being interrogated.
parameters	Optional list of ordered parameters for the instance method (function).

Value

The resource requested.

Examples

```
## Not run:
## Get the version of the currently installed LabelMapper annotator:
annotatorExt(labbcat.url, "LabelMapper", "getVersion")

## Get the summary of the segment to speakerDependentPhone mapping
## implemented by the LabelMapper annotator:
summaryJson <- annotatorExt(labbcat.url,
             "LabelMapper", "summarizeMapping", list("segment","speakerDependentPhone"))
summary <- jsonlite::fromJSON(summaryJson)

## End(Not run)
```

appendFromPraat *Appends measurements from Praat to a dataframe of matches.*

Description

This is a version of processWithPraat that can have a dataframe of matches piped into it, and returns the dataframe with columns appended.

Usage

```
appendFromPraat(
  matches,
  start.column,
  end.column,
  praat.script,
  window.offset,
  gender.attribute = "participant_gender",
  attributes = NULL,
  no.progress = FALSE,
  labbcat.url = NULL,
  column.prefix = NULL
)
```

Arguments

matches	A dataframe returned by getMatches or getAllUtterances, identifying the results to which acoustic measurments should be appended.
start.column	The column of matches containing the start time in seconds.
end.column	The column of matches containing the end time in seconds.
praat.script	Script to run on each match. This may be a single string or a character vector.
window.offset	In many circumstances, you will want some context before and after the sample start/end time. For this reason, you can specify a "window offset" - this is a number of seconds to subtract from the sample start and add to the sample end time, before extracting that part of the audio for processing. For example, if the sample starts at 2.0s and ends at 3.0s, and you set the window offset to 0.5s, then Praat will extract a sample of audio from 1.5s to 3.5s, and do the selected processing on that sample. The best value for this depends on what the praat.script is doing; if you are getting formants from vowels, including some context ensures that the formants at the edges are more accurate (in LaBB-CAT's web interface, the default value for this 0.025), but if you're getting max pitch or COG during a segment, most likely you want a window.offset of 0 to ensure neighbouring segments don't influence the measurement.
gender.attribute	Which participant attribute represents the participant's gender.

attributes	Vector of participant attributes to make available to the script. For example, if you want to use different acoustic parameters depending on what the gender of the speaker is, including the "participant_gender" attribute will make a variable called participant_gender$ available to the praat script, whose value will be the gender of the speaker of that segment.
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().
labbcat.url	URL to the LaBB-CAT instance (instead of inferring it from matches).
column.prefix	A string to prefix each new column name with.

Details

It instructs the LaBB-CAT server to invoke Praat for a set of sound intervals, in order to extract acoustic measures.

The exact measurements to return depend on the praat.script that is invoked. This is a Praat script fragment that will run once for each sound interval specified.

There are functions to allow the generation of a number of pre-defined praat scripts for common tasks such as formant, pitch, intensity, and centre of gravity – see praatScriptFormants, praatScriptCentreOfGravity, praatScriptIntensity and praatScriptPitch.

You can provide your own script, either by building a string with your code, or loading one from a file.

LaBB-CAT prefixes praat.script with code to open a sound file and extract a defined part of it into a Sound object which is then selected.

LaBB-CAT Remove's this Sound object after the script finishes executing. Any other objects created by t
move'd before the end of the script (otherwise Praat runs out of memory during very large batches)

LaBB-CAT assumes that all calls to the function 'print' correspond to fields for export and each field must be printed on its own line. Specifically it scans for lines of the form:

print 'myOutputVariable' 'newline$'

Variables that can be assumed to be already set in the context of the script are:

- *windowOffset* – the value used for the Window Offset; how much context to include.
- *windowAbsoluteStart* – the start time of the window extracted relative to the start of the original audio file.
- *windowAbsoluteEnd* – the end time of the window extracted relative to the start of the original audio file.
- *windowDuration* – the duration of the window extracted (including window offset).
- *targetAbsoluteStart* – the start time of the target interval relative to the start of the original audio file.
- *targetAbsoluteEnd* – the end time of the target interval relative to the start of the original audio file.
- *targetStart* – the start time of the target interval relative to the start of the window extracted.
- *targetEnd* – the end time of the target interval relative to the start of the window extracted.
- *targetDuration* – the duration of the target interval.
- *sampleNumber* – the number of the sample within the set of samples being processed.
- *sampleName$* – the name of the extracted/selected Sound object.

Value

matches with the acoustic measurements appended as new columns.

See Also

- processWithPraat
- getMatches

Other Praat-related functions: fragmentLabels(), fragmentTranscripts(), praatScriptCentreOfGravity(), praatScriptFastTrack(), praatScriptFormants(), praatScriptIntensity(), praatScriptPitch(), processWithPraat()

Examples

```
## Not run:
## Get all tokens of /I/
results <- getMatches(labbcat.url, list(segment="I")) |>
    appendFromPraat( ## get F1 and F2 for the mid point of the vowel
       Target.segment.start, Target.segment.end, # for the vowel
       praatScriptFormants(),
       window.offset=0.5) ## get F1 and F2

## Get all tokens of /i:/
results <- getMatches(labbcat.url, list(segment="i")) |>
   appendFromPraat(
       Target.segment.start, Target.segment.end, ## for the target vowel...
       paste(
           ## ... get first 3 formants at three points during the sample ...
           praatScriptFormants(c(1,2,3), c(0.25,0.5,0.75)),
           ## ... the mean, min, and max pitch ...
           praatScriptPitch(get.mean=TRUE, get.minimum=TRUE, get.maximum=TRUE),
           ## ... the max intensity ...
           praatScriptIntensity(),
           ## ... and the CoG using powers 1 and 2
           praatScriptCentreOfGravity(powers=c(1.0,2.0))),
       window.offset=0.5)

## Get all tokens of /s/
results <- getMatches(labbcat.url, list(segment="s")) |>
   appendFromPraat(
       Target.segment.start, Target.segment.end,
     readLines("acousticMeasurements.praat")) ## execute a custom script loaded form a file

## End(Not run)
```

---

appendLabels *Appends labels of annotations on given layers to a dataframe of matches.*

Description

This is a version of getMatchLabels that can have a dataframe of matches piped into it, and returns the dataframe with columns appended.

Usage

```
appendLabels(
  matches,
  layer.ids,
  target.offset = 0,
  annotations.per.layer = 1,
  page.length = 1000,
  no.progress = FALSE,
  labbcat.url = NULL,
  column.prefix = NULL
)
```

Arguments

`matches`	A dataframe returned by getMatches or getAllUtterances, identifying the results to which annotation labels should be appended.
`layer.ids`	A vector of layer IDs.
`target.offset`	The distance from the original target of the match, e.g. • *0* – find annotations of the match target itself • *1* – find annotations of the token immediately *after* match target • *-1* – find annotations of the token immediately *before* match target
`annotations.per.layer`	The number of annotations on the given layer to retrieve. In most cases, there's only one annotation available. However, tokens may, for example, be annotated with 'all possible phonemic transcriptions', in which case using a value of greater than 1 for this parameter provides other phonemic transcriptions, for tokens that have more than one.
`page.length`	In order to prevent timeouts when there are a large number of matches or the network connection is slow, rather than retrieving matches in one big request, they are retrieved using many smaller requests. This parameter controls the number of results retrieved per request.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().
`labbcat.url`	URL to the LaBB-CAT instance (instead of inferring it from `matches`).
`column.prefix`	A string to prefix each new column name with.

Value

`matches` with the labels appended as new columns.

See Also

- getMatchLabels
- getMatches

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(orthography="quake")) |>
   appendLabels("topic") ## Get the topic annotations for the matches

## End(Not run)
```

`appendOffsets` *Appends temporal alignments on given layers to a dataframe of matches.*

Description

Appends labels and start/end offsets of annotations on a given layer, to a given dataframe of matches returned from getMatches or getAllUtterances.

Usage

```
appendOffsets(
  matches,
  layer.ids,
  target.offset = 0,
  annotations.per.layer = 1,
  anchor.confidence.min = 50,
  page.length = 1000,
  no.progress = FALSE,
  labbcat.url = NULL,
  column.prefix = NULL
)
```

Arguments

`matches`	A dataframe returned by getMatches or getAllUtterances, identifying the results to which annotation labels should be appended.
`layer.ids`	A vector of layer IDs.
`target.offset`	The distance from the original target of the match, e.g. • *0* – find annotations of the match target itself • *1* – find annotations of the token immediately *after* match target • *-1* – find annotations of the token immediately *before* match target

`annotations.per.layer`
: The number of annotations on the given layer to retrieve. In most cases, there's only one annotation available. However, tokens may, for example, be annotated with 'all possible phonemic transcriptions', in which case using a value of greater than 1 for this parameter provides other phonemic transcriptions, for tokens that have more than one.

`anchor.confidence.min`
: The minimum confidence for alignments, e.g.
  - *0* – return all alignments, regardless of confidence;
  - *50* – return only alignments that have been at least automatically aligned;
  - *100* – return only manually-set alignments.

`page.length`
: In order to prevent timeouts when there are a large number of matches or the network connection is slow, rather than retrieving matches in one big request, they are retrieved using many smaller requests. This parameter controls the number of results retrieved per request.

`no.progress`
: TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

`labbcat.url`
: URL to the LaBB-CAT instance (instead of inferring it from `matches`).

`column.prefix`
: A string to prefix each new column name with.

Details

This is a version of getMatchAlignments that can have a dataframe of matches piped into it, and returns the dataframe with columns appended.

You can specify a threshold for confidence in the alignment, which is a value from 0 (not aligned) to 100 (manually aligned). The default is 50 (automatically aligned), so only alignments that have been at least automatically aligned are specified. For cases where there's a token but its alignment confidence falls below the threshold, a label is returned, but the start/end times are NA.

Value

`matches` with the labels, start times, and end times appended as new columns.

See Also

- getMatches
- getMatchAlignments

Examples

```
## Not run:
## Get all tokens of /I/
results <- getMatches(labbcat.url, list(segment="I")) |>
    appendOffsets("segment", target.offset=1, ## Get the segment following the token
        anchor.confidence.min=100) ## with alignment if it's been manually aligned

## End(Not run)
```

`countAnnotations` *Gets the number of annotations on the given layer of the given transcript*

Description

Returns the number of annotations on the given layer of the given transcript.

Usage

```
countAnnotations(labbcat.url, id, layer.id, max.ordinal = NULL)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	A transcript ID (i.e. transcript name)
`layer.id`	A layer ID
`max.ordinal`	The maximum ordinal for the counted annotations. e.g. a max.ordinal of 1 will ensure that only the first annotation for each parent is returned. If max.ordinal is null, then all annotations are counted, regardless of their ordinal.

Value

The number of annotations on that layer

See Also

- getTranscriptIds
- getTranscriptIdsInCorpus
- getTranscriptIdsWithParticipant

Examples

```
## Not run:
## Count the number of words in UC427_ViktoriaPapp_A_ENG.eaf
token.count <- countAnnotations(labbcat.url, "UC427_ViktoriaPapp_A_ENG.eaf", "orthography")

## End(Not run)
```

countMatchingAnnotations *Gets the number of annotations matching a particular pattern*

Description

Returns the number of annotations in the corpus that match the given expression.

Usage

```
countMatchingAnnotations(labbcat.url, expression)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
expression	An expression that determines which annotations match. This must match by either id or layer.id. The expression language is currently not well defined, but is based on JavaScript syntax. e.g.:

- `id == 'ew_0_456'`
- `['ew_2_456', 'ew_2_789', 'ew_2_101112'].includes(id)`
- `layerId == 'orthography' && !/th[aeiou].+/.test(label)`
- `graph.id == 'AdaAicheson-01.trs' && layer.id == 'orthography' && start.offset >`
- `layer.id == 'utterance' && all('word').includes('ew_0_456')`
- `layerId = 'utterance' && labels('orthography').includes('foo')`
- `layerId = 'utterance' && labels('participant').includes('Ada')`

Value

The number of annotations that match the expression.

See Also

getMatchingAnnotations

Examples

```
## Not run:
## count the number of topic tags that include the word 'quake'
countMatchingAnnotations(labbcat.url, "layer.id == 'topic' && /.*quake.*/.test(label)")

## End(Not run)
```

`deleteLayer` *Deletes an existing layer*

Description

This function deletes an existing annotation layer, including all annotation data associated with it.

Usage

```
deleteLayer(labbcat.url, layer.id)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`layer.id`	The ID of the layer to delete.

Details

You must have administration privileges in LaBB-CAT in order to be able to use this function.

Value

NULL, or an error message if deletion failed.

See Also

Other Annotation layer functions: `generateLayer()`, `getLayer()`, `getLayerIds()`, `getLayers()`, `newLayer()`, `saveLayer()`

Examples

```
## Not run:
## Delete the phonemes layer
deleteLayer(labbcat.url, "phonemes")

## End(Not run)
```

`deleteLexicon` *Delete a previously loaded lexicon*

Description

By default LaBB-CAT includes a layer manager called the Flat Lexicon Tagger, which can be configured to annotate words with data from a dictionary loaded from a plain text file (e.g. a CSV file).

Usage

```
deleteLexicon(labbcat.url, lexicon)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance.
`lexicon`	The name of the lexicon to delete, e.g. 'cmudict'

Details

This function deletes such a lexicon, which was previously added using loadLexicon.

You must have editing privileges in LaBB-CAT in order to be able to use this function.

Value

NULL if the deletion was successful, or an error message if not.

See Also

Other dictionary functions: `addDictionaryEntry()`, `addLayerDictionaryEntry()`, `getDictionaries()`, `getDictionaryEntries()`, `loadLexicon()`, `removeDictionaryEntry()`, `removeLayerDictionaryEntry()`

Examples

```
## Not run:
## Delete the previously loaded CMU Pronouncing Dictionary lexicon
deleteLexicon(labbcat.url, "cmudict")

## End(Not run)
```

deleteMedia *Delete a transcript's media file*

Description

This function deletes the given media file associated with the given transcript.

Usage

```
deleteMedia(labbcat.url, id, file.name)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The ID transcript whose media will be deleted.
file.name	The media file name, e.g. media.file$name

Details

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

See Also

- getAvailableMedia
- saveMedia

Examples

```
## Not run:
## delete the mp3 file of a transcript from the server
deleteMedia(labbcat.url, "my-transcript.eaf", "my-transcript.mp3")

## End(Not run)
```

deleteParticipant *Deletes a participant record*

Description

This function deletes the identified participant from the corpus, but only if they do not appear in any transcripts.

Usage

```
deleteParticipant(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The participant ID - either the unique internal database ID, or their name.

Value

TRUE if the participant's record was delete, FALSE otherwise.

See Also

- getParticipant
- saveParticipant

Examples

```
## Not run:
## Create a new participant record
saveParticipant(labbcat.url, "Juan Perez")

### Delete the participant we just created
deleteParticipant(labbcat.url, "Juan Perez")

## End(Not run)
```

---

deleteTranscript *Delete a transcript from the corpus*

Description

This function deletes the given transcript, and all associated files.

Usage

```
deleteTranscript(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The ID transcript to delete.

Details

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

Value

The ID of the deleted transcript

Examples

```
## Not run:
## delete a transcript from the server
deleteTranscript(labbcat.url, "my-transcript.eaf")

## End(Not run)
```

`expressionFromAttributeValue`

Generates a query expression for matching a single-value transcript/participant attribute, for use with getMatches

---

Description

This function generates a query expression fragment which can be passed as the transcript.expression or participant.expression parameter of getMatches, (or the expression parameter of getMatchingTranscriptIds or getMatchingParticipantIds) using a list of possible values for a given attribute.

Usage

```
expressionFromAttributeValue(transcript.attribute, values, not = FALSE)
```

Arguments

`transcript.attribute`	The transcript attribute to filter by.
`values`	A list of possible values for transcript.attribute.
`not`	Whether to match the given IDs (FALSE), or everything *except* the given IDs.

Details

The attribute defined by transcript.attribute is expected to have exactly one value. If it may have multiple values, use expressionFromAttributeValues instead.

Value

A transcript query expression which can be passed as the transcript.expression parameter of getMatches or the expression parameter of getMatchingTranscriptIds

See Also

- expressionFromAttributeValues
- expressionFromTranscriptTypes
- expressionFromIds
- getMatches

Examples

```
## Not run:
## Perform a search
languages <- c("en","en-NZ")
results <- getMatches(labbcat.url, list(segment="I"),
                      transcript.expression = expressionFromAttributeValue(
                            "transcript_language", languages))

## End(Not run)
```

`expressionFromAttributeValues` *Generates a query expression for matching a multi-value transcript/participant attribute, for use with getMatches*

Description

This function generates a query expression fragment which can be passed as the transcript.expression or participant.expression parameter of getMatches, (or the expression parameter of getMatchingTranscriptIds or getMatchingParticipantIds) using a list of possible values for a given transcript attribute.

Usage

```
expressionFromAttributeValues(transcript.attribute, values, not = FALSE)
```

Arguments

`transcript.attribute`	The transcript attribute to filter by.
`values`	A list of possible values for transcript.attribute.
`not`	Whether to match the given IDs (FALSE), or everything *except* the given IDs.

Details

The attribute defined by transcript.attribute is expected to have possibly more than one value. If it can have only one value, use expressionFromAttributeValue instead.

Value

A transcript query expression which can be passed as the transcript.expression parameter of getMatches or the expression parameter of getMatchingTranscriptIds

See Also

expressionFromAttributeValue

expressionFromTranscriptTypes

expressionFromIds

getMatches

Examples

```
## Not run:
## Perform a search
languages <- c("en","es")
results <- getMatches(labbcat.url, list(segment="I"),
                      participant.expression = expressionFromAttributeValues(
                            "participant_languagesSpoken", languages))

## End(Not run)
```

---

expressionFromAttributeValuesCount

Generates a query expression for matching a transcript/participant attribute, for use with getMatches

---

Description

This function generates a query expression fragment which can be passed as the transcript.expression or participant.expression parameter of getMatches, (or the expression parameter of getMatchingTranscriptIds or getMatchingParticipantIds) matching by the number of values for a given attribute.

Usage

```
expressionFromAttributeValuesCount(
  transcript.attribute,
  comparison = "==",
  count
)
```

Arguments

transcript.attribute	The transcript attribute to filter by.
comparison	A string representing the operator to use for comparison, one of "<", "<=", "==", "!=", ">=", ">".
count	The number to compare the count of values to.

Details

The attribute defined by transcript.attribute is expected to have possibly more than one value, although single-value attributes may have 0 or 1 values, and this function can be used to distinguish these two possibilities as well.

Value

A transcript query expression which can be passed as the transcript.expression parameter of getMatches or the expression parameter of getMatchingTranscriptIds

See Also

expressionFromAttributeValues

expressionFromTranscriptTypes

expressionFromIds

getMatches

Examples

```
## Not run:
## Search only transcripts including multilingual participants
results <- getMatches(labbcat.url, list(segment="I"),
                      participant.expression = expressionFromAttributeValuesCount(
                            "participant_languages", ">=", 2))

## Search only transcripts with no restrictions specified
results <- getMatches(labbcat.url, list(segment="I"),
                      transcript.expression = expressionFromAttributeValuesCount(
                            "transcript_restrictions", "==", 0))

## End(Not run)
```

---

expressionFromIds — *Generates a query expression for matching transcripts or participants by ID, for use with getMatches*

Description

This function generates a query expression fragment which can be passed as the transcript.expression or participant.expression parameter of getMatches, using a list of corresponding IDs.

Usage

```
expressionFromIds(ids, not = FALSE)
```

Arguments

ids	A list of IDs.
not	Whether to match the given IDs (FALSE), or everything *except* the given IDs.

Value

A query expression which can be passed as the transcript.expression or participant.expression parameter of getMatches or the expression parameter of getMatchingTranscriptIds or getMatchingParticipantIds

See Also

expressionFromAttributeValue

expressionFromAttributeValues

expressionFromTranscriptTypes

getMatches

Examples

```
## Not run:
## Perform a search
transcript.ids <- c("AP511_MikeThorpe.eaf", "BR2044_OllyOhlson.eaf")
results <- getMatches(labbcat.url, list(segment="I"),
                      transcript.expression = expressionFromIds(transcript.ids))

## End(Not run)
```

---

expressionFromTranscriptTypes

Generates a transcript query expression for matching transcripts by type, for use with getMatches or getMatchingTranscriptIds

---

Description

This function generates a transcript query expression fragment which can be passed as the transcript.expression parameter of getMatches, (or the expression parameter of getMatchingTranscriptIds) in order to identify transcripts using a list of transcript types.

Usage

```
expressionFromTranscriptTypes(transcript.types, not = FALSE)
```

Arguments

transcript.types	A list of transcript types.
not	Whether to match the given IDs (FALSE), or everything *except* the given IDs.

Value

A transcript query expression which can be passed as the transcript.expression parameter of getMatches or the expression parameter of getMatchingTranscriptIds

See Also

expressionFromAttributeValue

expressionFromAttributeValues

expressionFromIds

getMatches

Examples

```
## Not run:
## Perform a search of interviews or monologues
transcript.types <- c("interview","monologue")
results <- getMatches(labbcat.url, list(segment="I"),
  transcript.expression = expressionFromTranscriptTypes(transcript.types))

## Perform a search of all transcripts that aren't word-lists.
results <- getMatches(labbcat.url, list(segment="I"),
  transcript.expression = expressionFromTranscriptTypes("wordlist", NOT=true))

## End(Not run)
```

---

formatTranscript *Gets transcript(s) in a given format*

Description

This function gets whole transcripts from 'LaBB-CAT', converted to a given format (by default, Praat TextGrid).

Usage

```
formatTranscript(
  labbcat.url,
  id,
  layer.ids,
  mime.type = "text/praat-textgrid",
  path = ""
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The transcript ID (transcript name) of the sound recording, or a vector of transcript IDs. If the same ID appears more than one, the formatted file is downloaded only once.
layer.ids	A vector of layer IDs.
mime.type	Optional content-type - "text/praat-textgrid" is the default, but your LaBB-CAT installation may support other formats, which can be discovered using getSerializerDescriptors.
path	Optional path to directory where the files should be saved.

Details

NB Although many formats will generate exactly one file for each interval (e.g. mime.type=text/praat-textgrid), this is not guaranted; some formats generate a single file or a fixed collection of files regardless of how many IDs there are.

Value

The name of the file, which is saved in the current directory, or the given path, or a list of names of files, if multiple id's were specified.

If a list of files is returned, they are in the order that they were returned by the server, which *should* be the order that they were specified in the id list.

See Also

getSerializerDescriptors

Examples

```
## Not run:
## Get the TextGrid of a recording
textgrid.file <- formatTranscript(labbcat.url, "AP2505_Nelson.eaf",
    c("word", "segment"), path="textgrids")

## Get all the transcripts of a given participant
transcript.ids <- getTranscriptIdsWithParticipant(labbcat.url, "AP2505_Nelson")

## Download all the TextGrids, including the utterances, transcript, and segment layers
textgrid.files <- formatTranscript(
    labbcat.url, transcript.ids, c("utterance", "word", "segment"))


## End(Not run)
```

`fragmentAudio` *Gets sound fragments from 'LaBB-CAT'.*

Description

This is a version of getSoundFragments that can have a dataframe of matches piped into it.

Usage

```
fragmentAudio(
  matches,
  sample.rate = NULL,
  path = "",
  no.progress = FALSE,
  start.column = Line,
  end.column = LineEnd,
  labbcat.url = NULL
)
```

Arguments

`matches`	A dataframe returned by getMatches or getAllUtterances, identifying the results to which acoustic measurments should be appended.
`sample.rate`	Optional sample rate in Hz - if a positive integer, then the result is a mono file with the given sample rate.
`path`	Optional path to directory where the files should be saved.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().
`start.column`	The column of `matches` containing the start time in seconds.
`end.column`	The column of `matches` containing the end time in seconds.
`labbcat.url`	URL to the LaBB-CAT instance (instead of inferring it from `matches`).

Details

It gets fragments of audio from LaBB-CAT, as wav files.

Value

`matches` with the acoustic measurements appended as new columns.

See Also

- processWithPraat
- getMatches

Examples

```
## Not run:
## Get all tokens of "the"
the.tokens <- getMatches(labbcat.url, "the")
## Get a 22kHz sample rate wav file for each matched utterance
the.wavs <- the.tokens |> fragmentTranscripts(sample.rate = 22050)

## End(Not run)
```

`fragmentData` *Gets binary annotation data in fragments.*

Description

This is a version of getFragmentAnnotationData that can have a dataframe of matches piped into it.

Usage

```
fragmentData(
  matches,
  layer.id,
  path = "",
  no.progress = FALSE,
  start.column = Line,
  end.column = LineEnd,
  labbcat.url = NULL
)
```

Arguments

`matches`	A dataframe returned by getMatches or getAllUtterances, identifying the results to which acoustic measurments should be appended.
`layer.id`	The ID of the MIME-typed layer.
`path`	Optional path to directory where the files should be saved.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().
`start.column`	The column of `matches` containing the start time in seconds.
`end.column`	The column of `matches` containing the end time in seconds.
`labbcat.url`	URL to the LaBB-CAT instance (instead of inferring it from `matches`).

Details

In some annotation layers, the annotations have not only a textual label, but also binary data associated with it; e.g. an image or a data file. In these cases, the 'type' of the layer is a MIME type, e.g. 'image/png'. This function gets annotations between given start/end times on the given MIME-typed layer, and retrieves the binary data as files, whose names are returned by the function.

Value

matches with the acoustic measurements appended as new columns.

See Also

- processWithPraat
- getMatches

Examples

```
## Not run:
## Get all tokens of "vivid"
vivid.tokens <- getMatches(labbcat.url, "vivid")
## Get mediapipe image annotations for during the tokens
vivid.faces <- vivid.tokens |>
    fragmentData(
        "mediapipe", path = "png",
        start.column=Target.word.start, end.column=Target.word.end)

## End(Not run)
```

---

fragmentLabels *Gets annotations in fragments.*

Description

This is a version of getFragmentAnnotations that can have a dataframe of matches piped into it.

Usage

```
fragmentLabels(
  matches,
  layer.ids,
  sep = " ",
  partial.containment = FALSE,
  no.progress = FALSE,
  start.column = Line,
  end.column = LineEnd,
  labbcat.url = NULL,
  column.prefix = NULL
)
```

Arguments

matches	A dataframe returned by getMatches or getAllUtterances, identifying the results to which acoustic measurments should be appended.
layer.ids	A vector of layer IDs.

sep	The separator to use when concatenating labels when multiple annotations are in the given interval.
partial.containment	Whether to include annotations that are only partially contained in the given interval.
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().
start.column	The column of matches containing the start time in seconds.
end.column	The column of matches containing the end time in seconds.
labbcat.url	URL to the LaBB-CAT instance (instead of inferring it from matches).
column.prefix	A string to prefix each new column name with.

Details

It gets annotations between given start/end times on given layers. If more than one annotation matches, labels are concatentated together.

Value

matches with the acoustic measurements appended as new columns.

See Also

- processWithPraat
- getMatches

Other Praat-related functions: appendFromPraat(), fragmentTranscripts(), praatScriptCentreOfGravity(), praatScriptFastTrack(), praatScriptFormants(), praatScriptIntensity(), praatScriptPitch(), processWithPraat()

Examples

```
## Not run:
## Get all tokens of /I/
results <- getMatches(labbcat.url, list(topic = ".*quake.*")) |>
    fragmentLabels( ## concatenate labels of words between topic.start and topic.end
       c("word"), start.column=topic.start, end.column=topic.end)

## End(Not run)
```

`fragmentTranscripts` *Gets transcript fragments in a given format.*

Description

This is a version of getFragments that can have a dataframe of matches piped into it.

Usage

```
fragmentTranscripts(
  matches,
  layer.ids,
  mime.type = "text/praat-textgrid",
  path = "",
  start.column = Line,
  end.column = LineEnd,
  labbcat.url = NULL
)
```

Arguments

`matches`	A dataframe returned by getMatches or getAllUtterances, identifying the results to which acoustic measurments should be appended.
`layer.ids`	A vector of layer IDs.
`mime.type`	Optional content-type - "text/praat-textgrid" is the default, but your LaBB-CAT installation may support other formats, which can be discovered using getSerializerDescriptors.
`path`	Optional path to directory where the files should be saved.
`start.column`	The column of `matches` containing the start time in seconds.
`end.column`	The column of `matches` containing the end time in seconds.
`labbcat.url`	URL to the LaBB-CAT instance (instead of inferring it from `matches`).

Details

It gets fragments of transcripts from LaBB-CAT, converted to a given file format (by default, Praat TextGrid).

NB Although many formats will generate exactly one file for each interval (e.g. mime.type=text/praat-textgrid), this is not guaranteed; some formats generate a single file or a fixed collection of files regardless of how many fragments there are.

Value

`matches` with the acoustic measurements appended as new columns.

See Also

- processWithPraat
- getMatches

Other Praat-related functions: appendFromPraat(), fragmentLabels(), praatScriptCentreOfGravity(), praatScriptFastTrack(), praatScriptFormants(), praatScriptIntensity(), praatScriptPitch(), processWithPraat()

Examples

```
## Not run:
## Get all tokens of "the"
the.tokens <- getMatches(labbcat.url, "the")
## Get a TextGrid for each matched utterance, including word and segment intervals
the.textgrids <- the.tokens |> fragmentTranscripts(c("utterance", "word", "segment"))
## Get a CSV for the same utterances
the.textgrids <- the.tokens |> fragmentTranscripts(
    c("utterance", "word", "segment"), mime.type = "text/csv", path="csv")

## End(Not run)
```

---

generateLayer *Generates a layer*

Description

Generates annotations on a given layer for all transcripts in the corpus.

Usage

```
generateLayer(labbcat.url, layer.id, no.progress = FALSE)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
layer.id	The ID of the layer to generate.
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

The final status of the layer generation task.

See Also

getAllUtterances

Other Annotation layer functions: deleteLayer(), getLayer(), getLayerIds(), getLayers(), newLayer(), saveLayer()

Examples

```
## Not run:
## Generate all phonemic transcription annotations
generateLayer(labbcat.url, "phonemes")

## End(Not run)
```

`generateLayerUtterances` *Generates a layer for a given set of utterances*

Description

Generates annotations on a given layer for a given set of utterances, e.g. force-align selected utterances of a participant.

Usage

```
generateLayerUtterances(
  labbcat.url,
  match.ids,
  layer.id,
  collection.name = NULL,
  no.progress = FALSE
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`match.ids`	A vector of annotation IDs, e.g. the MatchId column, or the URL column, of a results set.
`layer.id`	The ID of the layer to generate.
`collection.name`	An optional name for the collection, e.g. the participant ID.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

The final status of the layer generation task.

See Also

getAllUtterances

Examples

```
## Not run:
## Get all utterances of a participant
allUtterances <- getAllUtterances(labbcat.url, "AP2505_Nelson")

## Force-align the participant's utterances
generateLayerUtterances(labbcat.url, allUtterances$MatchId, "htk", "AP2505_Nelson")

## End(Not run)
```

---

getAllUtterances *Get all utterances of participants*

Description

Identifies all utterances of a given set of participants.

Usage

```
getAllUtterances(
  labbcat.url,
  participant.ids,
  transcript.types = NULL,
  main.participant = TRUE,
  max.matches = NULL,
  no.progress = FALSE
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`participant.ids`	A list of participant IDs to identify the utterances of.
`transcript.types`	An optional list of transcript types to limit the results to. If null, all transcript types will be searched.
`main.participant`	TRUE to search only main-participant utterances, FALSE to search all utterances.
`max.matches`	The maximum number of matches to return, or null to return all.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

A data frame identifying matches, containing the following columns:

- *SearchName* A name based on the pattern – the same for all rows
- *Number* Row number
- *Transcript* Name of the transcript in which the match was found
- *Line* The start offset of the utterance/line
- *LineEnd* The end offset of the utterance/line
- *MatchId* A unique ID for the matching target token
- *Before.Match* Transcript text immediately before the match
- *Text* Transcript text of the match
- *Before.Match* Transcript text immediately after the match
- *Target.word* Text of the target word token
- *Target.word.start* Start offset of the target word token
- *Target.word.end* End offset of the target word token
- *Target.segment* Label of the target segment (only present if the segment layer is included in the pattern)
- *Target.segment.start* Start offset of the target segment (only present if the segment layer is included in the pattern)
- *Target.segment.end* End offset of the target segment (only present if the segment layer is included in the pattern)

See Also

getParticipantIds

Examples

```
## Not run:
## get all utterances of the given participants
participant.ids <- getParticipantIds(labbcat.url)[1:3]
results <- getAllUtterances(labbcat.url, participant.ids)

## results$MatchId can be used to access results

## End(Not run)
```

getAnchors *Gets the given anchors in the given transcript*

Description

Lists the given anchors in the given transcript.

Usage

```
getAnchors(labbcat.url, id, anchor.id, page.length = 1000)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	A transcript ID (i.e. transcript name)
anchor.id	A vector of anchor IDs (or a string representing one anchor ID)
page.length	In order to prevent timeouts when there are a large number of matches or the network connection is slow, rather than retrieving anchors in one big request, they are retrieved using many smaller requests. This parameter controls the number of anchors retrieved per request.

Value

A named list of anchors, with members:

- *id* The annotation's unique ID,
- *offset* The offset from the beginning (in seconds if it's a transcript of a recording, or in characters if it's a text document)
- *confidence* A rating from 0-100 of the confidence of the offset, e.g. 10: default value, 50: force-aligned, 100: manually aligned

See Also

getAnnotations

Examples

```
## Not run:
## Get the first 20 orthography tokens in UC427_ViktoriaPapp_A_ENG.eaf
orthography <- getAnnotations(labbcat.url, "UC427_ViktoriaPapp_A_ENG.eaf", "orthography", 20, 0)

## Get the start anchors for the above tokens
word.starts <- getAnchors(labbcat.url, "UC427_ViktoriaPapp_A_ENG.eaf", orthography$startId)

## End(Not run)
```

`getAnnotations` *Gets the annotations on the given layer of the given transcript*

Description

Returns the annotations on the given layer of the given transcript.

Usage

```
getAnnotations(
  labbcat.url,
  id,
  layer.id,
  max.ordinal = NULL,
  page.length = NULL,
  page.number = NULL
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	A transcript ID (i.e. transcript name)
`layer.id`	A layer ID
`max.ordinal`	The maximum ordinal for the returned annotations. e.g. a max.ordinal of 1 will ensure that only the first annotation for each parent is returned. If max.ordinal is null, then all annotations are returned, regardless of their ordinal.
`page.length`	The maximum number of annotations to return, or null to return all
`page.number`	The zero-based page number to return, or null to return the first page

Value

A named list of annotations, with members:

- *id* The annotation's unique ID
- *layerId* The name of the layer it comes from
- *label* The value of the annotation
- *startId* The ID of the start anchor,
- *endId* The ID of the end anchor,
- *parentId* The ID of the parent annotation,
- *ordinal* The ordinal of the annotation among its peers,
- *confidence* A rating from 0-100 of the confidence of the label e.g. 10: default value, 50: automatically generated, 100: manually annotated

See Also

- getTranscriptIds
- getTranscriptIdsInCorpus
- getTranscriptIdsWithParticipant
- countAnnotations

Examples

```
## Not run:
## Get all the orthography tokens in UC427_ViktoriaPapp_A_ENG.eaf
orthography <- getAnnotations(labbcat.url, "UC427_ViktoriaPapp_A_ENG.eaf", "orthography")

## Get the first 20 orthography tokens in UC427_ViktoriaPapp_A_ENG.eaf
orthography <- getAnnotations(labbcat.url, "UC427_ViktoriaPapp_A_ENG.eaf", "orthography", 20, 0)

## End(Not run)
```

`getAnnotatorDescriptor` *Gets annotator information*

Description

Retrieve information about an annotator. Annotators are modules that perform different annotation tasks. This function provides information about a given annotator, for example the currently installed version of the module, what configuration parameters it requires, etc.

Usage

```
getAnnotatorDescriptor(labbcat.url, annotator.id)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance.
`annotator.id`	ID of the annotator module.

Value

The annotator info:

- *annotatorId* The annotators's unique ID
- *version* The currently install version of the annotator.
- *info* HTML-encoded description of the function of the annotator.
- *infoText* A plain text version of $info (converted automatically).

- *hasConfigWebapp* Determines whether the annotator includes a web-app for installation or general configuration.
- *configParameterInfo* An HTML-encoded definition of the installation config parameters, including a list of all parameters, and the encoding of the parameter string.
- *configParameterInfoText* A plain text version of $configParameterInfo (converted automatically).
- *hasTaskWebapp* Determines whether the annotator includes a web-app for task parameter configuration.
- *taskParameterInfo* An HTML-encoded definition of the task parameters, including a list of all parameters, and the encoding of the parameter string.
- *taskParameterInfoText* A plain text version of $taskParameterInfo (converted automatically).
- *hasExtWebapp* Determines whether the annotator includes an extras web-app which implements functionality for providing extra data or extending functionality in an annotator-specific way.
- *extApiInfo* An HTML-encoded document containing information about what endpoints are published by the ext web-app.
- *extApiInfoText* A plain text version of $extApiInfo (converted automatically).

See Also

- annotatorExt
- newLayer

Examples

```
## Not run:
## Get information about the BAS Annotator
basAnnotator <- getAnnotatorDescriptor("https://labbcat.canterbury.ac.nz/demo/", "BASAnnotator")
cat(basAnnotator$infoText)

## End(Not run)
```

`getAvailableMedia` *List the media available for the given transcript*

Description

List the media available for the given transcript

Usage

```
getAvailableMedia(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	A transcript ID (i.e. transcript name)

Value

A named list of media files available for the given transcript, with members:

- *trackSuffix* The track suffix of the media
- *mimeType* The MIME type of the file
- *url* URL to the content of the file
- *name* Name of the file

See Also

- getTranscriptIds
- saveMedia
- deleteMedia

Examples

```
## Not run:
## List the media files available for BR2044_OllyOhlson.eaf
media <- getAvailableMedia(labbcat.url, "BR2044_OllyOhlson.eaf")

## End(Not run)
```

---

getCorpusIds *Gets a list of corpus IDs*

Description

Returns a list of corpora in the given 'LaBB-CAT' instance.

Usage

```
getCorpusIds(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A list of corpus IDs

Examples

```
## Not run:
## List corpora
corpora <- getCorpusIds("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

`getDeserializerDescriptors` *Lists the descriptors of all registered deserializers*

Description

Returns a list of deserializers, which are modules that import transcriptions and annotation structures from a specific file format, e.g. Praat TextGrid, plain text, etc.

Usage

```
getDeserializerDescriptors(labbcat.url)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance

Value

A list of serializers, each including the following information:

- *name* The name of the format.
- *version* The installed version of the serializer module.
- *fileSuffixes* The normal file name suffixes (extensions) of the files.
- *mimeType* The MIME type of the format, i.e. the value to use as the *mimeType* parameter of getFragments

Examples

```
## Not run:
## List file upload formats supported
formats <- getDeserializerDescriptors("https://labbcat.canterbury.ac.nz/demo/")

## can we upload as plain text?
plainTextSupported <- "text/plain" %in% formats$mimeType

## End(Not run)
```

getDictionaries *List the dictionaries available*

Description

List the dictionaries available

Usage

```
getDictionaries(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A named list of layer manager IDs, each of which containing a list of dictionaries that the layer manager makes available.

See Also

Other dictionary functions: addDictionaryEntry(), addLayerDictionaryEntry(), deleteLexicon(), getDictionaryEntries(), loadLexicon(), removeDictionaryEntry(), removeLayerDictionaryEntry()

Examples

```
## Not run:
## List the dictionaries available
dictionaries <- getDictionaries("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

getDictionaryEntries *Lookup entries in a dictionary*

Description

Lookup entries in a dictionary

Usage

```
getDictionaryEntries(labbcat.url, manager.id, dictionary.id, keys)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
manager.id	The layer manager ID of the dictionary, as returned by getDictionaries
dictionary.id	The ID of the dictionary, as returned by getDictionaries
keys	A list of keys (words) identifying entries to look up

Value

A data frame with the keys and their dictionary entries, if any.

See Also

Other dictionary functions: addDictionaryEntry(), addLayerDictionaryEntry(), deleteLexicon(), getDictionaries(), loadLexicon(), removeDictionaryEntry(), removeLayerDictionaryEntry()

Examples

```
## Not run:
keys <- c("the", "quick", "brown", "fox")

## get the pronunciations according to CELEX
entries <- getDictionaryEntries(labbcat.url, "CELEX-EN", "Phonology (wordform)", keys)

## End(Not run)
```

---

getFragmentAnnotationData

Gets binary annotation data in fragments.

Description

In some annotation layers, the annotations have not only a textual label, but also binary data associated with it; e.g. an image or a data file. In these cases, the 'type' of the layer is a MIME type, e.g. 'image/png'. This function gets annotations between given start/end times on the given MIME-typed layer, and retrieves the binary data as files, whose names are returned by the function.

Usage

```
getFragmentAnnotationData(
  labbcat.url,
  transcript.id,
  start,
  end,
  layer.id,
  path = "",
  no.progress = FALSE
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
transcript.id	The transcript ID (transcript name) of the sound recording, or a vector of transcript IDs.
start	The start time in seconds, or a vector of start times.
end	The end time in seconds, or a vector of end times.
layer.id	The ID of the MIME-typed layer.
path	Optional path to directory where the files should be saved.
no.progress	TRUE to supress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

The names of the files.

See Also

- getFragmentAnnotations
- getFragments
- getSoundFragments

Examples

```
## Not run:
## Get mediapipe image annotations for the eleventh second of a transcript
png.files <- getFragmentAnnotationData(
    labbcat.url, c("AP511_MikeThorpe.eaf"), c(10), c(11), c("mediapipe"), path = "png")

## End(Not run)
```

---

getFragmentAnnotations

Gets annotations in fragments.

Description

This function gets annotations between given start/end times on given layers. If more than one annotation matches, labels are concatentated together.

Usage

```
getFragmentAnnotations(
  labbcat.url,
  transcript.id,
  participant.id,
  start,
  end,
  layer.ids,
  sep = " ",
  partial.containment = FALSE,
  no.progress = FALSE
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
transcript.id	The transcript ID (transcript name) of the sound recording, or a vector of transcript IDs.
participant.id	The participant ID of the annotations, or a vector of participant IDs.
start	The start time in seconds, or a vector of start times.
end	The end time in seconds, or a vector of end times.
layer.ids	A vector of layer IDs.
sep	The separator to use when concatenating labels when multiple annotations are in the given interval.
partial.containment	Whether to include annotations that are only partially contained in the given interval.
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

A data frame with three columns for each layer in layer.ids:

- The annotation labels concatenated together
- The start time of the first annotation
- The end time of the last annotation

See Also

- getFragments
- getSoundFragments

Examples

```
## Not run:
## Get some span-layer intervales
topics <- getMatches(labbcat.url, list(topic = ".*quake.*"))

## Get concantenated word tokens for each topic annotation
topic.tokens <- getFragmentAnnotations(
   labbcat.url, topics$Transcript, topics$Participant, topics$topic.start, topics$topic.end,
    c("word"))

## End(Not run)
```

`getFragments` *Gets transcript fragments in a given format.*

Description

This function gets fragments of transcripts from LaBB-CAT, converted to a given file format (by default, Praat TextGrid).

Usage

```
getFragments(
  labbcat.url,
  id,
  start,
  end,
  layer.ids,
  mime.type = "text/praat-textgrid",
  path = ""
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	The transcript ID (transcript name) of the sound recording, or a vector of transcript IDs.
`start`	The start time in seconds, or a vector of start times.
`end`	The end time in seconds, or a vector of end times.
`layer.ids`	A vector of layer IDs.
`mime.type`	Optional content-type - "text/praat-textgrid" is the default, but your LaBB-CAT installation may support other formats, which can be discovered using getSerializerDescriptors.
`path`	Optional path to directory where the files should be saved.

Details

NB Although many formats will generate exactly one file for each interval (e.g. mime.type=text/praat-textgrid), this is not guaranteed; some formats generate a single file or a fixed collection of files regardless of how many fragments there are.

Value

The name of the file, which is saved in the current directory, or a list of names of files, if multiple id's/start's/end's were specified

If a list of files is returned, they are in the order that they were returned by the server, which *should* be the order that they were specified in the id/start/end lists.

See Also

getSerializerDescriptors

Examples

```
## Not run:
## Get the 5 seconds starting from 10s after the beginning of a recording
textgrid.file <- getFragments(labbcat.url, "AP2505_Nelson.eaf", 10.0, 15.0,
    c("transcript", "phonemes"), path="samples")

## Load some search results previously exported from LaBB-CAT
results <- read.csv("results.csv", header=T)

## Get a list of fragment TextGrids, including the utterances, transcript, and phonemes layers
textgrid.files <- getFragments(
    labbcat.url, results$Transcript, results$Line, results$LineEnd,
    c("utterance", "word", "phonemes"))

## Get a list of fragment TextGrids
textgrid.files <- getFragments(
    labbcat.url, results$Transcript, results$Line, results$LineEnd)

## End(Not run)
```

---

getGraphIds *Deprecated synonym for getTranscriptIds.*

Description

Returns a list of graph IDs (i.e. transcript names).

Usage

```
getGraphIds(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A list of graph IDs

See Also

getTranscriptIds

Examples

```
## Not run:
## List all transcripts
transcripts <- getGraphIds("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

---

getGraphIdsInCorpus *Deprecated synonym for getTranscriptIdsInCorpus.*

Description

Returns a list of corpora in the given 'LaBB-CAT' instance.

Usage

```
getGraphIdsInCorpus(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The ID (name) of the corpus

Value

A list of corpus IDs

See Also

getGraphIdsInCorpus

Examples

```
## Not run:
## List transcripts in the QB corpus
transcripts <- getGraphIdsInCorpus("https://labbcat.canterbury.ac.nz/demo/", "QB")

## End(Not run)
```

`getGraphIdsWithParticipant` *Deprecated synonym for getTranscriptIdsWithParticipant.*

Description

Returns a list of IDs of graphs (i.e. transcript names) that include the given participant.

Usage

```
getGraphIdsWithParticipant(labbcat.url, id)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	A participant ID

Value

A list of graph IDs

See Also

`getTranscriptIdsWithParticipant`

Examples

```
## Not run:
## List transcripts in which UC427_ViktoriaPapp_A_ENG speaks
transcripts <- getGraphIdsWithParticipant(labbcat.url, "UC427_ViktoriaPapp_A_ENG")

## End(Not run)
```

getId *Gets the store's ID*

Description

The store's ID - i.e. the ID of the 'LaBB-CAT' instance.

Usage

```
getId(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

The annotation store's ID

Examples

```
## Not run:
## Get ID of LaBB-CAT instance
instance.id <- getId("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

getLayer *Gets a layer definition*

Description

Gets a layer definition

Usage

```
getLayer(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	ID of the layer to get the definition for

Value

The definition of the given layer, with members:

- *id* The layer's unique ID
- *parentId* The layer's parent layer ID
- *description* The description of the layer
- *alignment* The layer's alignment - 0 for none, 1 for point alignment, 2 for interval alignment
- *peers* Whether children have peers or not
- *peersOverlap* Whether child peers can overlap or not
- *parentIncludes* Whether the parent t-includes the child
- *saturated* Whether children must temporally fill the entire parent duration (true) or not (false)
- *parentIncludes* Whether the parent t-includes the child
- *type* The type for labels on this layer
- *validLabels* List of valid label values for this layer

See Also

Other Annotation layer functions: `deleteLayer()`, `generateLayer()`, `getLayerIds()`, `getLayers()`, `newLayer()`, `saveLayer()`

Examples

```
## Not run:
## Get the definition of the orthography layer
orthography.layer <- getLayer("https://labbcat.canterbury.ac.nz/demo/", "orthography")

## End(Not run)
```

`getLayerIds` *Gets a list of layer IDs*

Description

Layer IDs are annotation 'types'.

Usage

```
getLayerIds(labbcat.url)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance

Value

A list of layer IDs

See Also

Other Annotation layer functions: deleteLayer(), generateLayer(), getLayer(), getLayers(), newLayer(), saveLayer()

Examples

```
## Not run:
## Get names of all layers
layer.ids <- getLayerIds("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

---

getLayers *Gets a list of layer definitions*

Description

Gets a list of layer definitions

Usage

```
getLayers(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A list of layer definitions, with members:

- *id* The layer's unique ID
- *parentId* The layer's parent layer ID
- *description* The description of the layer
- *alignment* The layer's alignment - 0 for none, 1 for point alignment, 2 for interval alignment
- *peers* Whether children have peers or not
- *peersOverlap* Whether child peers can overlap or not
- *parentIncludes* Whether the parent t-includes the child
- *saturated* Whether children must temporally fill the entire parent duration (true) or not (false)
- *parentIncludes* Whether the parent t-includes the child
- *type* The type for labels on this layer
- *validLabels* List of valid label values for this layer

See Also

Other Annotation layer functions: deleteLayer(), generateLayer(), getLayer(), getLayerIds(), newLayer(), saveLayer()

Examples

```
## Not run:
## Get definitions of all layers
layers <- getLayers("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

---

getMatchAlignments *Gets temporal alignments of matches on a given layer*

Description

Gets labels and start/end offsets of annotations on a given layer, identified by given match IDs.

Usage

```
getMatchAlignments(
  labbcat.url,
  match.ids,
  layer.ids,
  target.offset = 0,
  annotations.per.layer = 1,
  anchor.confidence.min = 50,
  include.match.ids = FALSE,
  page.length = 1000,
  no.progress = FALSE
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
match.ids	A vector of annotation IDs, e.g. the MatchId column, or the URL column, of a results set.
layer.ids	A vector of layer IDs.
target.offset	The distance from the original target of the match, e.g. • *0* – find annotations of the match target itself • *1* – find annotations of the token immediately *after* match target • *-1* – find annotations of the token immediately *before* match target

`annotations.per.layer`
: The number of annotations on the given layer to retrieve. In most cases, there's only one annotation available. However, tokens may, for example, be annotated with 'all possible phonemic transcriptions', in which case using a value of greater than 1 for this parameter provides other phonemic transcriptions, for tokens that have more than one.

`anchor.confidence.min`
: The minimum confidence for alignments, e.g.
  - *0* – return all alignments, regardless of confidence;
  - *50* – return only alignments that have been at least automatically aligned;
  - *100* – return only manually-set alignments.

`include.match.ids`
: Whether or not the data frame returned includes the original MatchId column or not.

`page.length`
: In order to prevent timeouts when there are a large number of matches or the network connection is slow, rather than retrieving matches in one big request, they are retrieved using many smaller requests. This parameter controls the number of results retrieved per request.

`no.progress`
: TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Details

You can specify a threshold for confidence in the alignment, which is a value from 0 (not aligned) to 100 (manually aligned). The default is 50 (automatically aligned), so only alignments that have been at least automatically aligned are specified. For cases where there's a token but its alignment confidence falls below the threshold, a label is returned, but the start/end times are NA.

Value

A data frame with label, start time, and end time, for each layer.

See Also

- getMatches
- getMatchLabels

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(segment="I"))

## Get the segment following the token, with alignment if it's been manually aligned
following.segment <- getMatchAlignments(labbcat.url, results$MatchId, "segment",
    target.offset=1, anchor.confidence.min=100)

## End(Not run)
```

`getMatches` *Search for tokens*

Description

Searches through transcripts for tokens matching the given pattern.

Usage

```
getMatches(
  labbcat.url,
  pattern,
  participant.expression = NULL,
  transcript.expression = NULL,
  main.participant = TRUE,
  aligned = NULL,
  matches.per.transcript = NULL,
  words.context = 0,
  max.matches = NULL,
  overlap.threshold = NULL,
  anchor.confidence.min = NULL,
  page.length = 1000,
  no.progress = FALSE
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`pattern`	An object representing the pattern to search for. This can be: • A string, representing a search of the orthography layer - spaces are taken to be word boundaries • A single named list, representing a one-column search - names are taken to be layer IDs • A list of named lists, representing a multi-column search - the outer list represents the columns of the search matrix where each column 'immediately follows' the previous, and the names of the inner lists are taken to be layer IDs • A named list (or for segment layers, a list of named lists) fully replicating the structure of the search matrix in the LaBB-CAT browser interface, with one element called "columns", containing a named list for each column. Each element in the "columns" named list contains an element named "layers", whose value is a named list (or a list of named lists) for patterns to match on each layer, and optionally an element named "adj", whose value is a number representing the maximum distance, in tokens, between this

column and the next column - if "adj" is not specified, the value defaults to 1, so tokens are contiguous.
Each element in the "layers" named list is named after the layer it matches, and the value is a named list with the following possible elements:

- *pattern* A regular expression to match against the label
- *min* An inclusive minimum numeric value for the label
- *max* An exclusive maximum numeric value for the label
- *not* TRUE to negate the match
- *anchorStart* TRUE to anchor to the start of the annotation on this layer (i.e. the matching word token will be the first at/after the start of the matching annotation on this layer)
- *anchorEnd* TRUE to anchor to the end of the annotation on this layer (i.e. the matching word token will be the last before/at the end of the matching annotation on this layer)
- *target* TRUE to make this layer the target of the search; the results will contain one row for each match on the target layer

Examples of valid pattern objects include:

```
## the word 'the' followed immediately by a word starting with an orthographic vowel
pattern <- "the [aeiou].*"

## a word spelt with "k" but pronounced "n" word initially
pattern <- list(orthography = "k.*", phonemes = "n.*")

## the word 'the' followed immediately by a word starting with a phonemic vowel
pattern <- list(
    list(orthography = "the"),
    list(phonemes = "[cCEFHiIPqQuUV0123456789~#\\$@].*"))

## the word 'the' followed immediately or with one intervening word by
## a hapax legomenon (word with a frequency of 1) that doesn't start with a vowel
pattern <- list(columns = list(
    list(layers = list(
           orthography = list(pattern = "the")),
         adj = 2),
    list(layers = list(
       phonemes = list(not = TRUE, pattern = "[cCEFHiIPqQuUV0123456789~#\\$@].*"),
           frequency = list(max = "2")))))

## words that contain the /I/ phone followed by the /l/ phone
## (multiple patterns per word currently only works for segment layers)
pattern <- list(segment = list("I", "l"))

## words that contain the /I/ phone followed by the /l/ phone, targeting the /l/ segment
## (multiple patterns per word currently only works for segment layers)
pattern <- list(segment = list("I", list(pattern="l", target=T)))

## words where the spelling starts with "k", but the first segment is /n/
```

```
pattern <- list(
  orthography = "k.*",
  segment = list(pattern = "n", anchorStart = T)
```

`participant.expression`
: An optional participant query expression for identifying participants to search the utterances of. This should be the output of expressionFromIds, expressionFromAttributeValue, or expressionFromAttributeValues, or more than one concatentated together and delimited by ' && '. If not supplied, utterances of all participants will be searched.

`transcript.expression`
: An optional transript query expression for identifying transcripts to search in. This should be the output of expressionFromIds, expressionFromTranscriptTypes, expressionFromAttributeValue, or expressionFromAttributeValues, or more than one concatentated together and delimited by ' && '. If not supplied, all transcripts will be searched.

`main.participant`
: TRUE to search only main-participant utterances, FALSE to search all utterances.

`aligned`
: This parameter is deprecated and will be removed in future versions; please use `anchor.confidence.min = 50` instead.

`matches.per.transcript`
: Optional maximum number of matches per transcript to return. NULL means all matches.

`words.context`
: Number of words context to include in the 'Before.Match' and 'After.Match' columns in the results.

`max.matches`
: The maximum number of matches to return, or null to return all.

`overlap.threshold`
: The percentage overlap with other utterances before simultaneous speech is excluded, or null to include overlapping speech.

`anchor.confidence.min`
: The minimum confidence for alignments, e.g.

- *0* - return all alignments, regardless of confidence;
- *50* - return only alignments that have been at least automatically aligned;
- *100* - return only manually-set alignments.

`page.length`
: In order to prevent timeouts when there are a large number of matches or the network connection is slow, rather than retrieving matches in one big request, they are retrieved using many smaller requests. This parameter controls the number of results retrieved per request.

`no.progress`
: TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

A data frame identifying matches, containing the following columns:

- *Title* The title of the LaBB-CAT instance
- *Version* The current version of the LaBB-CAT instance
- *SearchName* A name based on the pattern – the same for all rows
- *MatchId* A unique ID for the matching target token
- *Transcript* Name of the transcript in which the match was found
- *Participant* Name of the speaker
- *Corpus* The corpus of the transcript
- *Line* The start offset of the utterance/line
- *LineEnd* The end offset of the utterance/line
- *Before.Match* Transcript text immediately before the match
- *Text* Transcript text of the match
- *After.Match* Transcript text immediately after the match
- *Number* Row number
- *URL* URL of the first matching word token
- *Target.word* Text of the target word token
- *Target.word.start* Start offset of the target word token
- *Target.word.end* End offset of the target word token
- *Target.segment* Label of the target segment (only present if the segment layer is included in the pattern)
- *Target.segment.start* Start offset of the target segment (only present if the segment layer is included in the pattern)
- *Target.segment.end* End offset of the target segment (only present if the segment layer is included in the pattern)

See Also

- getFragments
- getSoundFragments
- getMatchLabels
- getMatchAlignments
- processWithPraat
- getParticipantIds

Examples

```
## Not run:
## the word 'the' followed immediately by a word starting with an orthographic vowel
theThenOrthVowel <- getMatches(labbcat.url, "the [aeiou]")

## a word spelt with "k" but pronounced "n" word initially
knWords <- getMatches(labbcat.url, list(orthography = "k.*", phonemes = "n.*"))
```

```
## the word 'the' followed immediately by a word starting with an phonemic vowel
theThenPhonVowel <- getMatches(
  labbcat.url, list(
    list(orthography = "the"),
    list(phonemes = "[cCEFHiIPqQuUV0123456789~#\\$@].*")))

## the word 'the' followed immediately or with one intervening word by
## a hapax legomenon (word with a frequency of 1) that doesn't start with a vowel
results <- getMatches(
  labbcat.url, list(columns = list(
    list(layers = list(
           orthography = list(pattern = "the")),
         adj = 2),
    list(layers = list(
           phonemes = list(not=TRUE, pattern = "[cCEFHiIPqQuUV0123456789~#\\$@].*"),
           frequency = list(max = "2")))),
  overlap.threshold = 5)

## all tokens of the KIT vowel, from the interview or monologue
## of the participants AP511_MikeThorpe and BR2044_OllyOhlson
results <- getMatches(labbcat.url, list(segment="I"),
  participant.expression = expressionFromIds(c("AP511_MikeThorpe","BR2044_OllyOhlson")),
  transcript.expression = expressionFromTranscriptTypes(c("interview","monologue")))

## all tokens of the KIT vowel for male speakers who speak English
results <- getMatches(labbcat.url, list(segment="I"),
  participant.expression = paste(
    expressionFromAttributeValue("participant_gender", "M"),
    expressionFromAttributeValues("participant_languages_spoken", "en"),
    sep=" && "))

## results$Text is the text that matched
## results$MatchId can be used to access results using other functions

## End(Not run)
```

`getMatchingAnnotationData` *Gets binary data for annotations that match a particular pattern.*

Description

In some annotation layers, the annotations have not only a textual label, but also binary data associated with it; e.g. an image or a data file. In these cases, the 'type' of the layer is a MIME type, e.g. 'image/png'. This function gets annotations that match the given expression on a MIME-typed layer, and retrieves the binary data as files, whose names are returned by the function.

Usage

```
getMatchingAnnotationData(labbcat.url, expression, path = "")
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
expression	An expression that determines which annotations match. This must match by either id or layer.id. The expression language is currently not well defined, but is based on JavaScript syntax. e.g. • id == 'e_144_17346' • ['e_144_17346', 'e_144_17347', 'e_144_17348'].includes(id) • layer.id == 'mediapipe' && graph.id == 'AdaAicheson-01.trs'
path	Optional path to directory where the files should be saved.

Value

The names of the files.

See Also

- getMatchingAnnotations
- getFragmentAnnotationData

Examples

```
## Not run:
## Get mediapipe image annotations for the eleventh second of a transcript
expression = paste(sep="&&",
               "layer.id == 'mediapipe'",
               "graph.id == 'AP511_MikeThorpe.eaf'",
               "start.offset >= 10",
               "end.offset < 11")
png.files <- getMatchingAnnotationData(labbcat.url, expression, path="png")

## End(Not run)
```

---

getMatchingAnnotations

Gets a list of annotations that match a particular pattern

Description

Returns the annotations in the corpus that match the given expression.

Usage

```
getMatchingAnnotations(
  labbcat.url,
  expression,
  page.length = NULL,
  page.number = NULL
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
expression	An expression that determines which annotations match. This must match by either id or layer.id. The expression language is currently not well defined, but is based on JavaScript syntax. e.g. • id == 'ew_0_456' • layerId == 'orthography' && !/th[aeiou].+/.test(label) • graph.id == 'AdaAicheson-01.trs' && layer.id == 'orthography' && start.offset > • layer.id == 'utterance' && all('word').includes('ew_0_456') • layerId = 'utterance' && labels('orthography').includes('foo') • layerId = 'utterance' && labels('participant').includes('Ada')
page.length	The maximum number of IDs to return, or null to return all
page.number	The zero-based page number to return, or null to return the first page

Details

The results can be exhaustive, by omitting page.length and page.number, or they can be a subset (a 'page') of results, by given page.length and page.number values.

Value

A list of annotations.

See Also

countMatchingAnnotations

Examples

```
## Not run:
## get all topic annotations whose label includes the word 'quake'
quake.topics <- getMatchingAnnotations(
                   labbcat.url, "layer.id == 'topic' && /.*quake.*/.test(label)")

## End(Not run)
```

---

getMatchingGraphIds *Deprecated synonym for getMatchingTranscriptIds.*

Description

Gets a list of IDs of graphs (i.e. transcript names) that match a particular pattern.

Usage

```
getMatchingGraphIds(
  labbcat.url,
  expression,
  page.length = NULL,
  page.number = NULL,
  order = NULL
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
expression	An expression that determines which graphs match
page.length	The maximum number of IDs to return, or null to return all
page.number	The zero-based page number to return, or null to return the first page
order	An expression that determines the order the graphs are listed in - if specified, this must include the keyword 'ASC' for ascending or 'DESC' for descending order.

Details

The results can be exhaustive, by omitting pageLength and page.number, or they can be a subset (a 'page') of results, by given pageLength and page.number values.

The order of the list can be specified. If ommitted, the graphs are listed in ID order.

The expression language is currently not well defined, but is based on JavaScript syntax.

- The *labels* function can be used to represent a list of all the annotation labels on a given layer. For example, each transcript can have multiple participants, so the participant labels (names) are represented by: labels('participant')'
- Use the *includes* function on a list to test whether the list contains a given element. e.g. to match transcripts that include the participant 'Joe' use: `labels('participant').includes('Joe')`
- Use the *first* function to identify the first (or the only) annotation on a given layer. e.g. the annotation representing the transcript's corpus is: `first('corpus')`
- Single annotations have various attributes, including 'id', 'label', 'ordinal', etc. e.g. the name of the transcript's corpus is: `first('corpus').label`
- Regular expressions can be matched by using expressions like '/regex/.test(str)', e.g. to test if the ID starts with 'BR' use: `/^BR.+/.test(id)` or to test if the transcript's corpus includes a B use: `/.*B.*/.test(first('corpus').label)`

Expressions such as those in the examples can be used.

Value

A list of graph IDs (i.e. transcript names)

Examples

```
## Not run:
## Get all transcripts whose names start with "BR"
transcripts <- getMatchingGraphIds(labbcat.url, "/^BR.+/.test(id)")

## Get the first twenty transcripts in the "QB" corpus
transcripts <- getMatchingGraphIds(
        labbcat.url, "first('corpus').label = 'QB'", 20, 0)

## Get the second transcript that has "QB247_Jacqui" as a speaker
transcripts <- getMatchingGraphIds(
        labbcat.url, "labels('participant').includes('QB247_Jacqui')", 1, 1)

## Get all transcripts in the QB corpus whose names start with "BR"
## in word-count order
transcripts <- getMatchingGraphIds(
        labbcat.url, "first('corpus').label = 'QB' && /^BR.+/.test(id)",
        order="first('transcript_word_count').label ASC")

## End(Not run)
```

---

getMatchingParticipantIds

Gets a list of IDs of participants that match a particular pattern

---

Description

Gets a list of IDs of participants that match a particular pattern.

Usage

```
getMatchingParticipantIds(
  labbcat.url,
  expression,
  page.length = NULL,
  page.number = NULL
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
expression	An expression that determines which participants match
page.length	The maximum number of IDs to return, or null to return all
page.number	The zero-based page number to return, or null to return the first page

Details

The results can be exhaustive, by omitting page.length and page.number, or they can be a subset (a 'page') of results, by given page.length and page.number values.

The expression language is currently not well defined, but is based on JavaScript syntax.

- The *labels* function can be used to represent a list of all the annotation labels on a given layer. For example, each participant can have multiple corpora, so the corpus labels (names) are represented by: `labels('corpus')`
- Use the *includes* function on a list to test whether the list contains a given element. e.g. to match participants that include the corpus 'QB' use: `labels('corpus').includes('QB')`
- Use the *first* function to identify the first (or the only) annotation on a given layer. e.g. the annotation representing the participant's gender is: `first('participant_gender')`
- Single annotations have various attributes, including 'id', 'label', 'ordinal', etc. e.g. the label of the participant's gender is: `first('participant_gender').label`
- Regular expressions can be matched by using expressions like '/regex/.test(str)', e.g. to test if the ID starts with 'BR' use: `/^BR.+/.test(id)` or to test if the participant's gender includes 'binary' use: `/.*binary.*/.test(first('participant_gender').label)`

Expressions such as those in the examples can be used.

Value

A list of paricipant IDs

Examples

```
## Not run:
## Get all participants whose IDs start with "BR"
participants <- getMatchingParticipantIds(labbcat.url, "/^BR.+/.test(id)")

## Get the first twenty transcripts in the "QB" corpus
participants <- getMatchingParticipantIds(
        labbcat.url, "labels('corpus').includes('QB')", 20, 0)

## Get all participants in the "QB" corpus that have "Jacqui" as part of the ID
participants <- getMatchingTranscriptParticipantIds(
        labbcat.url, "labels('corpus').includes('QB') && /^BR.+/.test(id)")

## End(Not run)
```

---

getMatchingTranscriptIds

Gets a list of IDs of transcripts that match a particular pattern

---

Description

Gets a list of IDs of transcripts (i.e. transcript names) that match a particular pattern.

Usage

```
getMatchingTranscriptIds(
  labbcat.url,
  expression,
  page.length = NULL,
  page.number = NULL,
  order = NULL
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
expression	An expression that determines which transcripts match
page.length	The maximum number of IDs to return, or null to return all
page.number	The zero-based page number to return, or null to return the first page
order	An expression that determines the order the transcripts are listed in - if specified, this must include the keyword 'ASC' for ascending or 'DESC' for descending order.

Details

The results can be exhaustive, by omitting page.length and page.number, or they can be a subset (a 'page') of results, by given page.length and page.number values.

The order of the list can be specified. If ommitted, the transcripts are listed in ID order.

The expression language is currently not well defined, but is based on JavaScript syntax.

- The *labels* function can be used to represent a list of all the annotation labels on a given layer. For example, each transcript can have multiple participants, so the participant labels (names) are represented by: `labels('participant')`
- Use the *includes* function on a list to test whether the list contains a given element. e.g. to match transcripts that include the participant 'Joe' use: `labels('participant').includes('Joe')`
- Use the *first* function to identify the first (or the only) annotation on a given layer. e.g. the annotation representing the transcript's corpus is: `first('corpus')`
- Single annotations have various attributes, including 'id', 'label', 'ordinal', etc. e.g. the name of the transcript's corpus is: `first('corpus').label`
- Regular expressions can be matched by using expressions like '/regex/.test(str)', e.g. to test if the ID starts with 'BR' use: `/^BR.+/.test(id)` or to test if the transcript's corpus includes a B use: `/.*B.*/.test(first('corpus').label)`

Expressions such as those in the examples can be used.

Value

A list of transcript IDs (i.e. transcript names)

Examples

```
## Not run:
## Get all transcripts whose names start with "BR"
transcripts <- getMatchingTranscriptIds(labbcat.url, "/^BR.+/.test(id)")

## Get the first twenty transcripts in the "QB" corpus
transcripts <- getMatchingTranscriptIds(
        labbcat.url, "first('corpus').label = 'QB'", 20, 0)

## Get the second transcript that has "QB247_Jacqui" as a speaker
transcripts <- getMatchingTranscriptIds(
        labbcat.url, "labels('participant').includes('QB247_Jacqui')", 1, 1)

## Get all transcripts in the QB corpus whose names start with "BR"
## in word-count order
transcripts <- getMatchingTranscriptIds(
        labbcat.url, "first('corpus').label = 'QB' && /^BR.+/.test(id)",
        order="first('transcript_word_count').label ASC")

## End(Not run)
```

`getMatchLabels` *Gets labels of annotations on a given layer, identified by given match IDs*

Description

Gets labels of annotations on a given layer, identified by given match IDs

Usage

```
getMatchLabels(
  labbcat.url,
  match.ids,
  layer.ids,
  target.offset = 0,
  annotations.per.layer = 1,
  include.match.ids = FALSE,
  page.length = 1000,
  no.progress = FALSE
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`match.ids`	A vector of annotation IDs, e.g. the MatchId column, or the URL column, of a results set.

`layer.ids` A vector of layer IDs.

`target.offset` The distance from the original target of the match, e.g.

- *0* – find annotations of the match target itself
- *1* – find annotations of the token immediately *after* match target
- *-1* – find annotations of the token immediately *before* match target

`annotations.per.layer`
The number of annotations on the given layer to retrieve. In most cases, there's only one annotation available. However, tokens may, for example, be annotated with 'all possible phonemic transcriptions', in which case using a value of greater than 1 for this parameter provides other phonemic transcriptions, for tokens that have more than one.

`include.match.ids`
Whether or not the data frame returned includes the original MatchId column or not.

`page.length` In order to prevent timeouts when there are a large number of matches or the network connection is slow, rather than retrieving matches in one big request, they are retrieved using many smaller requests. This parameter controls the number of results retrieved per request.

`no.progress` TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

A data frame of labels.

See Also

- getMatches
- getMatchAlignments

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(orthography="quake"))

## Get the topic annotations for the matches
topics <- getMatchLabels(labbcat.url, results$MatchId, "topic")

## End(Not run)
```

`getMedia` *Downloads a given media track for a given transcript*

Description

Downloads a given media track for a given transcript

Usage

```
getMedia(
  labbcat.url,
  id,
  track.suffix = "",
  mime.type = "audio/wav",
  path = ""
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance.
`id`	A transcript ID (i.e. transcript name).
`track.suffix`	The track suffix of the media.
`mime.type`	The MIME type of the media, e.g. "audio/wav" or "application/f0".
`path`	Optional path to directory where the file should be saved.

Value

The name of the file, which is saved in the current directory, or the given path if specified

See Also

- getTranscriptIds
- getMediaUrl

Examples

```
## Not run:
## Download the WAV file for BR2044_OllyOhlson.eaf
wav <- getMedia(labbcat.url, "BR2044_OllyOhlson.eaf")

## Download the 'QuakeFace' video file for BR2044_OllyOhlson.eaf
quakeFaceMp4 <- getMedia(labbcat.url, "BR2044_OllyOhlson.eaf", "_face", "video/mp4")

## End(Not run)
```

getMediaTracks *List the predefined media tracks available for transcripts*

Description

List the predefined media tracks available for transcripts

Usage

```
getMediaTracks(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A list of media track definitions.

Examples

```
## Not run:
## Get the media tracks configured in LaBB-CAT
tracks <- getMediaTracks("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

getMediaUrl *Gets the URL of the given media track for a given transcript*

Description

Gets the URL of the given media track for a given transcript

Usage

```
getMediaUrl(labbcat.url, id, track.suffix = "", mime.type = "audio/wav")
```

Arguments

labbcat.url	URL to the LaBB-CAT instance.
id	A transcript ID (i.e. transcript name).
track.suffix	The track suffix of the media.
mime.type	The MIME type of the media, e.g. "audio/wav" or "application/f0".

Value

A URL to the given media for the given transcript.

See Also

- getTranscriptIds
- getMedia

Examples

```
## Not run:
## Get URL for the WAV file for BR2044_OllyOhlson.eaf
wavUrl <- getMediaUrl(labbcat.url, "BR2044_OllyOhlson.eaf")

## Get URL for the 'QuakeFace' video file for BR2044_OllyOhlson.eaf
quakeFaceMp4Url <- getMediaUrl(labbcat.url, "BR2044_OllyOhlson.eaf", "_face", "video/mp4")

## End(Not run)
```

---

getParticipant *Gets information about a single participant*

Description

Returns a nested named list with the participant information, including the given participant attributes.

Usage

```
getParticipant(labbcat.url, id, layer.ids)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	A participant ID
layer.ids	A vector of layer IDs corresponding to participant attributes, eg. c('participant_gender', 'participant_year_of_birth')

Value

A named list of representing the participant and its attributes, with members:

- *id* The participant's unique internal database ID
- *label* The ID (name) of the participant
- *annotations* A named list of participant attributes e.g. the label of the participant's 'gender' attribute would be: participant$annotations$participant_gender$label

See Also

- getParticipantAttributes
- saveParticipant
- deleteParticipant

Examples

```
## Not run:
## Get the gender and year of birth of AP511_MikeThorpe
participant <- getParticipant(labbcat.url, "AP511_MikeThorpe",
                  c("participant_gender", "participant_year_of_birth"))

print(paste("ID:", participant$label,
            "Gender:", participant$annotations$participant_gender$label,
            "YOB:", participant$annotations$participant_year_of_birth$label))

## End(Not run)
```

`getParticipantAttributes` *Gets participant attribute values for given participant IDs*

Description

Gets participant attribute values for given participant IDs

Usage

```
getParticipantAttributes(labbcat.url, participant.ids, layer.ids)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`participant.ids`	A vector of participant IDs
`layer.ids`	A vector of layer IDs corresponding to participant attributes. In general, these are layers whose ID is prefixed 'participant_', however formally it's any layer where layer$parentId == 'participant' && layer$alignment == 0.

Value

A data frame of attribute value labels.

Examples

```
## Not run:
## Get gender and age for all participants
attributes <- getParticipantAttributes(labbcat.url,
            getParticipantIds(labbcat.url),
            c('participant_gender', 'participant_age'))


## End(Not run)
```

`getParticipantIds` *Gets a list of participant IDs*

Description

Returns a list of participant IDs.

Usage

```
getParticipantIds(labbcat.url)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance

Value

A list of participant IDs

Examples

```
## Not run:
## List all speakers
speakers <- getParticipantIds("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

`getSerializerDescriptors` *Lists the descriptors of all registered serializers*

Description

Returns a list of serializers, which are modules that export annotation structures as a specific file format, e.g. Praat TextGrid, plain text, etc., so the *mimeType* of descriptors reflects what *mimeType*s can be specified for getFragments.

Usage

```
getSerializerDescriptors(labbcat.url)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance

Value

A list of serializers, each including the following information:

- *name* The name of the format.
- *version* The installed version of the serializer module.
- *fileSuffixes* The normal file name suffixes (extensions) of the files.
- *mimeType* The MIME type of the format, i.e. the value to use as the *mimeType* parameter of getFragments

See Also

getFragments

Examples

```
## Not run:
## List file export formats supported
formats <- getSerializerDescriptors("https://labbcat.canterbury.ac.nz/demo/")

## can we export as plain text?
plainTextSupported <- "text/plain" %in% formats$mimeType

## End(Not run)
```

getSoundFragments *Gets sound fragments from 'LaBB-CAT'.*

Description

Gets sound fragments from 'LaBB-CAT'.

Usage

```
getSoundFragments(
  labbcat.url,
  ids,
  start.offsets,
  end.offsets,
  sample.rate = NULL,
  path = "",
  no.progress = FALSE
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
ids	The transcript ID (transcript name) of the sound recording, or a vector of transcript IDs.
start.offsets	The start time in seconds, or a vector of start times.
end.offsets	The end time in seconds, or a vector of end times.
sample.rate	Optional sample rate in Hz - if a positive integer, then the result is a mono file with the given sample rate.
path	Optional path to directory where the files should be saved.
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

The name of the file, which is saved in the current directory, or a list of names of files, if multiple id's/start's/end's were specified

If a list of files is returned, they are in the order that they were returned by the server, which *should* be the order that they were specified in the id/start/end lists.

Examples

```
## Not run:
## Get the 5 seconds starting from 10s after the beginning of a recording
wav.file <- getSoundFragments(labbcat.url, "AP2505_Nelson.eaf", 10.0, 15.0, path="samples")

## Get the 5 seconds starting from 10s as a mono 22kHz file
```

```
wav.file <- getSoundFragments(labbcat.url, "AP2505_Nelson.eaf", 10.0, 15.0, 22050)

## Load some search results previously exported from LaBB-CAT
results <- read.csv("results.csv", header=T)

## Get a list of fragments
wav.files <- getSoundFragments(labbcat.url, results$Transcript, results$Line, results$LineEnd)

## Get a list of fragments
wav.file <- getSoundFragments(
              labbcat.url, results$Transcript, results$Line, results$LineEnd)

## End(Not run)
```

`getSystemAttribute` *Gets the value of the given system attribute*

Description

Gets the value of the given system attribute

Usage

```
getSystemAttribute(labbcat.url, attribute)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`attribute`	Name of the attribute.

Value

The value of the given attribute.

See Also

getLayers

Examples

```
## Not run:
## Get the name of the LaBB-CAT instance
title <- getSystemAttribute("https://labbcat.canterbury.ac.nz/demo/", "title")

## End(Not run)
```

getTranscriptAttributes *Gets transcript attribute values for given transcript IDs*

Description

Gets transcript attribute values for given transcript IDs

Usage

```
getTranscriptAttributes(labbcat.url, transcript.ids, layer.ids)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
transcript.ids	A vector of transcript IDs
layer.ids	A vector of layer IDs corresponding to transcript attributes. In general, these are layers whose ID is prefixed 'transcript_', however formally it's any layer where layer$parentId == 'transcript' && layer$alignment == 0, which includes 'corpus' as well as transcript attribute layers.

Value

A data frame of attribute value labels.

Examples

```
## Not run:
## Get language, duration, and corpus for transcripts starting with 'BR'
attributes <- getTranscriptAttributes(labbcat.url,
            getMatchingTranscriptIds(labbcat.url, "/'BR.+'/.test(id)"),
            c('transcript_language', 'transcript_duration', 'corpus'))


## End(Not run)
```

getTranscriptIds *Gets a list of transcript IDs*

Description

Returns a list of transcript IDs (i.e. transcript names).

Usage

```
getTranscriptIds(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A list of transcript IDs

Examples

```
## Not run:
## List all transcripts
transcripts <- getTranscriptIds("https://labbcat.canterbury.ac.nz/demo/")

## End(Not run)
```

---

getTranscriptIdsInCorpus

Gets a list of transcript in a corpus

Description

Returns a list of transcript IDs in the given corpus.

Usage

```
getTranscriptIdsInCorpus(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The ID (name) of the corpus

Value

A list of transcript IDs

Examples

```
## Not run:
## List transcripts in the QB corpus
transcripts <- getTranscriptIdsInCorpus("https://labbcat.canterbury.ac.nz/demo/", "QB")

## End(Not run)
```

getTranscriptIdsWithParticipant *Gets a list of IDs of transcripts that include the given participant*

Description

Returns a list of IDs of transcripts (i.e. transcript names) that include the given participant.

Usage

```
getTranscriptIdsWithParticipant(labbcat.url, id)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	A participant ID

Value

A list of transcript IDs

See Also

getParticipantIds

Examples

```
## Not run:
## List transcripts in which UC427_ViktoriaPapp_A_ENG speaks
transcripts <- getTranscriptIdsWithParticipant(labbcat.url, "UC427_ViktoriaPapp_A_ENG")

## End(Not run)
```

getUserInfo *Gets information about the current user*

Description

Returns information about the current user, including the roles or groups they are in.

Usage

```
getUserInfo(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

A named list containing information about current the LaBB-CAT user.

See Also

labbcatCredentials

Examples

```
## Not run:
## List file export formats supported
me <- getUserInfo("https://labbcat.canterbury.ac.nz/demo/")

## am I an administrator?
admin <- "admin" %in% me$roles

## End(Not run)
```

---

labbcatCredentials *Sets the username and password for a given LaBB-CAT server*

Description

Sets the username and password that the package should use for connecting to a given LaBB-CAT server in future function calls.

Usage

```
labbcatCredentials(labbcat.url, username, password, auth.method = NULL)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance.
username	The LaBB-CAT username, if it is password-protected.
password	The LaBB-CAT password, if it is password-protected.
auth.method	The HTTP authentication method to use (e.g. "Basic" or "Form", or NULL to detect automatically.

Details

If you are using R interactively, this step is optional, as all functions will prompt the user for the username and password if required. If the script is running in RStudio, then the RStudio password input dialog is used, hiding the credentials from view. Otherwise, the console is used, and credentials are visible.

The recommended approach is to **not** use labbcatCredentials, to avoid saving user credentials in script files that may eventually become visible to other. Use labbcatCredentials **only** in cases where the script execution is unsupervised, e.g. if you are executing an R script from a shell script, or using Knit to render an Rmarkdown document.

If you must use labbcatCredentials, avoid including the actual username and password in your script. The recommended approach is to store the username and password (and perhaps the URL too) in your '.Renviron' file (in your home directory, or the project directory), like this:

```
LABBCAT_URL=https://labbcat.canterbury.ac.nz/demo/
LABBCAT_USERNAME=demo
LABBCAT_PASSWORD=demo
```

And then call Sys.getenv to retrieve the username/password, as illustrated in the example.

Value

NULL if the username/password are correct, and a string describing the problem if a problem occurred, e.g. "Credentials rejected" if the username/password are incorrect, or a string starting "Version mismatch" if the server's version of LaBB-CAT is lower than the minimum required.

Examples

```
## Not run:
## load the LaBB-CAT URL from .Renviron
labbcat.url <- Sys.getenv('LABBCAT_URL')

## load the username/password from .Renviron so secrets are not included in the script:
labbcatCredentials(
    labbcat.url, Sys.getenv('LABBCAT_USERNAME'), Sys.getenv('LABBCAT_PASSWORD'))

## End(Not run)
```

---

labbcatTimeout *Sets the timeout for request to the LaBB-CAT server in future function calls. The default timeout is 10 seconds*

Description

Sets the timeout for request to the LaBB-CAT server in future function calls. The default timeout is 10 seconds

Usage

```
labbcatTimeout(seconds = NULL)
```

Arguments

seconds	The number of seconds before requests return with a timeout error.

Value

The request timeout in seconds

Examples

```
## Not run:
## the request timeout
labbcatTimeout(30)

## End(Not run)
```

---

labbcatVersionInfo *Gets version information of all components of LaBB-CAT*

Description

Version information includes versions of all components and modules installed on the LaBB-CAT server, including format converters and annotator modules.

Usage

```
labbcatVersionInfo(labbcat.url)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance

Value

The versions of different components of LaBB-CAT, divided into sections:

- *System* Overall LaBB-CAT system components
- *Formats* Annotation format conversion modules
- *Layer Managers* Annotator module versions
- *3rd Pary Software* Versions of software installed on the server that LaBB-CAT integrates with, e.g. Praat, FastTrack, etc.
- *RDBMS* MySQL Server version information

Examples

```
## Not run:
## Get ID of LaBB-CAT instance
versionInfo <- labbcatVersionInfo("https://labbcat.canterbury.ac.nz/demo/")
print(paste("LaBB-CAT version", versionInfo$System$'LaBB-CAT', " Full version info:"))
print(t(as.data.frame(versionInfo)))

## End(Not run)
```

`loadLexicon` *Upload a flat lexicon file for lexical tagging*

Description

By default LaBB-CAT includes a layer manager called the Flat Lexicon Tagger, which can be configured to annotate words with data from a dictionary loaded from a plain text file (e.g. a CSV file). The file must have a 'flat' structure in the sense that it's a simple list of dictionary entries with a fixed number of columns/fields, rather than having a complex structure.

Usage

```
loadLexicon(
  labbcat.url,
  file,
  lexicon,
  field.delimiter,
  field.names,
  quote = "",
  comment = "",
  skip.first.line = FALSE,
  no.progress = FALSE
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance.
`file`	The full path name of the lexicon file.
`lexicon`	The name for the resulting lexicon. If the named lexicon already exists, it will be completely replaced with the contents of the file (i.e. all existing entries will be deleted befor adding new entries from the file). e.g. 'cmudict'
`field.delimiter`	The character used to delimit fields in the file. If this is " - ", rows are split on only the first space, in line with common dictionary formats. e.g. ',' for Comma Separated Values (CSV) files.
`field.names`	A list of field names, delimited by field.delimiter, e.g. 'Word,Pronunciation'.

quote	The character used to quote field values (if any), e.g. '"'.
comment	The character used to indicate a line is a comment (not an entry) (if any) e.g. '#'.
skip.first.line	Whether to ignore the first line of the file (because it contains field names).
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Details

This function uploads such a lexicon file, for use in tagging tokens.

You must have editing privileges in LaBB-CAT in order to be able to use this function.

Value

An error message, or NULL if the upload was successful.

See Also

Other dictionary functions: addDictionaryEntry(), addLayerDictionaryEntry(), deleteLexicon(), getDictionaries(), getDictionaryEntries(), removeDictionaryEntry(), removeLayerDictionaryEntry()

Examples

```
## Not run:
## Upload the CMU Pronouncing Dictionary
loadLexicon(labbcat.url, "cmudict", " - ", "", ";", "Word - Pron", FALSE, "cmudict.txt")

## End(Not run)
```

---

newLayer *Creates a new layer*

Description

This function creates a new annotation layer. The layer may be configured with a layer manager ID and task parameters, for automatic annotation. If so, this function will create the layer and the automation task, but automatic annotation will not be run by this function. To generate the automatic annotations, use generateLayer.

Usage

```
newLayer(
  labbcat.url,
  layer.id,
  description,
  type = "string",
  alignment = 0,
  category = "General",
  parent.id = "word",
  annotator.id = NULL,
  annotator.task.parameters = NULL
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
layer.id	The ID of the layer to create, which must be unique to the LaBB-CAT instance.
description	A description of the annotations the layer will contain.
type	The type of data the labels will represent. Valid values are "string", "number", "ipa" (for phoneme representations), or "boolean" (labels "0" or "1").
alignment	How annotations on the layer will relate to time alignment; valid values are 0 (no alignment; annototations are just tags on the parent annotation), 1 (instants; annotations mark a single point in time), or 2 (intervals; annotations have a start and end time).
category	The project/category the layer belongs to.
parent.id	The parent layer; valid values are "word" (for word layers), "segment" (for segment layers) "turn" (for phrase layers), or "transcript" (for span layers).
annotator.id	The ID of the layer manager that automatically fills in annotations on the layer, if any
annotator.task.parameters	The configuration the layer manager should use when filling the layer with annotations. This is a string whose format is specific to each layer manager.

Details

You must have administration privileges in LaBB-CAT in order to be able to use this function.

Value

The resulting layer definition, with members:

- *id* The layer's unique ID
- *parentId* The layer's parent layer ID
- *description* The description of the layer
- *alignment* The layer's alignment - 0 for none, 1 for point alignment, 2 for interval alignment

- *peers* Whether children have peers or not
- *peersOverlap* Whether child peers can overlap or not
- *parentIncludes* Whether the parent t-includes the child
- *saturated* Whether children must temporally fill the entire parent duration (true) or not (false)
- *parentIncludes* Whether the parent t-includes the child
- *type* The type for labels on this layer
- *validLabels* List of valid label values for this layer

See Also

Other Annotation layer functions: deleteLayer(), generateLayer(), getLayer(), getLayerIds(), getLayers(), saveLayer()

Examples

```
## Not run:
## Upload the CMU Pronouncing Dictionary
loadLexicon(labbcat.url, "cmudict", " - ", "", ";", "Word - Pron", FALSE, "cmudict.txt")

## Create a layer that tags each token with its CMU Pronouncing Dictionary pronunciation
newLayer(labbcat.url, "pronunciation", "CMU Dict pronunciations encoded in ARPAbet",
         annotator.id="FlatFileDictionary",
         annotator.task.parameters=
           "tokenLayerId=orthography&tagLayerId=phonemes&dictionary=cmudict:Word->Pron")

## Generate the pronunciation tags
generateLayer(labbcat.url, "pronunciation")

## End(Not run)
```

---

newTranscript *Upload a new transcript*

Description

This function adds a transcript and optionally a media file to the corpus.

Usage

```
newTranscript(
  labbcat.url,
  transcript,
  media = NULL,
  transcript.type = NULL,
  corpus = NULL,
  episode = NULL,
  no.progress = FALSE
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
transcript	The path to the transcript to upload.
media	The path to the media to upload, if any.
transcript.type	The transcript type.
corpus	The corpus to add the transcript to.
episode	The transcript's episode.
no.progress	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Details

NB This method of uploading is an alternative to using transcriptUpload and transcriptUploadParameters.

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

Value

The ID of the new transcript in the corpus

See Also

- transcriptUpload
- transcriptUploadParameters
- transcriptUploadDelete
- updateTranscript

Examples

```
## Not run:
## Get attributes for new transcript
corpus <- getCorpusIds(labbcat.url)[1]
transcript.type.layer <- getLayer(labbcat.url, "transcript_type")
transcript.type <- transcript.type.layer$validLabels[[1]]

## upload transcript
newTranscript(
    labbcat.url, "my-transcript.eaf", "my-transcript.wav",
    "", transcript.type, corpus, "episode-1")

## End(Not run)
```

praatScriptCentreOfGravity *Generates a script for extracting the CoG, for use with processWithPraat*

Description

This function generates a Praat script fragment which can be passed as the praat.script parameter of processWithPraat, in order to extract one or more spectral centre of gravity (CoG) measurements.

Usage

```
praatScriptCentreOfGravity(powers = c(2), spectrum.fast = TRUE)
```

Arguments

powers	A vector of numbers specifying which powers to query for to extract, e.g. c(1.0,2.0).
spectrum.fast	Whether to use the 'fast' option when creating the spectrum object to query.

Value

A script fragment which can be passed as the praat.script parameter of processWithPraat

See Also

Other Praat-related functions: appendFromPraat(), fragmentLabels(), fragmentTranscripts(), praatScriptFastTrack(), praatScriptFormants(), praatScriptIntensity(), praatScriptPitch(), processWithPraat()

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(segment="I"))

## Get centres of gravity for all matches
cog <- processWithPraat(
             labbcat.url,
             results$MatchId, results$Target.segment.start, results$Target.segment.end,
             praatScriptCentreOfGravity(powers=c(1.0,2.0)))

## End(Not run)
```

praatScriptFastTrack *Generates a script for extracting formants using FastTrack, for use with processWithPraat*

Description

This function generates a Praat script fragment which can be passed as the praat.script parameter of processWithPraat, in order to extract selected formants using the FastTrack Praat plugin.

Usage

```
praatScriptFastTrack(
  formants = c(1, 2),
  sample.points = c(0.5),
  lowest.analysis.frequency = 5000,
  lowest.analysis.frequency.male = 4500,
  highest.analysis.frequency = 7000,
  highest.analysis.frequency.male = 6500,
  gender.attribute = "participant_gender",
  value.for.male = "M",
  time.step = 0.002,
  tracking.method = "burg",
  number.of.formants = 3,
  maximum.f1.frequency = 1200,
  maximum.f1.bandwidth = NULL,
  maximum.f2.bandwidth = NULL,
  maximum.f3.bandwidth = NULL,
  minimum.f4.frequency = 2900,
  enable.rhotic.heuristic = TRUE,
  enable.f3.f4.proximity.heuristic = TRUE,
  number.of.steps = 20,
  number.of.coefficients = 5
)
```

Arguments

formants	A vector of integers specifying which formants to extract, e.g c(1,2) for the first and second formant.
sample.points	A vector of numbers (0 <= sample.points <= 1) specifying multiple points at which to take the measurement. The default is a single point at 0.5 - this means one measurement will be taken halfway through the target interval. If, for example, you wanted eleven measurements evenly spaced throughout the interval, you would specify sample.points as being c(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0).
lowest.analysis.frequency	Lowest analysis frequency (Hz) by default.

`lowest.analysis.frequency.male`
: Lowest analysis frequency (Hz) for male speakers, or NULL to use the same value as lowest.analysis.frequency.

`highest.analysis.frequency`
: Highest analysis frequency (Hz) by default.

`highest.analysis.frequency.male`
: Highest analysis frequency (Hz) for male speakers, or NULL to use the same value as highest.analysis.frequency.

`gender.attribute`
: Name of the LaBB-CAT participant attribute that contains the participant's gender - normally this is "participant_gender".

`value.for.male` The value that the gender.attribute has when the participant is male.

`time.step` Time step in seconds.

`tracking.method`
: tracking_method parameter for trackAutoselectProcedure; "burg" or "robust".

`number.of.formants`
: Number of formants to track - 3 or 4.

`maximum.f1.frequency`
: Specifying a non-NULL value enables the F1 frequency heuristic: Median F1 frequency should not be higher than this value.

`maximum.f1.bandwidth`
: Specifying a non-NULL value (e.g. 500) enables the F1 bandwidth heuristic: Median F1 bandwidth should not be higher than this value.

`maximum.f2.bandwidth`
: Specifying a non-NULL value (e.g. 600) enables the F2 bandwidth heuristic: Median F2 bandwidth should not be higher than this value.

`maximum.f3.bandwidth`
: Specifying a non-NULL value (e.g. 900) enables the F3 bandwidth heuristic: Median F3 bandwidth should not be higher than this value.

`minimum.f4.frequency`
: Specifying a non-NULL value enables the F4 frequency heuristic: Median F4 frequency should not be lower than this value.

`enable.rhotic.heuristic`
: Whether to enable the rhotic heuristic: If F3 < 2000 Hz, F1 and F2 should be at least 500 Hz apart.

`enable.f3.f4.proximity.heuristic`
: Whether to enable the F3/F4 proximity heuristic: If (F4 - F3) < 500 Hz, F1 and F2 should be at least 1500 Hz apart.

`number.of.steps`
: Number of analyses between low and high analysis limits. More analysis steps may improve results, but will increase analysis time (50 percent more steps = around 50 percent longer to analyze).

`number.of.coefficients`
: Number of coefficients for formant prediction. More coefficients allow for more sudden, and 'wiggly' formant motion.

Details

The FastTrack Praat plugin, developed by Santiago Barreda, automatically runs multiple formant analyses on each segment, selects the best (the smoothest, with optional heuristics), and makes the winning formant object available for measurement. For more information, see https://github.com/santiagobarreda/FastTrack

Value

A script fragment which can be passed as the praat.script parameter of processWithPraat

See Also

Other Praat-related functions: appendFromPraat(), fragmentLabels(), fragmentTranscripts(), praatScriptCentreOfGravity(), praatScriptFormants(), praatScriptIntensity(), praatScriptPitch(), processWithPraat()

Examples

```
## Not run:
## Get all tokens of the KIT vowel
results <- getMatches(labbcat.url, list(segment="I"))

## Get the first 3 formants at three points during the vowel
formants <- processWithPraat(
              labbcat.url,
              results$MatchId, results$Target.segment.start, results$Target.segment.end,
              window.offset=0.025,
              praatScriptFastTrack(formants=c(1,2,3),
              sample.points=c(0.25,0.5,0.75)))

## End(Not run)
```

---

praatScriptFormants *Generates a script for extracting formants, for use with processWithPraat*

Description

This function generates a Praat script fragment which can be passed as the praat.script parameter of processWithPraat, in order to extract selected formants.

Usage

```
praatScriptFormants(
  formants = c(1, 2),
  sample.points = c(0.5),
  time.step = 0,
  max.number.formants = 5,
```

```
  max.formant = 5500,
  max.formant.male = 5000,
  gender.attribute = "participant_gender",
  value.for.male = "M",
  window.length = 0.025,
  preemphasis.from = 50
)
```

Arguments

formants	A vector of integers specifying which formants to extract, e.g c(1,2) for the first and second formant.
sample.points	A vector of numbers (0 <= sample.points <= 1) specifying multiple points at which to take the measurement. The default is a single point at 0.5 - this means one measurement will be taken halfway through the target interval. If, for example, you wanted eleven measurements evenly spaced throughout the interval, you would specify sample.points as being c(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0).
time.step	Time step in seconds, or 0.0 for 'auto'.
max.number.formants	Maximum number of formants.
max.formant	Maximum formant value (Hz) for all speakers, or for female speakers, if max.formant.male is also specified.
max.formant.male	Maximum formant value (Hz) for male speakers, or NULL to use the same value as max.formant.
gender.attribute	Name of the LaBB-CAT participant attribute that contains the participant's gender - normally this is "participant_gender".
value.for.male	The value that the gender.attribute has when the participant is male.
window.length	Window length in seconds.
preemphasis.from	Pre-emphasis from (Hz)

Details

The praatScriptFastTrack function provides an alternative to this function which uses the FastTrack Praat plugin for formant analysis.

Value

A script fragment which can be passed as the praat.script parameter of processWithPraat

See Also

Other Praat-related functions: `appendFromPraat()`, `fragmentLabels()`, `fragmentTranscripts()`, `praatScriptCentreOfGravity()`, `praatScriptFastTrack()`, `praatScriptIntensity()`, `praatScriptPitch()`, `processWithPraat()`

Examples

```
## Not run:
## Get all tokens of the KIT vowel
results <- getMatches(labbcat.url, list(segment="I"))

## Get the first 3 formants at three points during the vowel
formants <- processWithPraat(
              labbcat.url,
              results$MatchId, results$Target.segment.start, results$Target.segment.end,
              window.offset=0.025,
              praatScriptFormants(formants=c(1,2,3),
              sample.points=c(0.25,0.5,0.75)))

## End(Not run)
```

`praatScriptIntensity` *Generates a script for extracting maximum intensity, for use with processWithPraat*

Description

This function generates a Praat script fragment which can be passed as the praat.script parameter of processWithPraat, in order to extract maximum intensity value.

Usage

```
praatScriptIntensity(
  minimum.pitch = 100,
  time.step = 0,
  subtract.mean = TRUE,
  get.maximum = TRUE,
  sample.points = NULL,
  interpolation = "cubic",
  skip.errors = TRUE
)
```

Arguments

`minimum.pitch`	Minimum pitch (Hz).
`time.step`	Time step in seconds, or 0.0 for 'auto'.
`subtract.mean`	Whether to subtract the mean or not.
`get.maximum`	Extract the maximum intensity for the sample.
`sample.points`	A vector of numbers (0 <= sample.points <= 1) specifying multiple points at which to take the measurement. The default is NULL, meaning no individual measurements will be taken (only the aggregate values identified by get.mean, get.minimum, and get.maximum). A single point at 0.5 means one measurement will be taken halfway through the target interval. If, for example, you wanted

eleven measurements evenly spaced throughout the interval, you would specify sample.points as being c(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0).

`interpolation` If sample.points are specified, this is the interpolation to use when getting individual values. Possible values are 'nearest', 'linear', 'cubic', 'sinc70', or 'sinc700'.

`skip.errors` Sometimes, for some segments, Praat fails to create an Intensity object. If skip.errors = TRUE, analysis those segments will be skipped, and corresponding pitch values will be returned as "–undefined–". If skip.errors = FALSE, the error message from Praat will be returned in the Error field, but no pitch measures will be returned for any segments in the same recording.

Value

A script fragment which can be passed as the praat.script parameter of processWithPraat

See Also

Other Praat-related functions: `appendFromPraat()`, `fragmentLabels()`, `fragmentTranscripts()`, `praatScriptCentreOfGravity()`, `praatScriptFastTrack()`, `praatScriptFormants()`, `praatScriptPitch()`, `processWithPraat()`

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(segment="s"))

## Get max intensity, and intensity at three points during the segment, for all matches
intensity <- processWithPraat(
              labbcat.url,
              results$MatchId, results$Target.segment.start, results$Target.segment.end,
              praatScriptIntensity(sample.points = c(.25, .5, .75)))

## End(Not run)
```

---

praatScriptPitch *Generates a script for extracting pitch, for use with processWithPraat*

Description

This function generates a Praat script fragment which can be passed as the praat.script parameter of processWithPraat, in order to extract pitch information.

Usage

```
praatScriptPitch(
  get.mean = TRUE,
  get.minimum = FALSE,
  get.maximum = FALSE,
  time.step = 0,
  pitch.floor = 60,
  max.number.of.candidates = 15,
  very.accurate = FALSE,
  silence.threshold = 0.03,
  voicing.threshold = 0.5,
  octave.cost = 0.01,
  octave.jump.cost = 0.35,
  voiced.unvoiced.cost = 0.35,
  pitch.ceiling = 500,
  pitch.floor.male = 30,
  voicing.threshold.male = 0.4,
  pitch.ceiling.male = 250,
  gender.attribute = "participant_gender",
  value.for.male = "M",
  sample.points = NULL,
  interpolation = "linear",
  skip.errors = TRUE
)
```

Arguments

get.mean	Extract the mean pitch for the sample.
get.minimum	Extract the minimum pitch for the sample.
get.maximum	Extract the maximum pitch for the sample.
time.step	Step setting for praat command
pitch.floor	Minimum pitch (Hz) for all speakers, or for female speakers, if pitch.floor.male is also specified.
max.number.of.candidates	Maximum number of candidates setting for praat command
very.accurate	Accuracy setting for praat command
silence.threshold	Silence threshold setting for praat command
voicing.threshold	Voicing threshold (Hz) for all speakers, or for female speakers, if voicing.threshold.male is also specified.
octave.cost	Octave cost setting for praat command
octave.jump.cost	Octave jump cost setting for praat command
voiced.unvoiced.cost	Voiced/unvoiced cost setting for praat command

`pitch.ceiling` Maximum pitch (Hz) for all speakers, or for female speakers, if pitch.floor.male is also specified.

`pitch.floor.male` Minimum pitch (Hz) for male speakers.

`voicing.threshold.male` Voicing threshold (Hz) for male speakers.

`pitch.ceiling.male` Maximum pitch (Hz) for male speakers.

`gender.attribute` Name of the LaBB-CAT participant attribute that contains the participant's gender - normally this is "participant_gender".

`value.for.male` The value that the gender.attribute has when the participant is male.

`sample.points` A vector of numbers (0 <= sample.points <= 1) specifying multiple points at which to take the measurement. The default is NULL, meaning no individual measurements will be taken (only the aggregate values identified by get.mean, get.minimum, and get.maximum). A single point at 0.5 means one measurement will be taken halfway through the target interval. If, for example, you wanted eleven measurements evenly spaced throughout the interval, you would specify sample.points as being c(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0).

`interpolation` If sample.points are specified, this is the interpolation to use when getting individual values. Possible values are 'nearest' or 'linear'.

`skip.errors` Sometimes, for some segments, Praat fails to create a Pitch object. If skip.errors = TRUE, analysis those segments will be skipped, and corresponding pitch values will be returned as "–undefined–". If skip.errors = FALSE, the error message from Praat will be returned in the Error field, but no pitch measures will be returned for any segments in the same recording.

Value

A script fragment which can be passed as the praat.script parameter of processWithPraat

See Also

Other Praat-related functions: `appendFromPraat()`, `fragmentLabels()`, `fragmentTranscripts()`, `praatScriptCentreOfGravity()`, `praatScriptFastTrack()`, `praatScriptFormants()`, `praatScriptIntensity()`, `processWithPraat()`

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(segment="I"))

## Get pitch mean, max, and min, and the midpoint of the segment, for each match
pitch <- processWithPraat(
             labbcat.url,
             results$MatchId, results$Target.segment.start, results$Target.segment.end,
             praatScriptPitch(get.mean=TRUE, get.minimum=TRUE, get.maximum=TRUE,
```

```
                                    sample.points = c(.5)))

## End(Not run)
```

`processWithPraat` *Process a set of intervals with Praat*

Description

This function instructs the LaBB-CAT server to invoke Praat for a set of sound intervals, in order to extract acoustic measures.

Usage

```
processWithPraat(
  labbcat.url,
  match.ids,
  start.offsets,
  end.offsets,
  praat.script,
  window.offset,
  gender.attribute = "participant_gender",
  attributes = NULL,
  no.progress = FALSE
)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`match.ids`	A vector of annotation IDs, e.g. the MatchId column, or the URL column, of a results set.
`start.offsets`	The start time in seconds, or a vector of start times.
`end.offsets`	The end time in seconds, or a vector of end times.
`praat.script`	Script to run on each match. This may be a single string or a character vector.
`window.offset`	In many circumstances, you will want some context before and after the sample start/end time. For this reason, you can specify a "window offset" - this is a number of seconds to subtract from the sample start and add to the sample end time, before extracting that part of the audio for processing. For example, if the sample starts at 2.0s and ends at 3.0s, and you set the window offset to 0.5s, then Praat will extract a sample of audio from 1.5s to 3.5s, and do the selected processing on that sample. The best value for this depends on what the praat.script is doing; if you are getting formants from vowels, including some context ensures that the formants at the edges are more accurate (in LaBB-CAT's web interface, the default value for this 0.025), but if you're getting max pitch or COG during a segment, most likely you want a window.offset of 0 to ensure neighbouring segments don't influence the measurement.

`gender.attribute`
: Which participant attribute represents the participant's gender.

`attributes`
: Vector of participant attributes to make available to the script. For example, if you want to use different acoustic parameters depending on what the gender of the speaker is, including the "participant_gender" attribute will make a variable called participant_gender$ available to the praat script, whose value will be the gender of the speaker of that segment.

`no.progress`
: TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Details

The exact measurements to return depend on the praat.script that is invoked. This is a Praat script fragment that will run once for each sound interval specified.

There are functions to allow the generation of a number of pre-defined praat scripts for common tasks such as formant, pitch, intensity, and centre of gravity – see praatScriptFormants, praatScriptCentreOfGravity, praatScriptIntensity and praatScriptPitch.

You can provide your own script, either by building a string with your code, or loading one from a file.

LaBB-CAT prefixes praat.script with code to open a sound file and extract a defined part of it into a Sound object which is then selected.

LaBB-CAT `Remove's this Sound object after the script finishes executing. Any other objects created by t`move'd before the end of the script (otherwise Praat runs out of memory during very large batches)

LaBB-CAT assumes that all calls to the function 'print' correspond to fields for export and each field must be printed on its own line. Specifically it scans for lines of the form:

print 'myOutputVariable' 'newline$'

Variables that can be assumed to be already set in the context of the script are:

- *windowOffset* – the value used for the Window Offset; how much context to include.
- *windowAbsoluteStart* – the start time of the window extracted relative to the start of the original audio file.
- *windowAbsoluteEnd* – the end time of the window extracted relative to the start of the original audio file.
- *windowDuration* – the duration of the window extracted (including window offset).
- *targetAbsoluteStart* – the start time of the target interval relative to the start of the original audio file.
- *targetAbsoluteEnd* – the end time of the target interval relative to the start of the original audio file.
- *targetStart* – the start time of the target interval relative to the start of the window extracted.
- *targetEnd* – the end time of the target interval relative to the start of the window extracted.
- *targetDuration* – the duration of the target interval.
- *sampleNumber* – the number of the sample within the set of samples being processed.
- *sampleName$* – the name of the extracted/selected Sound object.

Value

A data frame of acoustic measures, one row for each matchId.

See Also

Other Praat-related functions: appendFromPraat(), fragmentLabels(), fragmentTranscripts(), praatScriptCentreOfGravity(), praatScriptFastTrack(), praatScriptFormants(), praatScriptIntensity(), praatScriptPitch()

Examples

```
## Not run:
## Perform a search
results <- getMatches(labbcat.url, list(segment="I"))

## get F1 and F2 for the mid point of the vowel
formants <- processWithPraat(
       labbcat.url,
       results$MatchId, results$Target.segment.start, results$Target.segment.end,
       praatScriptFormants())

## get first 3 formants at three points during the sample, the mean, min, and max
## pitch, the max intensity, and the CoG using powers 1 and 2
acoustic.measurements <- processWithPraat(
       labbcat.url,
       results$MatchId, results$Target.segment.start, results$Target.segment.end,
       paste(
           praatScriptFormants(c(1,2,3), c(0.25,0.5,0.75)),
           praatScriptPitch(get.mean=TRUE, get.minimum=TRUE, get.maximum=TRUE),
           praatScriptIntensity(),
           praatScriptCentreOfGravity(powers=c(1.0,2.0))),
       window.offset=0.5)

## execute a custom script loaded form a file
acoustic.measurements <- processWithPraat(
       labbcat.url,
       results$MatchId, results$Target.segment.start, result$Target.segment.end,
       readLines("acousticMeasurements.praat"))

## End(Not run)
```

---

removeDictionaryEntry *Removes an entry from a dictionary*

Description

This function removes an existing entry from the given dictionary.

Usage

```
removeDictionaryEntry(
  labbcat.url,
  manager.id,
  dictionary.id,
  key,
  entry = NULL
)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
manager.id	The layer manager ID of the dictionary, as returned by getDictionaries
dictionary.id	The ID of the dictionary, as returned by getDictionaries
key	The key (word) in the dictionary to remove an entry for.
entry	The value (definition) for the given key, or NULL to remove all entries for the key.

Details

You must have edit privileges in LaBB-CAT in order to be able to use this function.

Value

NULL if the entry was removed, or a list of error messages if not.

See Also

Other dictionary functions: addDictionaryEntry(), addLayerDictionaryEntry(), deleteLexicon(), getDictionaries(), getDictionaryEntries(), loadLexicon(), removeLayerDictionaryEntry()

Examples

```
## Not run:
## Remove a pronuciation of the word "robert" from the CELEX wordform pronunciation dictionary
removeDictionaryEntry(labbcat.url, "CELEX-EN", "Phonology (wordform)", "robert", "'rQ-bErt")

## End(Not run)
```

`removeLayerDictionaryEntry` *Removes an entry from a layer dictionary*

Description

This function removes an existing entry from the dictionary that manages a given layer, and updates all affected tokens in the corpus. Words can have multiple entries.

Usage

```
removeLayerDictionaryEntry(labbcat.url, layer.id, key, entry = NULL)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`layer.id`	The ID of the layer with a dictionary configured to manage it.
`key`	The key (word) in the dictionary to remove an entry from.
`entry`	The value (definition) for the given key, or NULL to remove all entries for the given key.

Details

You must have edit privileges in LaBB-CAT in order to be able to use this function.

Value

NULL if the entry was added, or a list of error messages if not.

See Also

Other dictionary functions: `addDictionaryEntry()`, `addLayerDictionaryEntry()`, `deleteLexicon()`, `getDictionaries()`, `getDictionaryEntries()`, `loadLexicon()`, `removeDictionaryEntry()`

Examples

```
## Not run:
## Remove a pronunciation for "robert" from the phonemes layer dictionary
removeLayerDictionaryEntry(labbcat.url, "phonemes", "robert", "'rQ-bErt")

## End(Not run)
```

`renameParticipants` *Renames a list of participants*

Description

This function changes the IDs of a given set of participants, where possible.

Usage

```
renameParticipants(labbcat.url, current.ids, new.ids, no.progress = FALSE)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`current.ids`	A vector of participant IDs that as they are currently defined in the corpus.
`new.ids`	A vector of new participant IDs, each element corresponding to an ID in current.ids.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Value

A vector of results, each element corresponding to an ID in current.ids. If the ID was successfully changed, the corresponding element is TRUE. If the ID could not be changed (e.g. because there is already an existing participant using the new ID), then the corresponding element is FALSE.

See Also

- getParticipantIds
- getMatchingParticipantIds
- getParticipant
- saveParticipant
- deleteParticipant

Examples

```
## Not run:
## Create some new participant records
old.ids <- c("test-id-1","test-id-2","test-id-3")
for (id in old.ids) saveParticipant(labbcat.url, id)

## Batch change the IDs
new.ids <- c("test-id-1-changed","test-id-2-changed","test-id-3-changed")
renameParticipants(labbcat.url, old.ids, new.ids)

## Delete the participants we just created
```

```
for (id in new.ids) deleteParticipant(labbcat.url, id)

## End(Not run)
```

`saveLayer` *Saves the details of an existing layer*

Description

This function saves the definition of an existing annotation layer.

Usage

```
saveLayer(labbcat.url, layer)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`layer`	A named list object representing the layer attributes, as would be returned by getLayer or newLayer, with members: • *id* The layer's unique ID • *parentId* The layer's parent layer ID • *description* The description of the layer • *alignment* The layer's alignment - 0 for none, 1 for point alignment, 2 for interval alignment • *peers* Whether children have peers or not • *peersOverlap* Whether child peers can overlap or not • *parentIncludes* Whether the parent t-includes the child • *saturated* Whether children must temporally fill the entire parent duration (true) or not (false) • *parentIncludes* Whether the parent t-includes the child • *type* The type for labels on this layer • *validLabels* List of valid label values for this layer

Details

You must have administration privileges in LaBB-CAT in order to be able to use this function.

Value

The resulting layer definition, with members:

- *id* The layer's unique ID
- *parentId* The layer's parent layer ID

- *description* The description of the layer
- *alignment* The layer's alignment - 0 for none, 1 for point alignment, 2 for interval alignment
- *peers* Whether children have peers or not
- *peersOverlap* Whether child peers can overlap or not
- *parentIncludes* Whether the parent t-includes the child
- *saturated* Whether children must temporally fill the entire parent duration (true) or not (false)
- *parentIncludes* Whether the parent t-includes the child
- *type* The type for labels on this layer
- *validLabels* List of valid label values for this layer

See Also

Other Annotation layer functions: `deleteLayer()`, `generateLayer()`, `getLayer()`, `getLayerIds()`, `getLayers()`, `newLayer()`

Examples

```
## Not run:
## Get the pronunciation layer definition
pronunciation <- getLayer(labbcat.url, "pronunciation")

## Change some details of the definition
pronunciation$description <- "CMU Dict pronunciations encoded in DISC"
pronunciation$type <- "ipa"

## Save the changes to the layer definition
saveLayer(labbcat.url, pronunciation)

## End(Not run)
```

---

saveMedia *Uploads the given media for the given transcript*

Description

This function upload a media file to LaBB-CAT, associating it with a given transcript.

Usage

```
saveMedia(labbcat.url, id, media, track.suffix = NULL)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	The transcript ID.
`media`	The path to the media to upload.
`track.suffix`	The track suffix for the media, if any.

Details

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

Value

A named list describing the attributes of the uploaded media:

- *trackSuffix* The track suffix of the media
- *mimeType* The MIME type of the file
- *url* URL to the content of the file
- *name* Name of the file in LaBB-CAT

See Also

- getAvailableMedia
- deleteMedia

Examples

```
## Not run:

## upload transcript
saveMedia(
    labbcat.url, "my-transcript.eaf", "my-transcript/audio/room-mic.wav", "-room")

## End(Not run)
```

---

saveParticipant *Saves information about a single participant*

Description

This function allows the participant attributes and the ID of a given participant to be updated.

Usage

```
saveParticipant(labbcat.url, id, label = id, attributes = NULL)
```

Arguments

labbcat.url	URL to the LaBB-CAT instance
id	The participant ID - either the unique internal database ID, or their name.
label	The new ID (name) for the participant
attributes	A named list of participant attribute values - the names are the participant attribute layer IDs, and the values are the corresponding new attribute values. The pass phrase for participant access can also be set by specifying a "_password" attribute.

Details

To change the ID of an existing participant, pass the old/current ID as the `id`, and pass the new ID as the `label`.

If the participant ID does not already exist in the database, a new participant record is created.

Value

TRUE if the participant's record was updated, FALSE if there were no changes detected.

See Also

- getParticipant
- deleteParticipant

Examples

```
## Not run:
## Create a new participant record
saveParticipant(labbcat.url, "Juan Perez", attributes=list(participant_gender="M"))

## Change the name and the gender of the participant record
saveParticipant(labbcat.url, "Juan Perez", "Maria Perez", list(participant_gender="F"))

### Delete the participant we just created
deleteParticipant(labbcat.url, "Maria Perez")

## End(Not run)
```

`transcriptUpload` *Upload a transcript file and associated media files.*

Description

Uploading files is the first stage in adding or modifying a transcript to LaBB-CAT. The second stage is transcriptUploadParameters()

Usage

```
transcriptUpload(labbcat.url, transcript, media = NULL, merge = FALSE)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`transcript`	The path to the transcript to upload.
`media`	The path to the media to upload, if any.
`merge`	Whether the upload corresponds to updates to an existing transcript (TRUE) or a new transcript (FALSE).

Details

NB Using transcriptUpload and transcriptUploadParameters is an alternative to using newTranscript or updateTranscript.

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

Value

A named list with members:

- *id* The unique identifier to use for this upload when subsequently calling transcriptUploadParameters()
- *parameters* A list of named lists representing the parameters that require values to be passed into transcriptUploadParameters(). The parameters returned may include both information required by the format deserializer (e.g. mappings from tiers to LaBB-CAT layers) and also general information required by LaBB-CAT (e.g. the corpus, episode, and type of the transcript). Each parameter returned is a dict that may contain the following attributes: - "name" - The name that should be used when specifying the value for the parameter when calling transcriptUploadParameters() - "label" - A label for the parameter intended for display to the user. - "hint" - A description of the purpose of the parameter, for display to the user. - "type" - The type of the parameter, e.g. "String", "Double", "Integer", "Boolean". - "required" - True if the value must be specified, False if it is optional. - "value" - A default value for the parameter. - "possibleValues" - A list of possible values, if the possibilities are limited to a finite set. The required parameters may include both information required by the format deserializer (e.g. mappings from tiers to LaBB-CAT layers) and also general information required by LaBB-CAT, such as: - "labbcat_corpus" - The corpus the new transcript(s) belong(s) to. - "labbcat_episode" - The episode the new transcript(s) belong(s) to. - "labbcat_transcript_type" - The transcript type for the new transcript(s). - "labbcat_generate" - Whether to re-regenerate automated annotation layers or not.

See Also

- transcriptUploadParameters
- transcriptUploadDelete
- newTranscript
- updateTranscript

Examples

```
## Not run:
## Get attributes for new transcript
corpus <- getCorpusIds(labbcat.url)[1]
transcript.type.layer <- getLayer(labbcat.url, "transcript_type")
transcript.type <- transcript.type.layer$validLabels[[1]]

## upload transcript and its media
result <- transcriptUpload(labbcat.url, "my-transcript.eaf", "my-transcript.wav", FALSE)
```

```
## use the default parameter values
parameterValues <- list()
for(p in 1:length(parameters$name)) parameterValues[parameters$name[p]] <- parameters$value[p]

## set the upload parameters to finalise the upload
transcript.id <- transcriptUploadParameters(labbcat.url, result$id, parameterValues)

## End(Not run)
```

`transcriptUploadDelete` *Cancel a transcript upload started by a previous call to transcriptUpload().*

Description

This cancels a transcript upload started by a previous call to transcriptUpload() deleting any uploaded files from the server.

Usage

```
transcriptUploadDelete(labbcat.url, id)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	Upload ID returned by the prior call to transcriptUpload().

See Also

- transcriptUpload
- transcriptUploadParameters
- newTranscript
- updateTranscript

Examples

```
## Not run:
## Get attributes for new transcript
corpus <- getCorpusIds(labbcat.url)[1]
transcript.type.layer <- getLayer(labbcat.url, "transcript_type")
transcript.type <- transcript.type.layer$validLabels[[1]]

## upload transcript and its media
result <- transcriptUpload(labbcat.url, "my-transcript.eaf", "my-transcript.wav", FALSE)

## Changed our mind, cancel this upload
transcriptUploadDelete(labbcat.url, result$id)

## End(Not run)
```

`transcriptUploadParameters` *Set the parameters of a transcript already uploaded with transcriptUpload.*

Description

The second part of a transcript upload process started by a call to transcriptUpload(), which specifies values for the parameters required to save the uploaded transcript to LaBB-CAT's database.

Usage

```
transcriptUploadParameters(labbcat.url, id, parameters, no.progress = FALSE)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`id`	Upload ID returned by the prior call to transcriptUpload().
`parameters`	A named list where each name is the name of a parameter returned by transcriptUpload(), and the value is the parameters value.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Details

NB Using transcriptUpload and transcriptUploadParameters is an alternative to using newTranscript or updateTranscript.

If the response includes more parameters, then this method should be called again to supply their values.

Value

The ID of the new transcript in the corpus

See Also

- transcriptUpload
- transcriptUploadDelete
- newTranscript
- updateTranscript

Examples

```
## Not run:
## Get attributes for new transcript
corpus <- getCorpusIds(labbcat.url)[1]
transcript.type.layer <- getLayer(labbcat.url, "transcript_type")
transcript.type <- transcript.type.layer$validLabels[[1]]

## upload transcript and its media
result <- transcriptUpload(labbcat.url, "my-transcript.eaf", "my-transcript.wav", FALSE)

## use the default parameter values
parameterValues <- list()
for(p in 1:length(parameters$name)) parameterValues[parameters$name[p]] <- parameters$value[p]

## set the upload parameters to finalise the upload
transcript.id <- transcriptUploadParameters(labbcat.url, result$id, parameterValues)

## End(Not run)
```

---

updateFragment *Update a transcript fragment*

Description

This function uploads a file (e.g. Praat TextGrid) representing a fragment of a transcript, with annotations or alignments to update in LaBB-CAT's version of the transcript.

Usage

```
updateFragment(labbcat.url, fragment.path)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`fragment.path`	The path to the fragment to upload.

Details

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

Value

A named list with information about the fragment that was updated.

Examples

```
## Not run:
## upload new verison of transcript transcript
updateFragment(labbcat.url, "my-transcript__1.234-5.678.TextGrid")

## End(Not run)
```

`updateTranscript` *Update an existing transcript*

Description

This function uploads a new version of an existing transcript.

Usage

```
updateTranscript(labbcat.url, transcript.path, no.progress = FALSE)
```

Arguments

`labbcat.url`	URL to the LaBB-CAT instance
`transcript.path`	The path to the transcript to upload.
`no.progress`	TRUE to suppress visual progress bar. Otherwise, progress bar will be shown when interactive().

Details

NB This method of uploading is an alternative to using transcriptUpload and transcriptUploadParameters.

For this function to work, the credentials used to connect to the server must have at least 'edit' access.

Value

The ID of the updated transcript in the corpus

See Also

- transcriptUpload
- transcriptUploadParameters
- transcriptUploadDelete
- newTranscript

Examples

```
## Not run:
## upload new verison of transcript transcript
updateTranscript(labbcat.url, "my-transcript.eaf")

## End(Not run)
```

Index