Package 'emuR'

September 27, 2023

Version 2.4.1

Date 2023-09-26

Title Main Package of the EMU Speech Database Management System

Description Provide the EMU Speech Database Management System (EMU-SDMS) with database management, data extraction, data preparation and data visualization facilities. See <https://ips-lmu.github.io/The-EMU-SDMS-Manual/> for more details.

License GPL (>= 2)

Depends R (>= 3.5.0)

Imports tools, utils, graphics, methods, rlang, stringr (>= 1.4.0), uuid, base64enc, shiny, wrassp (>= 0.1.4), jsonlite (>= 1.6.1), RSQLite (>= 2.0.0), DBI (>= 0.3.1), httpuv (>= 1.3.2), dplyr (>= 0.7.8), readr (>= 1.1.1), tibble (>= 1.4.2), purrr (>= 0.2.4), tidyr(>= 0.8.2), mime(>= 0.6), rstudioapi(>= 0.10), httr(>= 1.4.1), V8(>= 3.4.0), cli(>= 2.5.0)

Suggests stats, grDevices, MASS, ggplot2(>= 2.1.0), testthat(>= 0.7.1.99), compare(>= 0.2.4), knitr(>= 1.7), rmarkdown(>= 0.9.2)

Encoding UTF-8

LazyLoad yes

LazyData yes

ZipData no

URL https://github.com/IPS-LMU/emuR, https://ips-lmu.github.io/The-EMU-SDMS-Manual/

BugReports https://github.com/IPS-LMU/emuR/issues

RoxygenNote 7.2.3

NeedsCompilation no

Author Markus Jochim [aut, cre] (<https://orcid.org/0000-0002-5638-4870>), Raphael Winkelmann [aut], Klaus Jaensch [aut, ctb],

Steve Cassidy [aut, ctb],
Jonathan Harrington [aut, ctb]

Maintainer Markus Jochim <markusjochim@phonetik.uni-muenchen.de>

Repository CRAN

Date/Publication 2023-09-27 07:00:10 UTC

R topics documented:

`emuR-package` *emuR - Main Package of the EMU Speech Database Management System*

Description

The emuR package provides the next iteration of the EMU Speech Database Management System with database management, data extraction, data preparation and data visualization facilities.

Details

This package is part of the next iteration of the EMU Speech Database Management System (EMU-SDMS) which aims to be as close to an all-in-one solution for generating, manipulating, querying, analyzing and managing speech databases as possible. For an overview of the system please visit this URL: `http://ips-lmu.github.io/EMU.html`.

It can be viewed as the main component of the EMU-SDMS as it acts as the central instance that is able to interact with every component of the system. It takes care of database managing duties by being able to interact with a speech database that is stored in the emuDB format. Further, it has easy to understand and learn yet expressive and powerful querying mechanics, that allow the user to easily query the annotation structures of the database. Lastly it provides easy data extraction capabilities that extract data (e.g. formant values) which corresponds to the result of a query.

For an introduction to the emuR package please see the `emuR_intro` vignette by calling: `vignette('emuR_intro')`

For information about the `emuDB` database format please see the `emuDB` vignette by calling: `vignette('emuDB')`

For information about the query language used by the EMU-SDMS please see the `EQL` vignette by calling: `vignette('EQL')`

Typical work-flow in emuR (emuDB required):

1. Load database into current R session - `load_emuDB`
2. Database annotation / visual inspection - `serve` and connect the EMU-webApp to the local server
3. Query database - `query` (sometimes followed by `requery_hier` or `requery_seq`)
4. Get trackdata (e.g. formant values) for the result of a query - `get_trackdata`
5. Data preparation
6. Visual data inspection
7. Further analysis and statistical processing

TIP: for a browsable overview of all the functions provided by emuR simply run the command `help.start()` -> click on packages -> click on emuR

References

Harrington, J. (2010). The Phonetic Analysis of Speech Corpora. Blackwell.

Examples

```
## Not run:
# create demo data including an emuDB called "ae"
create_emuRdemoData(dir = tempdir())

# construct path to demo emuDB
path2ae = file.path(tempdir(), "emuR_demoData", "ae")

# load emuDB into current R session
ae = load_emuDB(path2ae)

# query loaded emuDB
lvowels = query(ae, "Phonetic = i: | u: | o:")

# extract labels from query result
lvowels.labs = label(lvowels)

# list all ssffTrackDefinitions of emuDB
list_ssffTrackDefinitions(ae)

# get formant trackdata defined in ssffTrackDefinitions "fm" for query result
lvowels.fm = get_trackdata(ae, lvowels, "fm")

# extract track values at temporal midpoint of segments
lvowels.fmCut = dcut(lvowels.fm, .5, prop = TRUE)

# Plot the data as time signal and formant card
dplot(lvowels.fm[,1:2], lvowels.labs, normalise=TRUE, main = "Formants over vowel duration")
eplot(lvowels.fmCut[,1:2], lvowels.labs, dopoints=TRUE,
      doellipse=FALSE, main = "F1/F2 of vowel midpoint", form=TRUE,
      xlab = "F2 in Hz", ylab = "F1 in Hz")


# Plot of spectral data from 50% of aspiration duration
hs = query(ae,"Phonetic = H")
hs.labs = label(hs)
hs.dft = get_trackdata(ae, hs, "dft")
hs.dftCut = dcut(hs.dft, .5, prop=TRUE)
plot(hs.dftCut, hs.labs, main = "Spectral data of aspiration")


## End(Not run)
```

AddListRemoveAttrDefLabelGroup

Add / List / Remove labelGroup to / of / from attributeDefinition of emuDB

Description

Add / List / Remove label group to / of / from a specific attribute definition. This label group can be used as a short hand to reference groups of labels specific to an attribute definition (compared to global label groups that are added by `add_labelGroup`) in a `query`. A common example would be to add a label group for something like the phonetic category of nasals to be able reference them as "nasals" in a `query`. For more information on the structural elements of an emuDB see `vignette(emuDB)`.

Usage

```
add_attrDefLabelGroup(
  emuDBhandle,
  levelName,
  attributeDefinitionName,
  labelGroupName,
  labelGroupValues
)

list_attrDefLabelGroups(emuDBhandle, levelName, attributeDefinitionName)

remove_attrDefLabelGroup(
  emuDBhandle,
  levelName,
  attributeDefinitionName,
  labelGroupName
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`levelName`	name of level
`attributeDefinitionName`	name of attributeDefinition
`labelGroupName`	name of label group
`labelGroupValues`	character vector of labels

See Also

add_labelGroup

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

sampaNasals = c("m", "F", "n", "J", "N")

# add these values to the default Phonetic attribute
# definition of the Phonetic level of the ae emuDB
add_attrDefLabelGroup(emuDBhandle = ae,
                      levelName = "Phonetic",
                      attributeDefinitionName = "Phonetic",
                      labelGroupName = "sampaNasals",
                      labelGroupValues = sampaNasals)

# query the labelGroup
query(ae, "Phonetic=sampaNasals")


# list attribute definition label groups
# of attributeDefinition "Phonetic" of the level "Phonetic"
# of the ae emuDB
list_attrDefLabelGroups(emuDBhandle = ae,
                        levelName = "Phonetic" ,
                        attributeDefinitionName = "Phonetic")

# remove the newly added attrDefLabelGroup
remove_attrDefLabelGroup(emuDBhandle = ae,
                         levelName = "Phonetic",
                         attributeDefinitionName = "Phonetic",
                         labelGroupName = "sampaNasals")


## End(Not run)
```

`AddListRemoveLabelGroup` *Add / List / Remove global labelGroup to / of / from emuDB*

Description

Add / List / Remove label group that can be used as a short hand to reference groups of labels that are globally defined for the entire database (compared to attribute definition specific label groups that are added by `add_attrDefLabelGroup`) in a `query`. A common example would be to add a label group for something like the phonetic category of nasals to be able to reference them as "nasals" in a `query`. In theory you could use a labelGroupName as a label instance within the level, but since

this could lead to serious confusion, it is better avoided. For users transitioning from the legacy EMU system: Do not confuse a labelGroup with legal labels: a labelGroup had the unfortunate name 'legal labels' in the legacy EMU system. For more information on the structural elements of an emuDB see `vignette{emuDB}`.

Usage

```
add_labelGroup(emuDBhandle, name, values)

list_labelGroups(emuDBhandle)

remove_labelGroup(emuDBhandle, name)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`name`	name of label group
`values`	character vector of labels

See Also

add_attrDefLabelGroup

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

sampaNasals = c("m", "F", "n", "J", "N")

# add these values to the ae emuDB
# as a globally available labelGroup
add_labelGroup(emuDBhandle = ae,
               name = "sampaNasals",
               values = sampaNasals)

# query the labelGroup in the "Phonetic" level
query(emuDBhandle = ae,
      query = "Phonetic == sampaNasals")

# query the labelGroup in the "Phoneme" level
query(emuDBhandle = ae,
      query = "Phoneme == sampaNasals")

# list global label groups of ae emuDB
list_labelGroups(emuDBhandle = ae)

# remove the newly added labelGroup
```

```
remove_labelGroup(emuDBhandle = ae,
                  name = "sampaNasals")

## End(Not run)
```

`AddListRemoveLevelDefinitions`
Add / List / Remove level definition to / of / from emuDB

Description

Add / List / Remove database operation functions for level definitions. A level is a more general term for what is often referred to as a "tier". It is more general in the sense that people usually expect tiers to contain time information. Levels can either contain time information if they are of the type "EVENT" or of the type "SEGMENT" but are timeless if they are of the type "ITEM". For more information on the structural elements of an emuDB see `vignette(emuDB)`. Note that a level cannot be removed, if it contains instances of annotation items or if it is linked to another level. Further note, renaming a level definition can be done using `rename_attributeDefinition`.

Usage

```
add_levelDefinition(
  emuDBhandle,
  name,
  type,
  rewriteAllAnnots = TRUE,
  verbose = TRUE
)

list_levelDefinitions(emuDBhandle)

remove_levelDefinition(
  emuDBhandle,
  name,
  rewriteAllAnnots = TRUE,
  force = FALSE,
  verbose = TRUE
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`name`	name of level definition
`type`	type of level definition ("SEGMENT","EVENT","ITEM")

rewriteAllAnnots	should changes be written to file system (_annot.json files) (intended for expert use only)
verbose	Show progress bars and further information
force	delete all items incl. links pointing to those items from the levels

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# add level called "Phonetic2" to the ae emuDB
# that could for example contain the transcriptions of a second annotator
add_levelDefinition(emuDBhandle = ae,
                    name = "Phonetic2",
                    type = "SEGMENT")

# list level definition of ae emuDB
list_levelDefinitions(emuDBhandle = ae)

# remove newly added level definition
remove_levelDefinitions(emuDBhandle = ae,
                        name = "Phonetic2")

## End(Not run)
```

AddListRemoveLinkDefinition

Add / List / Remove linkDefinition to / of / from emuDB

Description

Add / List / Remove new link definition to / of / from emuDB. A link definition specifies the relationship between two levels, the super-level and the sub-level. The entirety of all link definitions of a emuDB specifies the hierarchical structure of the database. For more information on the structural elements of an emuDB see `vignette(emuDB)`.

Usage

```
add_linkDefinition(emuDBhandle, type, superlevelName, sublevelName)

list_linkDefinitions(emuDBhandle)

remove_linkDefinition(
```

```
  emuDBhandle,
  superlevelName,
  sublevelName,
  force = FALSE,
  verbose = TRUE
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
type	type of linkDefinition (either "ONE_TO_MANY", "MANY_TO_MANY" or "ONE_TO_ONE")
superlevelName	name of super-level of linkDefinition
sublevelName	name of sub-level of linkDefinition
force	delete all links belonging to the linkDefinition (**USE WITH CAUTION! VERY INVASIVE AKTION!**)
verbose	be verbose. Ask to delete links if force is TRUE.

Details

Link type descriptions:

- "ONE_TO_MANY"A single ITEM of the super-level can be linked to multiple ITEMs of the sub-level
- "MANY_TO_MANY"Multiple ITEMs of the super-level can be linked to multiple ITEMs of the sub-level
- "ONE_TO_ONE"A single ITEM of the super-level can be linked to a single ITEM of the sub-level

For all link types the rule applies that no links are allowed to cross any other links. Further, a linkDefinition can not be removed, if there are links present in the emuDB.

Examples

```
## Not run:

#################################
# prerequisite: loaded emuDB that was converted
# using the convert_TextGridCollection function called myTGcolDB
# (see ?load_emuDB and ?convert_TextGridCollection for more information)

# add link definition from super-level "Phoneme"
# to sub-level "Phonetic" of type "ONE_TO_MANY"
# for myTGcolDB emuDB
add_linkDefinition(emuDBhandle = myTGcolDB,
                   type = "ONE_TO_MANY",
                   superlevelName = "Phoneme",
                   sublevelName = "Phonetic")

# list link definitions for myTGcolDB emuDB
```

```
list_linkDefinitions(emuDBhandle = myTGcolDB)

# remove newly added link definition
remove_linkDefinition(emuDBhandle = myTGcolDB,
                      superlevelName = "Phoneme",
                      sublevelName = "Phonetic")



## End(Not run)
```

`AddListRemovePerspective` *Add / List / Remove perspective to / of / from emuDB*

Description

Add / List / Remove perspective to / of / from emuDB. The EMU-webApp subdivides different ways to look at an emuDB into so called perspectives. These perspectives, between which you can switch in the web application, contain information on what levels are displayed, which ssff-Tracks are drawn, and so on. For more information on the structural elements of an emuDB see `vignette{emuDB}`.

Usage

```
add_perspective(emuDBhandle, name)

list_perspectives(emuDBhandle)

remove_perspective(emuDBhandle, name)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`name`	name of perspective

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# add perspective called "justTones" to the ae emuDB
add_perspective(emuDBhandle = ae,
                name = "justTones")

# add levelCanvasOrder so only the "Tone" level is displayed
```

```
set_levelCanvasesOrder(emuDBhandle = ae,
                       perspectiveName = "justTones",
                       order = c("Tone"))

# list perspectives of ae emuDB
list_perspectives(emuDBhandle = ae)

# remove newly added perspective
remove_perspective(emuDBhandle = ae,
                   name = "justTones")


## End(Not run)
```

`AddListRemoveSsffTrackDefinition` *Add / List / Remove ssffTrackDefinition to / from / of emuDB*

Description

Add / List / Remove ssffTrackDefinition to / from / of emuDB. An ssffTrack (often simply referred to as a track) references data that is stored in the Simple Signal File Format (SSFF) in the according bundle folders. The two most common types of data are:

- complementary data that was acquired during the recording such as data acquired during electromagnetic articulographic (EMA) or electropalatography (EPG) recordings;
- derived data, i.e. data that was calculated from the original audio signal such as formant values and their bandwidths or the short-term Root Mean Square amplitude of the signal.

For more information on the structural elements of an emuDB see `vignette(emuDB)`.

Usage

```
add_ssffTrackDefinition(
  emuDBhandle,
  name,
  columnName = NULL,
  fileExtension = NULL,
  onTheFlyFunctionName = NULL,
  onTheFlyParams = NULL,
  onTheFlyOptLogFilePath = NULL,
  verbose = TRUE,
  interactive = TRUE
)

list_ssffTrackDefinitions(emuDBhandle)

remove_ssffTrackDefinition(emuDBhandle, name, deleteFiles = FALSE)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
name	name of ssffTrackDefinition
columnName	columnName of ssffTrackDefinition. If the onTheFlyFunctionName parameter is set and columnName isn't, the columnName will default to the first entry in wrasspOutputInfos[[onTheFlyFunctionName]]$tracks.
fileExtension	fileExtension of ssffTrackDefinitions. If the onTheFlyFunctionName parameter is set and fileExtension isn't, the fileExtension will default to the first entry in wrasspOutputInfos[[onTheFlyFunctionName]]$ext.
onTheFlyFunctionName	name of wrassp function to do on-the-fly calculation. If set to the name of a wrassp signal processing function, not only the emuDB schema is extended by the ssffTrackDefintion but also the track itself is calculated from the signal file and stored in the emuDB. See names(wrasspOutputInfos) for a list of all the signal processing functions provided by the wrassp package.
onTheFlyParams	a list of parameters that will be given to the function passed in by the onTheFlyFunctionName parameter. This list can easily be generated using the formals function on the according signal processing function provided by the wrassp package and then setting the parameter one wishes to change.
onTheFlyOptLogFilePath	path to optional log file for on-the-fly function
verbose	Show progress bars and further information
interactive	ask user for confirmation
deleteFiles	delete files that belong to ssffTrackDefinition on removal

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# add ssff track definition to ae emuDB
# calculating the according SSFF files (.zcr) on-the-fly
# using the wrassp function "zcrana" (zero-crossing-rate analysis)
add_ssffTrackDefinition(emuDBhandle = ae,
                        name = "ZCRtrack",
                        onTheFlyFunctionName = "zcrana")

# add ssff track definition to ae emuDB
# for SSFF files that will be added later (either
# by adding files to the emuDB using
# the add_files() function or by calculating
# them using the according function provided
# by the wrassp package)
add_ssffTrackDefinition(emuDBhandle = ae,
```

```
                        name = "formants",
                        columnName = "fm",
                        fileExtension = "fms")

# list ssff track definitions for ae emuDB
list_ssffTrackDefinitions(emuDBhandle = ae)

# remove newly added ssff track definition (does not delete
# the actual .zcr files)
remove_ssffTrackDefinition(emuDBhandle = ae,
                           name = "ZCRtrack")


## End(Not run)
```

`AddListRenameRemoveAttributeDefinitions`
Add / List / Rename / Remove attribute definition to / of / from emuDB

Description

Add / List / Rename / Remove database operation functions for attribute definition to / of / from an existing level definition of an emuDB. Attribute definitions can be viewed as definitions of parallel labels for the annotational units (ITEMs) of the emuDB. Each level definition is required to have at least one default attribute definition that has the same name as the level definition (automatically created by `add_levelDefinition`). For more information on the structural elements of an emuDB see `vignette(emuDB)`. Note that as with level definitions, an attribute definition to a level cannot be removed, if it contains labels in the emuDB.

As the only one of these operations, `rename_attributeDefinition` can also be used to manipulate (i.e. rename) a level definition. It is therefore not necessary to specify the name of the level that the attribute definition belongs to. While renaming a level or attribute definition, emuR will (1) rewrite the levelDefinitions in DBconfig, (2) rewrite the linkDefinitions in DBconfig, (3) rewrite the perspectives in DBconfig, (4) rewrite the anagestConfig in DBconfig, and (5) rewrite all _annot.json files. (5) May take quite a while, depending on the number of bundles in the database.

Usage

```
add_attributeDefinition(
  emuDBhandle,
  levelName,
  name,
  type = "STRING",
  rewriteAllAnnots = TRUE,
  verbose = TRUE
)

list_attributeDefinitions(emuDBhandle, levelName)
```

```
rename_attributeDefinition(
  emuDBhandle,
  origAttrDef,
  newAttrDef,
  verbose = TRUE
)

remove_attributeDefinition(
  emuDBhandle,
  levelName,
  name,
  force = FALSE,
  rewriteAllAnnots = TRUE,
  verbose = TRUE
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
levelName	name of level
name	name of attributeDefinition
type	type of attributeDefinition (currently only "STRING")
rewriteAllAnnots	should changes be written to file system (_annot.json files) (intended for expert use only)
verbose	if set to TRUE, more status messages are printed
origAttrDef	name of level/attribute definition in emuDB that is to be changed
newAttrDef	new name that shall be assigned to the level/attribute definition
force	delete all attribute definitions in annotations (== label entries)

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# add additional attribute definition to the "Phonetic" level
# of the ae emuDB that will contain the UTF8 IPA
# symbols of the phonetic transcriptions
add_attributeDefinition(emuDBhandle = ae,
                        levelName = "Phonetic",
                        name = "IPA-UTF8")

# list attribute definitions for level "Word"
# of the ae emuDB
```

```
list_attributeDefinitions(emuDBhandle = ae,
                          levelName = "Word")

# remove newly added attributeDefinition
remove_attributeDefinition(emuDBhandle = ae,
                           levelName = "Phonetic",
                           name = "IPA-UTF8")

## End(Not run)
```

`add_files` *Add files to emuDB*

Description

Add files to existing bundles of specified session of emuDB. Do not use this function to import new recordings (media files) and create bundles; see `?import_mediaFiles` to import new recordings. The files that are found in `dir` that have the extension `fileExtension` will be copied into the according bundle folder that have the same basename as the file. Note that the same bundle name may appear in different sessions, therefore you must specify the session in `targetSessionName`. For more information on the structural elements of an emuDB see `vignette{emuDB}`. Note that adding files does not mean the emuDB is automatically using these, unless you have defined the usage of these files (e.g. by ssffTrackDefinitions).

Usage

```
add_files(emuDBhandle, dir, fileExtension, targetSessionName = "0000")
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`dir`	directory containing files to be added
`fileExtension`	file extension of files to be added. If no . (dot) is found in this string (e.g. "zcr") then the bundle name matching is performed by removing `paste0(".", fileExtension)` from the files ("/path/to/msajc003.zcr" will become "msajc003") and the according bundle name will be searched. If a . (dot) if found within this string (e.g. "_annot.json") then the entire string is remove without prepending a . (dot) ("/path/to/msajc003_annot.json" will then become "msajc003")
`targetSessionName`	name of sessions containing bundles that the files will be added to

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
```

```
# (see ?load_emuDB for more information)

# specify path to folder containing the following
# files we wish to add to:
# msajc003.zcr, msajc010.zcr, msajc012.zcr, msajc015.zcr,
# msajc022.zcr, msajc023.zcr and msajc057.zcr
path2dir = "/path/to/dir/"

# add the files to session "0000" of the "ae" emuDB
add_files(emuDBhandle = ae,
          dir = path2dir,
          fileExtension = "zcr",
          targetSessionName = "0000")


## End(Not run)
```

`as.spectral` *Function to convert an object into an object of class 'spectral'.*

Description

The function converts a vector, matrix, or EMU-trackdata object into an object of the same class and of class 'spectral'

Usage

```
as.spectral(trackdata, fs)
```

Arguments

`trackdata`	A vector, matrix, or EMU-trackdata object.
`fs`	Either a single element numeric vector, or a numeric vector of the same length as the length of trackdata if trackdata is a vector, or of the same number of rows as trackdata

Details

If fs is a single element numeric vector, then the frequencies of trackdata are defined to extend to fs/2. If fs is missing, then the frequencies are 0:(N-1) where N is the length of trackdata.

Value

The same object but of class 'spectral'.

Author(s)

Jonathan Harrington

See Also

is.spectral plot.spectral

Examples

```
vec = 1:10
as.spectral(vec, 2000)
mat = rbind(1:10, 1:10)
as.spectral(mat)
# turn a spectral trackdata object into a trackdata object
tr = as.trackdata(rbind(fric.dft$data), fric.dft$index, fric.dft$ftime)
# turn it into a spectral trackdata object with sampling freq 16 kHz
tr = as.spectral(tr, 16000)
# list the frequencies
trackfreq(tr)
# Notice that only the $data is made into a spectral matrix,
# not the entire trackdata object
# so this is trackdata
class(tr)
# this is a spectral matrix
class(tr$data)
```

---

as.trackdata *Create an Emu trackdata object*

Description

Create an Emu trackdata object from a raw data matrix.

Usage

```
as.trackdata(data, index, ftime, trackname = "")
```

Arguments

data	A two dimensional matrix of numerical data.
index	Segment index, one row per segment, two columns give the start and end rows in the data matrix for each segment.
ftime	A two column matrix with one row per segment, gives the start and end times in milliseconds for each segment.
trackname	The name of the track.

Details

Emu trackdata objects contain possibly multi-column numerical data corresponding to a set of segments from a database. Data for each segment takes up a number of rows in the main `data` matrix, the start and end rows are stored in the `index` component. The `ftime` component contains the start and end times of the segment data.

Trackdata objects are returned by the `get_trackdata` function.

Value

The components are bound into a trackdata object.

See Also

`get_trackdata` `dplot`

Examples

```
# make a trackdata object of two data segments
data1 <- matrix( 1:10, ncol=2 )
data2 <- matrix( 11:20, ncol=2 )

nd1 <- nrow(data1)
nd2 <- nrow(data2)
index <- rbind( c( 1, nd1 ), c(nd1+1,nd1+nd2) )

times <- rbind( c( 100.0, 110.0 ), c( 200.0, 210.0 ) )

tdata <- as.trackdata( rbind( data1, data2 ), index, times, trackname="fake")

# describe the data
summary(tdata)
# get the data for the first segment
tdata[1]
# and the second
tdata[2]
```

`autobuild_linkFromTimes` *Autobuild links between two levels using their time information*

Description

Autobuild links between two time levels. This is typically done when converting from a database / annotation format that allows parallel time tiers / levels but does not permit annotational units to be linked to each other, except by matching time information (such as Praat's TextGrid format). The super-level has to be of the type SEGMENT and the sub-level either of type EVENT or of type SEGMENT. If this is the case and a according link definition is defined for the emuDB, this function automatically links the events or segments of the sub-level which occur within (startSample to (startSample + sampleDur)) the segments of the super-level to those segments.

Usage

```
autobuild_linkFromTimes(
  emuDBhandle,
  superlevelName,
  sublevelName,
  rewriteAllAnnots = TRUE,
  convertSuperlevel = FALSE,
  backupLevelAppendStr = "-autobuildBackup",
  newLinkDefType = NULL,
  verbose = TRUE
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`superlevelName`	name of level to link from (link definition required in emuDB)
`sublevelName`	name of level to link to (link definition required in emuDB)
`rewriteAllAnnots`	should changes be written to file system (_annot.json files) after completing autobuild process (intended for expert use only)
`convertSuperlevel`	if set to TRUE a backup of the superlevel will be created and the actual superlevel will be converted to a level of type ITEM
`backupLevelAppendStr`	string appended to level name for backup level
`newLinkDefType`	type of new linkDefinition (either `"ONE_TO_MANY"`, `"MANY_TO_MANY"` or `"ONE_TO_ONE"`) which is passed to `add_linkDefinition`. If NULL (the default) `add_linkDefinition` isn't called and a linkDefintion is expected to be present.
`verbose`	show progress bars and further information

Details

The type of link definition (ONE_TO_MANY, MANY_TO_MANY, ONE_TO_ONE) is relevant whether a link is generated or not (e.g. overlapping segments are linked in a MANY_TO_MANY relationship but not in a ONE_TO_MANY relationship). For more information on the structural elements of an emuDB see `vignette(emuDB)`.

See Also

add_linkDefinition

Examples

```
## Not run:

#################################
# prerequisite: loaded myTGcolDB emuDB
# (see ?create_emuRdemoData, ?convert_TextGridCollection,
#  and vignette(emuR_intro) for more information)

# add linkDefinition as one has to be present for
# the autobuild function to work
add_linkDefinition(emuDBhandle = myTGcolDB,
                   type = "ONE_TO_MANY",
                   superlevelName = "Syllable",
                   sublevelName = "Phoneme")

# envoke autobuild function to build hierarchy for converted TextGridCollection
autobuild_linkFromTimes(emuDBhandle = myTGcolDB,
                        superlevelName = "Syllable",
                        sublevelName = "Phoneme",
                        convertSuperlevel = TRUE)


## End(Not run)
```

---

bark *Convert Hertz to Bark and Bark to Hertz*

Description

The calculation is done using the formulae Traunmueller (1990)

Usage

```
bark(f, inv = FALSE, ...)
```

Arguments

f	A vector or matrix of data or a spectral object.
inv	A single element logical vector. If FALSE, data are converted from Hertz to Bark, if TRUE, data are converted from Bark to Hertz. (Does not apply if 'data' is an oject of class 'spectral'.
...	for generic only

Details

If 'data' is a spectral object, then
the frequencies are changed so that they are proportional
to the Bark scale and such that the Bark intervals
between frequencies are con stant between the lowest
and highest frequencies. More specifically,
suppose that a spectral object has frequencies
at 0, 1000, 2000, 3000, 4000 Hz. Then the corresponding
frequencies extend in Bark between 0 and 17.46329 Bark
in four equal intervals, and linear interpolation
is used with the 'approx' function to obtain
the dB values at those frequencies. Negative frequencies
which are obtained for values of about less than 40 Hz
are removed in the case of spectral objects.

Value

A vector or matrix or spectral object of the same length and dimensions as data.

Author(s)

Jonathan Harrington

References

Traunmueller, H. (1990) "Analytical expressions for the tonotopic sensory scale" J. Acoust. Soc. Am. 88: 97-100.

See Also

mel,
plot.spectral

Examples

```
# convert Hertz values to Bark

     vec <- c(500, 1500, 2500)

     vec

     bark(vec)
```

```
        # convert Hertz values to Bark and back to Hertz
        bark(bark(vec, inv=TRUE))


        # convert the $data values in a trackdata object to Bark
        # create a new track data object
        t1 <- dip.fdat
        t1[1]


        # convert Hertz to Bark
        t1$data <- bark(t1$data)
        t1[1]

# warp the frequency axis of a spectral object such
# that it is proportional to the Bark scale.
w = bark(e.dft)
oldpar = par(mfrow=c(1,2))
plot(w, type="l")

# The values of w are at equal Bark intervals. Compare
# with
plot(e.dft, freq=bark(trackfreq(e.dft)))
# the latter has a greater concentration of values
```

```
# in a higher frequency range.

par(oldpar)
```

`bind.trackdata` *bind trackdata*

Description

binds different trackdata objects together

Usage

```
## S3 method for class 'trackdata'
bind(...)
```

Arguments

`...`	trackdata objects

`bridge` *Three-columned matrix*

Description

An EMU dataset

Format

Three-columned matrix

buildtrack *Build trackdata objects from the output of by()*

Description

buildtrack() converts a list that is the output of by.trackdata() into a trackdata object if the list components are matrices whose rows are successive values in time.

Usage

```
buildtrack(mylist, ftime = NULL, trackname = "")
```

Arguments

mylist	a list that ist output from by()
ftime	ftime
trackname	name of track data object

Details

The default of by.trackdata() is to return a list. If each element of the list consists of a matrix whose rows are values occurring at the times given by the row dimension names of the matrix, then buildtrack() can be used to convert the list into a trackdata object. If the times are not given in the row dimension names, then these can be supplied as an additional argument to buildtrack()

Author(s)

Jonathan Harrington

See Also

by

Examples

```
#vowlax.fdat is a track data objects of formant of the vowlax segment list
#calculate the difference between adjacent formant values
p = by(vowlax.fdat[1,2],INDICES=NULL, diff)

p


#now build a track data object out of these values
m = buildtrack(p)

m
```

by.trackdata *A method of the generic function by for objects of class 'trackdata'*

Description

A given function 'FUN' is applied to the data corresponding to each segment of data.

Usage

```
## S3 method for class 'trackdata'
by(data, INDICES = NULL, FUN, ..., simplify = FALSE)
```

Arguments

data	a track data object
INDICES	a list of segment indices, like a label vector
FUN	a function that is applied to each segment
...	arguments of the function fun
simplify	simplify = TRUE , output is a matrix; simplify = FALSE a list is returned

Details

It is the same as trapply but with the extension to subsume calculation to groups of segments. Note, if you do not want to apply the function fun to a special group of segments, use trapply instead.

Value

list or vector

Author(s)

Jonathan Harrington

See Also

trapply, by, trackdata dapply smooth apply

Examples

```
data(demo.vowels)
data(demo.vowels.fm)


 #mean F1 subsumed for each vowel
 ################################
 lab = label(demo.vowels)
```

```
by(demo.vowels.fm[,1], lab ,sapply,mean,simplify=FALSE)


#mean F1 subsumed for segment onsets mids and offsets
##########################################
data = demo.vowels.fm
llabs = NULL
for (ind in 1:dim(data$ftime)[1]) {
  seglabs = rep("mid",data$index[ind,2]-data$index[ind,1]+1)
  seglabs[1] = "on"
  seglabs[length(seglabs)] = "off"
  llabs = as.vector(c(llabs , seglabs))
}

by(demo.vowels.fm[,1], llabs , sapply, mean , simplify=FALSE)

#mean F1 subsumed for segment onsets mids and offsets subsumed for each vowel
#####################################################################
by(demo.vowels.fm[,1], list(lab = lab, llabs = llabs) , sapply, mean , simplify=FALSE)
```

`cbind.trackdata` *A method of the generic function cbind for objects of class 'trackdata'*

Description

Different track data objects from one segment list are bound by combining the $data columns of the track data object by columns.

Usage

```
## S3 method for class 'trackdata'
cbind(...)
```

Arguments

`...`	track data objects

Details

All track data objects have to be track data of the same segment list. Thus $index and $ftime values have to be identically for all track data objects. Track data objects are created by get_trackdata(). The number of rows of the track data objects must match.

Value

A track data object with the same $index and ftime values of the source track data objects and with $data that includes all columns of $data of the source track data objects.

Author(s)

Jonathan Harrington

See Also

cbind, rbind.trackdata trackdata get_trackdata

Examples

```
data(vowlax)

#segment list vowlax - first segment only
vowlax[1,]

#F0 track data object for vowlax - first segment only
vowlax.fund[1,]

#rms track data object for vowlax - first segment only
vowlax.rms[1,]


#now combine both track data objects
fund.rms.lax = cbind(vowlax.fund, vowlax.rms)

#the combined track data object - first segment only
#The first column keeps vowlax.fund data, the second keeps vowlax.rms data
fund.rms.lax[1,]
```

---

classify *classify*

Description

classifies data

Usage

```
classify(data, train, metric = "bayes")
```

Arguments

data	data to classify
train	training data
metric	bayes or mahal

Value

The classification matrix.

Author(s)

Jonathan Harrington

Examples

```
## The function is currently defined as
function (data, train, metric = "bayes")
{
    probs <- distance(data, train, metric = metric)
    if (metric == "bayes") {
        best <- apply(probs, 1, max)
    }
    else if (metric == "mahal") {
        best <- apply(probs, 1, min)
    }
    result <- rep("", length(best))
    for (lab in 1:length(train$label)) {
        tmp <- probs[, lab] == best
        result[tmp] <- train$label[lab]
    }
    result
  }
```

`classplot` *Produce a classification plot from discriminant or SVM modelling*

Description

The function classifies all point specified within the ranges of xlim and ylim based on the training model specified in model. It then produces a two-dimensional plot colour-coded for classifications.

Usage

```
classplot(
  model,
  xlim,
  ylim,
  N = 100,
  pch = 15,
  col = NULL,
  legend = TRUE,
```

```
  position = "topright",
  bg = "gray90",
  ...
)
```

Arguments

model	A two-dimensional training model output from qda(), lda() of MASS package , or svm() of e1071 package
xlim	A vector of two numeric elements specifying the range on the x-axis (parameter 1) over which classifications should be made
ylim	A vector of two elements specifying the range on the y-axis (parameter 2) over which classifications should be made
N	A vector of one numeric element which specifies the density of classification (greater N gives higher density). The default is 100.
pch	A single element numeric vector specifying the plotting symbol to be used in the classification plot. Defaults to 15.
col	Either Null in which case the colours for the separate classes are col = c(1, 2, ...n) where n is the number of classes; or else a vector specifying the desired colours that is the same length as there are classes.
legend	A single element logical vector specifying whether a legend should be drawn. Defaults to TRUE
position	A single element vector specifying the position in the figure where the legend should be drawn. Defaults to "topright"
bg	A single element vector specifying the background colour on which the legend should be drawn.
...	Further arguments to plot.

Author(s)

Jonathan Harrington

See Also

qda, lda, svm of e1071 package. There is a function plot.svm which produces a prettier plot for SVMs.

Examples

```
library(MASS)
# Data from female speaker 68
temp = vowlax.spkr=="68"
# Quadratic discriminant analysis
fm.qda = qda(vowlax.fdat.5[temp,1:2], vowlax.l[temp])
# Linear discriminant analysis
fm.lda = lda(vowlax.fdat.5[temp,1:2], vowlax.l[temp])
```

```
xlim=c(0,1000)
ylim=c(0,3000)

oldpar = par(mfrow=c(1,2))
classplot(fm.qda, xlim=xlim, ylim=ylim, main="QDA")
classplot(fm.lda, xlim=xlim, ylim=ylim, main="LDA")

par(oldpar)


# install.packages("e1071")
# library(e1071)
# Support vector machine
## Not run: fm.svm = svm(vowlax.fdat.5[temp,1:2], factor(vowlax.l[temp]))
## Not run: xlim = range(vowlax.fdat.5[temp,1])
## Not run: ylim = range(vowlax.fdat.5[temp,2])
## Not run: classplot(fm.svm, xlim=xlim, ylim=ylim, xlab="F1", ylab="F2", main="SVM")
```

`convert_BPFCollection` *Convert a Bas Partitur File Collection (audio and BAS Partitur files) to an emuDB*

Description

Converts a Bas Partitur File Collection to an emuDB. Expects a collection of the following form: One master directory <sourceDir> containing any number of file pairs (= bundles). A file pair consists of an audio file with the extension <audioExt> and a BPF file with the extension <bpfExt>. Apart from extensions, the names of corresponding audio and BPF files must be identical. Each BPF file is converted into an emuDB annot file. An emuDB config file matching the data base is created after parsing.

Usage

```
convert_BPFCollection(
  sourceDir,
  targetDir,
  dbName,
  bpfExt = "par",
  audioExt = "wav",
  extractLevels = NULL,
  refLevel = NULL,
  newLevels = NULL,
  newLevelClasses = NULL,
  segmentToEventLevels = NULL,
  unifyLevels = NULL,
  verbose = TRUE
)
```

Arguments

sourceDir	path to the directory containing the Bas Partitur File collection
targetDir	directory where the new emuDB should be saved; if it does not exist, the function tries to create one
dbName	name given to the new emuDB
bpfExt	extension of BPF files (default = "par")
audioExt	extension of audio files (default = "wav")
extractLevels	optional vector containing the names of levels that should be extracted. If NULL (the default) all levels found in the BPF collection are extracted.
refLevel	optional name of level used as reference for symbolic links. If NULL (the default), a link-less data base is created.
newLevels	optional vector containing names of levels in the BPF collection that are not part of the standard BPF levels. See http://www.bas.uni-muenchen.de/forschung/Bas/BasFormatseng.html#Partitur_tiersdef for details on standard BPF levels.
newLevelClasses	optional vector containing the classes of levels in the newLevels vector as integers. Must have the same length and order as newLevels.
segmentToEventLevels	optional vector containing names of segment levels with overlapping segments. The parser treats segments on these levels as events (SEGMENT xyz becomes EVENT xyz_start and EVENT xyz_end). If a level contains segmental overlap but is not in this vector, the parser will throw an error. If overlap resolution leads to event overlap (e.g. if one segment's end coincides with the next segment's start), an error is thrown either way. If in doubt whether a level contains segmental overlap, try running the converter with segmentToEventLevels = NULL and see whether an error occurs.
unifyLevels	optional vector containing names of levels to be unified with the reference level. This means that they are treated as labels of the reference level rather than independent items. At the moment, only purely symbolic (class 1) levels can be unified. Links between the reference level and levels in unifyLevels must be one-to-one.
verbose	display infos, warnings and show progress bar

See Also

convert_TextGridCollection, convert_legacyEmuDB

Examples

```
## Not run:

##################################
# prerequisite: a dir with equally named file pairs *.wav and *.par
# (see ?create_emuRdemoData on how to create a demo)
```

```
# convert file pairs *.wav and *.par in /tmp/BPF_collection into emuRDB 'NewEmuR' in
# dir /tmp/DirNewEmuR; the tier 'ORT' acts as the (word) reference tier; the
# tier 'KAN' is one-to-one bound to 'ORT' as a label
convert_BPFCollection("/tmp/BPF_collection","/tmp/DirNewEmuR",'NewEmuR',
        bpfExt='par',audioExt='wav',refLevel='ORT',unifyLevels=c('KAN'))


## End(Not run)
```

---

convert_legacyEmuDB *Convert legacy EMU database to the emuDB format*

Description

Converts an existing legacy EMU database to emuDB database structure. Copies or rewrites signal files and converts the database configuration and annotation data. The legacy database must be addressed by its template file.

Usage

```
convert_legacyEmuDB(
  emuTplPath,
  targetDir,
  dbUUID = uuid::UUIDgenerate(),
  verbose = TRUE,
  ...
)
```

Arguments

emuTplPath	EMU template file path
targetDir	target directory
dbUUID	optional UUID of emuDB, will be generated by default
verbose	be verbose, default: TRUE
...	currently available additional options:

- rewriteSSFFTracks: if TRUE, rewrite SSFF tracks instead of copying the file to get rid of big endian encoded SSFF files (SPARC), default: TRUE
- ignoreMissingSSFFTrackFiles: if TRUE, missing SSFF track files are ignored, if FALSE an error will be generated, default: TRUE
- sourceFileTextEncoding: encoding of legacy database text files (template, label and hlb files), possible values: NULL, "latin1", "UTF-8" "bytes" or "unknown" :default NULL (uses encoding of operating system platform)
- symbolicLinkSignalFiles: if TRUE, signal files are symbolic linked instead of copied. Implies: rewriteSSFFTracks=FALSE, Default: FALSE

Details

The database will be converted if the legacy database template file `emuTplPath` could be found and successfully loaded and parsed. The legacy template file usually has the extension '.tpl'. The UUID of the new emuDB will be randomly generated by default. If `targetDir` does not exist, the directory and its parents will be created. A new directory with the name of the database and the suffix '_emuDB' will be created in the `targetDir`. If the new database directory exists already, the function stops with an error. The template file is converted to a JSON file.

Some of the flags of the legacy EMU template files are ignored (lines with this syntax: "set [flagName] [flagValue]", known ignored flag names are: 'LabelTracks', 'SpectrogramWhiteLevel', 'HierarchyViewLevels', 'SignalViewLevels'). Legacy EMU utterances are reorganized to sessions and bundles. The naming of the sessions depends on the wildcard path pattern of the primary track: If the path contains no wildcard, only one session with the name '0000' will be created. If the path contains one wildcard path element, the names of the directories matching the pattern will be used as session names. If the path contains more than one wildcard path element, the session name is the concatenation of directory names separated by an underscore character.

Media files (usually WAV files) are copied, SSFF track files are rewritten using the ASSP library of package `wrassp` by default (see option `rewriteSSFFTracks` below, see also read.AsspDataObj write.AsspDataObj). Annotations in EMU hierarchy (.hlb) files and ESPS label files are converted to one JSON file per bundle (utterance). Only those files get copied, which match the scheme of the template file. Additional files in the legacy database directories are ignored. The legacy EMU database will not be modified. For more information on the structural elements of an emuDB see `vignette{emuDB}`.

`options` is a list of key value pairs:

See Also

`load_emuDB`

Examples

```
## Not run:
## Convert legacy EMU database specified by EMU
## template file /mydata/EMU_legacy/ae/ae.tpl to directory /mydata/EMU/
## and load it afterwards

convert_legacyEmuDB("/mydata/EMU_legacy/ae/ae.tpl","/mydata/EMU/")
ae=load_emuDB("/mydata/EMU/ae_emuDB")

## Convert database "ae" and do not rewrite SSFF tracks

convert_legacyEmuDB("/mydata/EMU_legacy/ae/ae.tpl",
"/mydata/EMU/",
options=list(rewriteSSFFTracks=FALSE))

## Convert legacy database "ae" from emuR demo data and load converted emuDB

create_emuRdemoData()
demoTplPath=file.path(tempdir(),"emuR_demoData/legacy_ae/ae.tpl")
targetDir=file.path(tempdir(),"converted_to_emuR")
```

```
convert_legacyEmuDB(demoTplPath,targetDir)
dbHandle=load_emuDB(file.path(targetDir,"ae_emuDB"))


## End(Not run)
```

`convert_TextGridCollection` *Convert a TextGridCollection (e.g. .wav & .TextGrid files) to emuDB*

Description

Converts a TextGridCollection to an emuDB by searching a given directory for .wav & .TextGrid files (default extensions) with the same base name. First, the function generates a file pair list containing paths to files with the same base name. It then generates an emuDB DBconfig based on the first TextGrid in this list which specifies the allowed level names and types in the new emuDB. After this it converts all file pairs to the new format, checking whether they comply to the newly generated database configuration. For more information on the emuDB format see `vignette{emuDB}`. Note that since Praat TextGrids do not permit explicit hierarchical structures, the created emuDB does not contain any links or link definitions. You can however use the `autobuild_linkFromTimes` function after the conversion process to automatically build a hierarchal structure.

Usage

```
convert_TextGridCollection(
  dir,
  dbName,
  targetDir,
  tgExt = "TextGrid",
  audioExt = "wav",
  tierNames = NULL,
  verbose = TRUE
)
```

Arguments

`dir`	path to directory containing the TextGridCollection (nested directory structures are permitted as the function recursively searches through directories, generating the session names from dir. structure)
`dbName`	name given to the new emuDB
`targetDir`	directory where to save the new emuDB
`tgExt`	extension of TextGrid files (default=TextGrid, meaning file names of the form baseName.TextGrid)
`audioExt`	extension of audio files (default=wav, meaning file names of the form baseName.wav)

tierNames	character vector containing names of tiers to extract and convert. If NULL (the default) all tiers are converted.
verbose	display infos & show progress bar

Examples

```
## Not run:

##########################################################
# prerequisite: directory containing .wav & .TextGrid files
# (see \code{?create_emuRdemoData} how to create demo data)

# convert TextGridCollection and store
# new emuDB in folder provided by tempdir()
convert_TextGridCollection(dir = "/path/to/directory/",
                           dbName = "myTGcolDB",
                           targetDir = tempdir())


# same as above but this time only convert
# the information stored in the "Syllable" and "Phonetic" tiers
convert_TextGridCollection(dir = "/path/to/directory/",
                           dbName = "myTGcolDB",
                           targetDir = tempdir(),
                           tierNames = c("Syllable", "Phonetic"))


## End(Not run)
```

---

convert_txtCollection *Converts a collection of audio files and plain text transcriptions into an emuDB*

Description

This function takes as input pairs of media files (i.e. wav files) and plain text transcriptions files. It creates a new emuDB with one bundle per media file, and turns the associated transcription into an item in that bundle. For this purpose, media files and text files belonging to the same bundle must be named identically (with the exception of their respective file extensions). The newly created emuDB is stored in the target directory, and its handle is returned.

Usage

```
convert_txtCollection(
  dbName,
  sourceDir,
  targetDir,
  txtExtension = "txt",
  mediaFileExtension = "wav",
```

```
  attributeDefinitionName = "transcription",
  cleanWhitespaces = TRUE,
  verbose = TRUE
)
```

Arguments

dbName	name of the new emuDB
sourceDir	directory containing the plain text transcription files and media files
targetDir	directory where the new emuDB will be stored
txtExtension	file extension of transcription files
mediaFileExtension	file extension of media files
attributeDefinitionName	label name of the transcription items
cleanWhitespaces	if true, any sequence of whitespaces in the transcription (including newlines and tabs) is transformed into a single blank
verbose	display progress bar

See Also

convert_BPFCollection, convert_TextGridCollection

---

convert_wideToLong *convert tracks of a tibble trackdata object to the long form*

Description

Converts a trackdata tibble object of the form (==wide):

sl_rowIdx	...	T1	T2	T3	...	TN
1	...	T1_value	T2_value	T3_value	...	TN_value

to its long form equivalent:

sl_rowIdx	...	track_name	track_value
1	...	T1	T1_value
1	...	T2	T2_value
1	...	T3	T3_value
...	...	...	...
1	...	TN	TN_value

Usage

```
convert_wideToLong(td, calcFreqs = FALSE)
```

Arguments

td	wide form trackdata tibble object
calcFreqs	calculate an additional column containing frequency values from 0-nyquist frequency that match T1-TN (can be quite useful for spectral data)

Value

long form trackdata tibble object

---

coutts *Segment list of words, read speech, female speaker of Australian English from database epgcoutts*

Description

An EMU dataset

Format

segmentlist

---

coutts.epg *EPG-compressed trackdata from the segment list coutts*

Description

An EMU dataset

Format

segmentlist

---

coutts.l *Vector of word label from the segment list coutts*

Description

An EMU dataset

Format

segmentlist

coutts.rms *rms Data to coutts segment list*

Description

An EMU dataset

Format

segmentlist

Examples

```
data(coutts.rms)
```

coutts.sam *Trackdata of acoustic waveforms from the segment list coutts*

Description

An EMU dataset

Format

trackdata object

coutts2 *Segment list, same as coutts but at a slower speech rate*

Description

An EMU dataset

Format

segmentlist

coutts2.epg *EPG-compressed trackdata from the segment list coutts2*

Description

An EMU dataset

Format

trackdata object

coutts2.l *Vector of word label from the segment list coutts2*

Description

An EMU dataset

Format

vector of word label

coutts2.sam *Trackdata of acoustic waveforms from the segment list coutts2*

Description

An EMU dataset

Format

trackdata object

cr *Plot digital sinuoids.*

Description

The function plots and/or sums digital sinusoids for different parameter settings.

Usage

```
cr(
  A = 1,
  k = 1,
  p = 0,
  N = 16,
  samfreq = NULL,
  duration = NULL,
  const = NULL,
  expon = NULL,
  plotf = TRUE,
  ylim = NULL,
  xlim = NULL,
  values = FALSE,
  xlab = "Time (number of points)",
  ylab = "Amplitude",
  type = "b",
  bw = NULL,
  dopoints = FALSE,
  ...
)
```

Arguments

A	A vector of amplitude values. Defaults to A = 1
k	A vector of cycles (repetitions). Defaults to k = 1
p	A vector of phase values between -pi/2 and pi/2. Defaults to 0.
N	The number of points in the signal. Defaults to 16.
samfreq	If NULL, then a sinusoid is plotted with a frequency of k cycles per N points. Otherwise, if samfreq is an numeric, then the argument to k is interpreted as the frequency in Hz and the sinusoid at that frequency is plotted for however many points are specified by N. For example, if samfreq is 40 (Hz), and if N is 40 and k = 1, then 1 cycle of a 1 Hz sinusoid will be plotted.
duration	Specify the duration in ms. If NULL, the default, then the duration of the sinusoid is in points (N), otherwise if a numeric value is supplied, then in ms. For example, 1/2 second of a 1 cycle sinusoid at a sampling frequency of 40 Hz: duration = 500, k = 1, samfreq=40. A ms value can be supplied only if the sampling frequency is also specified.

const	A single numeric vector for shifting the entire sinusoid up or down the y-axis. For example, when const is 5, then 5 + s, where s is the sinusoid is plotted. Defaults to 0 (zero).
expon	A numeric vector. If supplied, then what is plotted is expon[j]^(c(0:(N - 1) * A cos (2 * pi * k/N * (0:(N-1))). For example, a decaying sinusoid is produced with cr(expon=-0.9). Defaults to NULL (i.e. to expon = 1).
plotf	A single-valued logical vector. If TRUE (default), the sinusoid is plotted.
ylim	A two-valued numeric vector for specifying the y-axis range.
xlim	A two-valued numeric vector for specifying the y-axis range.
values	If TRUE, then the values of the sinusoid are listed. Defaults to FALSE.
xlab	A character vector for plotting the x-axis title.
ylab	A character vector for plotting the y-axis title.
type	A character vector for specifying the line type (see par)
bw	A numeric vector for specifying the bandwidth, if the sampling frequency is supplied. The bandwidth is converted to an exponential (see expon using exp(-rad(bw/2, samfreq = samfreq).
dopoints	this is now redundant.
...	Option for supplying further graphical parameters - see par.

Author(s)

Jonathan Harrington

See Also

crplot

Examples

```
# cosine wave
cr()

# doubling the frequency, 1/3 amplitude, phase = pi/4, 50 points
cr(A=1/3, k=2, p=pi/4, N=50)

# sum 3 sinusoids of different frequencies)
cr(k=c(1, 3, 4))

# sum 2 sinusoids of different parameters
cr(c(1, 2), c(2, 10), c(0, -pi/3), N=200, type="l")


# store the above to a vector and overlay with noise
v = cr(c(1, 2), c(2, 10), c(0, -pi/3), N=200, type="l", values=TRUE)
r = runif(200, -3, 3)
v = v+r
```

```
plot(0:199, v, type="l")


# 100 points of a 50 Hz sinusoid with a 4 Hz bandwidth
# at a sampling frequency of 200 Hz
cr(k=50, bw=4, samfreq=2000, N=100)

# the same but shift the y-axis by +4 (d.c. offset=+4)
cr(const=4, k=50, bw=4, samfreq=2000, N=100)

# sinusoid multiplied by a decaying exponential (same effect as bandwidth)
cr(expon=-0.95,  N=200, type="l")
```

`create_emuDB` *Create empty emuDB*

Description

Creates an empty emuDB in the target directory specified

Usage

```
create_emuDB(
  name,
  targetDir,
  mediaFileExtension = "wav",
  store = TRUE,
  verbose = TRUE
)
```

Arguments

`name`	of new emuDB
`targetDir`	target directory to store the emuDB to
`mediaFileExtension`	defines mediaFileExtention (NOTE: currently only 'wav' (the default) is supported by all components of EMU)
`store`	store new created emuDB to file system
`verbose`	display infos & show progress bar

Details

Creates a new directory [name]_emuDB in targetDir. By default the emuDB is created in the R session, written to the filesystem and then purged from the R session.

Examples

```
## Not run:
# create empty emuDB in folder provided by tempdir()
create_emuDB(name = "myNewEmuDB",
             targetDir = tempdir())

## End(Not run)
```

`create_emuRdemoData` *Create demo data for the emuR package*

Description

Create a folder within the folder specified by the dir argument called emuR_demoData. This folder contains the folders:

- ae_emuDB: Containing an emuDB that adheres to the new format specification (as expected by the `load_emuDB` function). See `vignette(emuDB)` for more information on this database format.
- BPF_collection: Containing a BAS Partitur Format (BPF) file collection (as expected by the `convert_BPFCollection` function)
- legacy_ae: Containing a legacyEmuDB (as expected by the `convert_legacyEmuDB` function)
- TextGrid_collection: Containing a TextGrid file collection (as expected from the `convert_TextGridCollection` function)

Usage

```
create_emuRdemoData(dir = tempdir(), precache = FALSE)
```

Arguments

`dir`	directory to create demo data in (default= tempdir())
`precache`	creates an on-file-system cache for the ae emuDB to allow fast loading (see `load_emuDB` for details about the emuDB file cache)

Examples

```
## Not run:

# create demo data directory in directory
# provided by the tempdir function
create_emuRdemoData(dir = tempdir())

## End(Not run)
```

`create_emuRtrackdata` *create emuRtrackdata object*

Description

Joins `emuRsegs` and `trackdata` objects to create an `emuRtrackdata` object that is a sub-class of a `data.frame` object. This object can be viewed as a flat version of a `trackdata` object that also contains all the information of a `emuRsegs` object. It is meant to ease integration with other packages as it is based on the well known `data.frame` object.

Usage

```
create_emuRtrackdata(sl, td)
```

Arguments

`sl`	seglist of class `emuRsegs`
`td`	`trackdata` object generated from sl

Value

emuRtrackdata object

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# query emuDB (to get object of class emuRsegs)
sl = query(emuDBhandle = ae,
           query = "Phonetic == i:")

# get formats for SEGMENTs in sl (to get object of class trackdata)
td = get_trackdata(emuDBhandle = ae,
                   seglist = sl,
                   onTheFlyFunctionName = "forest")

# create emuRtrackdata object
create_emuRtrackdata(sl = sl, td = td)


## End(Not run)
```

`create_itemsInLevel` *Create new items programmatically*

Description

Allows creating annotation items programmatically on a single level. You have to pass in a data frame describing the new items. Each new item is identified by its `session`, `bundle`, `level`, and depending on the level type either:

- `sequence index (start_item_seq_idx)`: when level type = `ITEM`
- `start`: start time in ms * 1000 (see output of query) when level type = `EVENT`
- `start`: start time in ms * 1000 () when level type = `SEGMENT` (creates gapless segment groups where the last segment ends at the end of the audio file)

. The `level` with its associated `attributes` determines how many labels must be provided. You must provide a label for every existing attribute.

Sessions, bundles, levels and attributes must exist beforehand. The sequence index is explained below.

Within each bundle, there can be multiple annotation items on every level. Their order within the level is given by their sequence index. All *existing* items have a natural-valued sequence index and there are no gaps in the sequences (i.e. if a level contains N annotation items, they are indexed 1..N).

Any newly created item must be given a sequence index. The sequence index may be real-valued (it will automatically be replaced with a natural value). To prepend the new item to the existing one, pass a value lower than one. To append it to the existing items, you can either pass `NA` or any value that you know is greater than n (the number of existing items in that level). It does not need to be exactly n+1. To place the new item between two existing ones, use any real value between the sequence indexes of the existing neighbors.

If you are appending multiple items at the same time, every sequence index (including NA) can only be used once per session/bundle/level combination (because session/bundle/level/sequence index are a unique identifier of an item's).

After creating the items, all sequence indexes (which may now be real-valued, natural-valued or NA) are sorted in ascending order and then replaced with the values 1..n, where n is the number of items on that level. While sorting, NA values are placed at the end.

Usage

```
create_itemsInLevel(
  emuDBhandle,
  itemsToCreate,
  rewriteAllAnnots = TRUE,
  verbose = TRUE
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
itemsToCreate	A data frame with the columns • session, • bundle, • level, • start_item_seq_idx(start_item_seq_idx is used instead of seq_idx so that the result of a query call can be used directly. query can return a sequence of items defined by start_item_seq_idx and end_item_seq_idx which have the same value if single items are returned), • attribute, and • labels. *None* of the columns should be factors. sequenceIndex must be numeric (can be real-valued or natural-valued), all other columns must be of type character.
rewriteAllAnnots	should changes be written to file system (_annot.json files) (intended for expert use only)
verbose	if set to TRUE, more status messages are printed

---

create_links *create links between items*

Description

create links between items

Usage

```
create_links(emuDBhandle, links, rewriteAllAnnots = TRUE, verbose = TRUE)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
links	data.frame like object containing linking information. The required columns are: • session: • bundle • from_id • to_id
rewriteAllAnnots	should changes be written to file system (_annot.json files) (intended for expert use only)
verbose	if set to TRUE, more status messages are printed

`create_spectrogram_image_as_raster` *Create spectrogram image as raster*

Description

Create spectrogram image as raster

Usage

```
create_spectrogram_image_as_raster(
  audioFilePath,
  begin = 0,
  end = 0,
  windowSizeInSecs = 0.01,
  alpha = 0.16,
  lowerFreq = 0,
  upperFreq = 5000,
  window = "GAUSS",
  dynRangeInDB = 70,
  audioChannel = 1,
  preEmphasisFilterFactor = 0.97,
  invert = FALSE
)
```

Arguments

`audioFilePath`	path to audio file to plot spectrogram of
`begin`	begin time in seconds (passed into begin parameter of `wrassp::read.AsspDataObj`)
`end`	end time in seconds (passed into end parameter of `wrassp::read.AsspDataObj`)
`windowSizeInSecs`	window size in seconds
`alpha`	value of spectrogram
`lowerFreq`	lower frequency limit of spectrogram
`upperFreq`	upper frequency limit of spectrogram
`window`	window type used in spectrogram calculation. Allowed values are: • "BARTLETT" • "BARTLETTHANN" • "BLACKMAN" • "COSINE" • "GAUSS" (the default) • "HAMMING" • "HANN"

- "LANCZOS"
- "RECTANGULAR"
- "TRIANGULAR"

dynRangeInDB	dynamic range in DB of spectrogram
audioChannel	channel of audio file to draw spectrogram of (only applicable when using multi-channel audio files)
preEmphasisFilterFactor	used in time domain for amplifying high-freqs
invert	invert the colors of the spectrogram

Value

a image raster object

`crplot` *Function to plot a digital sinusoid and the circle from which it is derived.*

Description

A digital sinusoid is derived the movement of a point around a circle. The function shows the relationship between the two for various parameter settings.

Usage

```
crplot(
  A = 1,
  k = 1,
  p = 0,
  N = 16,
  const = NULL,
  figsize = 8,
  npoints = 500,
  col = 1,
  cplot = TRUE,
  splot = TRUE,
  numplot = TRUE,
  axes = TRUE,
  incircle = TRUE,
  arrow = TRUE,
  linetype = 1,
  textplot = NULL,
  lineplot = NULL,
  ylab = "Amplitude",
  super = NULL,
  xaxlab = NULL,
```

```
  type = "b",
  xlab = "Time (number of points)",
  fconst = 3.5/3.1,
  pointconst = 1.2
)
```

Arguments

A	Amplitude of the circle/sinusoid.
k	Frequency of the sinusoid
p	Phase of the sinusoid
N	Number of points per cycle or revolution.
const	A constant corresponding to k + A*cos(2*pi*k+p)
figsize	Set the figure size as pin <- c(figsize, figsize/2). Defaults to figsize = 8.
npoints	The number of points used in plotting the circle. Defaults to 500
col	An integer for the color in plotting the sinusoid and points around the circle
cplot	Now redundant
splot	Now redundant
numplot	Logical. If TRUE (defaults), the digital points around the circle are numbered
axes	Logical. If TRUE, plot axes.
incircle	Logical. If TRUE, plot an the angle between digital points in the circle.
arrow	Logical. If TRUE, plot an arrow on incircle showing the direction of movement.
linetype	Specify a linetype. Same function as lty in plot
textplot	A list containing $radius, $textin, $pivals for plotting text at specified angles and radii on the circle. $radius: a vector of amplitudes of the radii at which the text is to be plotted; $textin: a vector of character labels to be plotted; $pivals: the angle, in radians relative to zero radians (top of the circle) at which the text is to be plotted. Defaults to NULL
lineplot	Plot lines from the centre of the circle to the circumference. lineplot is a vector specifying the angle in radians (zero corresponds to the top of the circle)
ylab	Specify a y-axis label.
super	Superimpose a part solid circle and corresponding sinusoid. This needs to be a list containing $first and $last, which are values between 0 and 2*pi defining the beginning and ending of the part circle which is to be superimposed
xaxlab	Now redundant
type	Specify a type.
xlab	Specify an x-axis label.
fconst	A single element numeric vector for the aspect ratio in a postscript plot. Defaults to 3.5/3.1 which is appropriate for a postscript setting of setps(h=4, w=4)
pointconst	The radius for plotting the numbers around the circle. Defaults to 1.2 * A

Author(s)

Jonathan Harrington

References

Harrington, J, & Cassidy, S. 1999. Techniques in Speech Acoustics. Kluwer

See Also

cr

Examples

```
crplot()
# sine wave
crplot(p=-pi/2)

crplot(k=3)

# aliasing
crplot(k=15)
```

---

dapply *apply a function to each part of a trackdata object*

Description

Given an Emu trackdata object, dapply will apply a given function to the data corresponding to each segment of data. The result is a new trackdata object.

Usage

```
dapply(trackdata, fun, ...)
```

Arguments

trackdata	An Emu trackdata object
fun	A function taking a matrix of data and a vector of times and returning a list with components $data and $ftime.
...	Additional arguments to be passed to fun

Details

dapply can be used to apply an arbitrary function to trackdata extracted from an Emu database. It can be used for example to smooth the data (see dsmooth) or differentiate it (see ddiff).

Trackdata is made up of three components: a matrix of data $data, a matrix of indexes ($index) and a matrix of segment times ($ftime). The indexes contain the start and end rows for each segment in the trackdata, the time matrix contains the start and end times of each data segment.

The function fun supplied to dapply should take one matrix of data (corresponding to one segment) and a vector of two times being the start and end of the data. It should return a modified data matrix, which can have any number of rows or columns, and a new pair of start and end times. The new start and end times are necessary because the operation applied might shorten or interpolate the data and hence change the times corresponding to the first and last rows of data.

Value

An Emu trackdata object with components:

data	A matrix of data with all segments concatenated by row.
index	A two column matrix of the start and end rows for each segment
ftime	A two column matrix of the start and end times for each segment

See Also

dsmooth ddiff

Examples

```
data(dip)
## formant data of the first segment in segment list dip
fm <- dip.fdat[1]

testfun <- function(data, ftime, n) {
  ## return only the first n rows of data
  ## doesn't check to see if there really are n rows...
  newdata <- data[1:n,]
  ## calculate a new end time
  interval <- (ftime[2]-ftime[1])/nrow(data)
  ftime[2] <- ftime[1] + interval*n
  ## now return the required list
  return( list( data=newdata, ftime=ftime ) )
}

fm.first3 <- dapply( fm, testfun, 3 )
fm.first10 <- dapply( fm, testfun, 10 )
```

dbnorm *Function to dB-normalise spectral objects*

Description

The function can be used to rescale a spectrum to a dB value at a particular frequency - for example, to rescale the spectrum so that 3000 Hz has 0 dB and all other values are shifted in relation to this.

Usage

```
dbnorm(specdata, f = 0, db = 0)
```

Arguments

specdata	An object of class 'spectral'
f	A single element vector specifying the frequency. Defaults to 0
db	A single element vector specifying the dB value to which the spectrum is to be rescaled. Defaults to zero

Value

An object of the same class with rescaled dB values. The default is to rescale the dB-values of the spectrum to 0 dB at 0 Hz.

Author(s)

Jonathan Harrington

See Also

dbtopower plot.spectral

Examples

```
# normalise to - 40 dB at 1500 Hz
res = dbnorm(e.dft, 1500, 0)
# compare the two
ylim = range(c(res, e.dft))
plot(e.dft, ylim=ylim, type="l")
oldpar = par(new=TRUE)
plot(res, ylim=ylim, type="l", col=2)

par(oldpar)
```

dbtopower *Function for inter-converting between decibels and a linear scale*

Description

The function converts from decibels to a linear scale

Usage

```
dbtopower(specdata, const = 10, base = 10, inv = FALSE)
```

Arguments

specdata	A numeric object or an object of class trackdata
const	A single element numeric vector. Defaults to 10
base	A single element numeric vector. Defaults to 10
inv	Logical. If TRUE, then the conversion is from a logarithmic to an anti-logarithmic form, otherwise the other way round

Details

The function returns base^(specdata/const) if inv=FALSE, otherwise, const * log(dat, base=base). If the object to which this function is applied is of class 'trackdata' then this function is applied to $data.

Value

An object of the same class.

Author(s)

Jonathan Harrington

See Also

dbtopower plot.spectral

Examples

```
# convert 10 dB to a power ratio
vec = dbtopower(10)
# convert dB-data to a power ratio and back to decibels
res = dbtopower(vowlax.dft.5)
res = dbtopower(res, inv=TRUE)
```

dct *Discrete Cosine Transformation*

Description

Obtain the coefficients of the discrete cosine transformation (DCTRUE).

Usage

```
dct(data, m = NULL, fit = FALSE)
```

Arguments

data	a vector or single column matrix of numeric values to which the 2nd order polynomial is to be fitted.
m	The number of DCT coefficients that are returned or on which the smoothed trajectory is based. Defaults to NULL which returns coefficients of frequencies k = 0, 1,2 .. N-1 where N is the length of the input signal, wav. If fit = TRUE and k = NULL, then the the sum of all the cosine waves whose amplitudes are the DCT coefficients are returned - which is equal to the original signal. k must be between 2 and the length of the signal.
fit	if FALSE, return the DCT coefficients; if TRUE, the values of the smoothed trajectory are returned based on summing the cosine waves of the k lowest ordered DCT coefficients, where k is the argument given below.

Details

The function calculates the DCT coefficients for any vector or single-columned matrix. The function can also be used to obtain a smoothed trajectory of the input data by summing the cosine waves derived from the first few DCT coefficients.

The algorithm first reflects the input signal about the last data point, N. Thus if the input signal vec if of length N, the algorithm creates a vector c(vec, rev(vec[-c(1,N)])). and the R fft function is applied to this reflected signal. The DCT coefficients are real part of what is returned by fft i.e. the amplitudes of the cosine waves of frequencies k = 0, 1, 2, ...2 *(N-1) radians per sample. The phase is zero in all cases. The amplitudes are calculated in such a way such that if these cosine waves are summed, the original (reflected) signal is reconstructed. What is returned by dct() are the amplitudes of the cosine waves (DCT coefficients) up to a frequency of N radians/sample, i.e. a vector of cosine wave amplitudes that has the same length as the original signal and of frequencies k = 0, 1, 2, ... (N-1). Alternatively, if fit=TRUE, a smoothed signal of the same length as the original signal is obtained based on a summation of the lowest ordered DCT coefficients. This dct() algorithm returns very similar values to DCT() with inv=FALSE written by Catherine Watson and used in Watson & Harrington (1999).

Author(s)

Jonathan Harrington

References

Watson, C. & Harrington, J. (1999). Acoustic evidence for dynamic formant trajectories in Australian English vowels. Journal of the Acoustical Society of America, 106, 458-468.

Zahorian, S., and Jagharghi, A. (1993). Spectral-shape features versus formants as acoustic correlates for vowels, Journal of the Acoustical Society of America, 94, 19661982.

See Also

plafit by

Examples

```
data(vowlax)
# obtain the first four DCT coefficients
# (frequencies k = 0, 1, 2, 3) for some
# first formant frequency data
vec <- vowlax.fdat[1,1]$data
dct(vec, m=4)

# obtain the corresponding smoothed
# trajectory
dct(vec, m=4 , fit=TRUE)
```

---

dcut *Function to extract a vector or matrix from EMU-Trackdata at a single time point of to create another EMU-trackdata object between two times.*

Description

A general purpose tool for extracting data from track objects either at a particular time, or between two times. The times can be values in milliseconds or proportional times between zero (the onset) and one (the offset).

Usage

```
dcut(
  trackdata,
  left.time,
  right.time,
  single = TRUE,
  average = TRUE,
  prop = FALSE
)
```

Arguments

trackdata	An Emu trackdata object.
left.time	Either: a numeric vector of the same length as there are observations in trackdata. Or: a single value between 0 and 1. In the first case, the left time boundary of trackdata[n,] is cut at left.time[n], in the second case, and if prop=TRUE, it is cut at that proportional time.
right.time	Either: a numeric vector of the same length as there are observations in trackdata. Or: a single value between 0 and 1. In the first case, the right time boundary of trackdata[n,] is cut at right.time[n], in the second case, and if prop=TRUE, it is cut at that proportional time.
single	If TRUE, one value is returned per segment. This applies when the requested time falls between two track frames. When single=TRUE, the preceding value is returned, unless average=TRUE (see below), in which case the average value of the two frames is returned. when the right.time argument is omitted
average	A single element logical vector - see single above. Applies only when the right.times argument is omitted and when single = TRUE
prop	If TRUE left.time and right.time are interpreted as proportions, if FALSE, they are interpreted as millisecond times

Details

This function extracts data from each segment of a trackdata object.

If 'prop=FALSE' the time arguments ('left.time' and 'right.time') are interpreted as millisecond times and each should be a vector with the same length as the number of segments in 'trackdata'. If 'prop=TRUE' the time arguments should be single values between zero (the onset of the segment) and one (the offset).

If 'right.time' is omitted then a single data point corresponding to 'left.time' for each segment is returned.

Value

A trackdata object if both 'left.time' and 'right.time' are specified, otherwise a matrix if 'right.time' is unspecified and the trackdata object has multiple columns of data or a vector if right.time' is unspecified and the trackdata object has a single column of data.

Author(s)

Jonathan Harrington

See Also

get_trackdata, dplot, eplot

Examples

```
# the data values of the trackdata object at the temporal midpoint
# (midvals is matrix of F1 and F2 data)
dip.fdat[1:10]
midvals <- dcut(dip.fdat, 0.5, prop=TRUE)
midvals[1:10,]


# the data values of the trackdata object between
# extending from 20
# (bet is a trackdata object of F1 and F2 values)
bet <- dcut(dip.fdat, 0.2, 0.8, prop=TRUE)
bet[1]


# the data values of the trackdata object at 30 ms after
# the start time of the trackdata object
# (time30 is a matrix of F1 and F2 data
times <- dip.fdat$ftime[,1]+30
times[1:10]
time30 <- dcut(dip.fdat, times)
time30[1:10]


# the data values of the trackdata object
# between the  start time and 30 ms after the start  time
# (int is a trackdata object of F1 and F2 values extending
# from the start of the diphthongs up to 30 ms after the diphthongs)
int <- dcut(dip.fdat, dip.fdat$ftime[,1], times)
int[1]
```

`ddiff` *Differentiation of tracks*

Description

Differentiates a list, as returned by track, to the nth order, readjusting the index and ftime values each time.

Usage

```
ddiff(dataset, n = 1, smoothing = TRUE)
```

Arguments

`dataset`	track data object - a list as returned by track
`n`	the order of differentiation
`smoothing`	if TRUE track is smoothed

Author(s)

Jonathan Harrington

`delete_itemsInLevel` *Delete items programmatically*

Description

Allows to delete annotation items programmatically.

Usage

```
delete_itemsInLevel(
  emuDBhandle,
  itemsToDelete,
  rewriteAllAnnots = TRUE,
  verbose = TRUE
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`itemsToDelete`	A data frame with the columns • `session`, • `bundle`, • `level`, • `start_item_seq_idx` (`start_item_seq_idx` is used instead of e.g. `seq_idx` so that the result of a `query` call can be used directly. *None* of the columns should be factors. `sequenceIndex` must be numeric (natural-valued), all other columns must be of type character.
`rewriteAllAnnots`	should changes be written to file system (_annot.json files) (intended for expert use only)
`verbose`	if set to `TRUE`, more status messages are printed

demo.all *Emu segment list*

Description

Segment list of the demo database that is part of the Emu system. It is the result of a database query, that searched all segments at level Phonetic.

Format

First Column labels Second start time of the segment Third end time of the segment Fourth utterance name of the utterance the segment was found

Details

A segment list is created via query.

See Also

demo.vowels segmentlist

demo.all.f0 *F0 track data for segment list demo.vowels*

Description

A track list of the demo database that is part of the Emu system. It is the result of get F0 data for the segment list demo.vowels (see data(demo.vowels)).

Format

An object with $index, $ftime and $data

index: a two columned matrix with the range of the $data rows that belong to the segment ftime: a two columned matrix with the times marks of the segment data: a one columned matrix with the F0 values

Details

A track list is created via the get_trackdata function.

See Also

demo.all.rms segmentlist trackdata

demo.all.fm — *Formant track data for segment list demo.vowels*

Description

A track list of the demo database that is part of the Emu system. It is the result of get fm data for the segment list demo.vowels (see data(demo.vowels)).

Format

index: a two columned matrix with the range of the $data rows that belong to the segment ftime: a two columned matrix with the times marks of the segment data: a three columned matrix with the formant values of the first three formants for each segment

Details

A track list is created via the get_trackdata function.

See Also

demo.all.rms segmentlist trackdata

demo.all.rms — *Emu track data for a rms track for segment list demo.all*

Description

A track list of the demo database that is part of the Emu system. It is the result of get rms data for the segment list demo.all (data(demo.all)).

Format

A object with $index, $ftime and $data

index: a two columned matrix with the range of the $data rows that belong to the segment ftime: a two columned matrix with the times marks of the segment data: a vector with the rms data

Details

A track list is created via the get_trackdata function.

See Also

demo.vowels.fm segmentlist trackdata

demo.vowels *Emu segment List*

Description

Segment list of the demo database that is part of the Emu system. It is the result of a database query, that searched all vowel segments at level Phonetic.

Format

First Column labels Second start time of the segment Third end time of the segment Fourth utterance name of the utterance the segment was found

Details

A segment list is created via query.

See Also

demo.all segmentlist

demo.vowels.f0 *F0 track data for segment list demo.vowels*

Description

A track list of the demo database that is part of the Emu system. It is the result of get F0 data for the segment list demo.vowels (see data(demo.vowels)).

Format

An object with $index, $ftime and $data

index: a two columned matrix with the range of the $data rows that belong to the segment ftime: a two columned matrix with the times marks of the segment data: a one columned matrix with the F0 values

Details

A track list is created via the get_trackdata function.

See Also

demo.all.rms segmentlist trackdata

demo.vowels.fm *Formant track data for segment list demo.vowels*

Description

A track list of the demo database that is part of the Emu system. It is the result of get fm data for the segment list demo.vowels (see data(demo.vowels)).

Format

index: a two columned matrix with the range of the $data rows that belong to the segment ftime: a two columned matrix with the times marks of the segment data: a three columned matrix with the formant values of the first three formants for each segment

Details

A track list is created via the `get_trackdata` function.

See Also

`demo.all.rms` `segmentlist` `trackdata`

dextract *Extract a subset of data from a trackdata object*

Description

A function that cuts up trackdata either at a proportional time or proportionally between two times. It is a subsidiary function of dplot()

Usage

```
dextract(dataset, start, end)
```

Arguments

`dataset`	A trackdata object
`start`	A single valued numeric vector corresponding to a proportional time between zero (the onset of the trackdata) and one (the offset of the trackdata).
`end`	As start, but optional

Value

If both start and end are specified, a trackdata object is returned, otherwise a vector if the original trackdata is one-dimensional and the end argument is not used, or a matrix if the original trackdata has more than one dimension and the end argument is not used

Author(s)

Jonathan Harrington

See Also

dcut

Examples

```
data(demo.vowels.f0)
data(demo.vowels.fm)

form = demo.vowels.fm
# get the formants at the midpoint: f50 is a matrix
# same as dcut(form, .5, prop=TRUE)
f50 = dextract(form, 0.5)
# get the formants between the 25% and 75% time points
# fcut is a trackdata object
# same as dcut(form, .25, .75, prop=TRUE)
fcut = dextract(form, 0.25, 0.75)
# get  F0 at the midpoint. fzero50 is a vector
# same as dcut(fzero, .5, prop=TRUE)
fzero = demo.vowels.f0
fzero50 = dextract(fzero, 0.5)
```

---

dim.trackdata *A method of the generic function dim for objects of class 'trackdata'*

Description

The function returns the dimension attributes of a track data object.

Usage

```
## S3 method for class 'trackdata'
dim(x)
```

Arguments

x	a track data object

Details

The function returns the dimension attributes of a track data object as the number of segments x number of tracks. c(nrow(x$index), ncol(x$data))

Author(s)

Jonathan Harrington

Examples

```
#isol.fdat is the formant track of the segment list isol

#write out the dimension of the track data object
dim(isol.fdat)

#because there are 13 segments
isol.fdat$ftime

#and there are 4 rows for each segment (see here for the first segment)
isol.fdat$data[1,]
```

---

dimnames.trackdata *Dimnames of trackdata object*

Description

returns dimension names of trackdata objects

Usage

```
## S3 method for class 'trackdata'
dimnames(x)
```

Arguments

x	trackdata object

---

dip *Segment list of diphthongs, two speakers one male, one female , Standard North German, read speech from database kielread*

Description

An EMU dataset

Format

segmentlist

dip.fdat — *Trackdata of formants from the segment list dip*

Description

An EMU dataset

Format

trackdata object

dip.l — *Vector of phoneme labels from the segment list dip*

Description

An EMU dataset

Format

vector of phoneme lables

dip.spkr — *Vector of speaker labels from the segment list dip*

Description

An EMU dataset

Format

vector of speaker labels

dplot *A function to plot one or more columns of EMU-trackdata as a function of time (DEPRECATED see below)*

Description

A general purpose routine for plotting EMU-trackdata on a single plot. Tracks can be aligned at an arbitrary position, length normalised or averaged. The plots can be colour-coded for different category types. DEPRECATED as this function does not play well with with the new resultType = "tibble" of `get_trackdata()`. See https://ips-lmu.github.io/The-EMU-SDMS-Manual/recipe-plottingSnippets.html for an alternative plotting routines using ggplot2.

Usage

```
dplot(
  x,
  labs = NULL,
  offset = 0,
  prop = TRUE,
  average = FALSE,
  xlim = NULL,
  ylim = NULL,
  lty = FALSE,
  normalise = FALSE,
  colour = TRUE,
  lwd = NULL,
  pch = NULL,
  legend = "topright",
  axes = TRUE,
  type = "l",
  n = 20,
  ...
)
```

Arguments

x	An EMU-trackdata object
labs	A label vector with one element for each row in 'dataset'
offset	Either: A single numeric vector between 0 and 1. 0 and 1 denote synchronize the trackdata at their temporal onsets and offsets respectively; 0.5 denotes synchronization at the temporal midpoint, etc. Or a numeric vector of the same length as x specifying the synchronisation point per segment
prop	A single element character vector specifying whether the tracks should be aligned proportionally or relative to millisecond times. Defaults to proportional alignment
average	If TRUE, the data for each unique label in 'labs' is averaged

xlim	A vector of two numeric values specifying the x-axis range
ylim	A vector of two numeric values specifying the y-axis range
lty	A single element logical vector. Defaults to FALSE. If TRUE, plot each label type in a different linetype
normalise	If TRUE, the data for each segment is linearly time normalised so that all observations have the same length. The number of points used in the linear time normalisation is control by the argument n.
colour	A single element logical vector. Defaults to TRUE to plot each label type in a different colour
lwd	A code passed to the lwd argument in plotting functions. 'lwd' can be either a single element numeric vector, or its length must be equal to the number of unique types in labs. For example, if lwd=3 and if labs = c("a", "b", "a", "c"), then the output is c(3, 3, 3, 3). Alternatively, if lwd = c(2,3,1), then the output is c(2, 3, 2, 1) for the same example. The default is NULL in which case all lines are drawn with lwd=1
pch	A code passed to the pch argument in plotting functions. Functions in the same way as lwd above
legend	Either a character vector to plot the legend. Possible values are: "bottomright"', '"bottom"', '"bottomleft"', '"left"', '"topleft"', '"top"', '"topright"', '"right"' and '"center"'. This places the legend on the inside of the plot frame at the given location. Partial argument matching is used. Or a logical vector: legend = FALSE suppresses legend plotting. legend = TRUE plots it at the default, legend = "topright"
axes	A single element logical vector. Defaults to TRUE to plot the axes
type	The default line type. Default to "l" for a line plot
n	A single element numeric vector. Only used if normalise=TRUE. The number of data points used to linearly time normalise each track
...	graphical options par

Author(s)

Jonathan Harrington

See Also

dcut get_trackdata

Examples

```
# Plot of column 1 (which happens to be the 1st formant) of an EMU-trackdata object
dplot(dip.fdat[,1])


# As above but only observations 1 to 5
```

```
dplot(dip.fdat[1:5,1])


#  column 2 (which happens to be of the second formant) and colour-coded
# for each label-type
dplot(dip.fdat[,2], dip.l)


# put the legend bottom left
dplot(dip.fdat[,2], dip.l, legend="bottomleft")


# as above with no legend and averaged per category
dplot(dip.fdat[,2], dip.l, legend=FALSE, average=TRUE)


# both formants averaged
dplot(dip.fdat[,1:2], dip.l, average=TRUE)


# F2 only with linear-time normalisation
dplot(dip.fdat[,2], dip.l, norm=TRUE)


# linear time-normalisation, both formants and averaged
dplot(dip.fdat[,1:2], dip.l, norm=TRUE, average=TRUE)


# synchronise at the temporal midpoint before averaging, F2 only
dplot(dip.fdat[,2], dip.l, offset=0.5, average=TRUE)


# synchronise 60 ms before the diphthong offset
dplot(dip.fdat[,2], dip.l, offset=dip.fdat$ftime[,2]-60, prop=FALSE)


   # as above averaged, no colour with linetype,
# different plot symbols double line thickness in the range between +- 20 ms
   dplot(dip.fdat[,2], dip.l, offset=dip.fdat$ftime[,2]-60, prop=FALSE,
   average=TRUE, colour=FALSE, lty=TRUE, pch=1:3, lwd =2, type="b", xlim=c(-20, 20))
```

---

dsmooth *Smooth the data in a trackdata object.*

Description

Smooths each dataset in a trackdata object using a running mean smoother.

Usage

```
dsmooth(dataset)
```

Arguments

dataset	A trackdata object as returned from track.

Details

This function uses the dapply function to apply smooth to the data for each segment.

Value

The result of applying the smooth function to each column of the data for each segment in the trackdata object.

See Also

smooth, dapply

---

duplicate_level *Duplicate level*

Description

Duplicate level of emuDB including all of its items and its various attributeDefinitions. If the duplicateLinks variable is set to TRUE all the links to and from the original items are duplicated.

Usage

```
duplicate_level(
  emuDBhandle,
  levelName,
  duplicateLevelName,
  duplicateLinks = TRUE,
  linkDuplicates = FALSE,
  linkDefType = "ONE_TO_ONE",
  verbose = TRUE
)
```

Arguments

emuDBhandle	emuDB handle object (see load_emuDB)
levelName	name of level in emuDB that is to be duplicated
duplicateLevelName	name given to newly duplicated level

duplicateLinks	if set to TRUE (the default) all the links to and from the original items are duplicated to point to the new items of the new duplicate level.
linkDuplicates	link the duplicated ITEMs to the originals. This can only be set to TRUE if duplicateLinks is set to FALSE.
linkDefType	type given to link definition. Only relevant if linkDuplicates is set to TRUE.
verbose	show progress bars and further information

See Also

load_emuDB

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# duplicate Phonetic level
duplicate_level(ae, levelName = "Phonetic",
                    duplicateLevelName = "Phonetic2")


## End(Not run)
```

---

dur *duration*

Description

calculates durations

Usage

```
dur(x)
```

Arguments

x	???

e.dft — *Spectral vector of a single E vowel produced by a male speaker of Standard North German.*

Description

An EMU dataset

Format

spectral vector

ellipse — *Calculate ellipse coordinates*

Description

Calculates ellipse coordinates for eplot

Usage

```
ellipse(x, y, rx, ry, orient, incr = 360/100)
```

Arguments

x	X coordinate of center
y	y coordinate of center
rx	Radius in the x direction
ry	Radius in the y direction
orient	Orientation, in radians. The angle of the major axis to the x axis.
incr	The increment between points, in degrees.

Value

A matrix of x and y coordinates for the ellipse.

See Also

eplot

`emuRsegs` *emuR segment list*

Description

An emuR segment list is a list of segment descriptors. Each segment descriptor describes a sequence of annotation elements. The list is usually a result of an emuDB query using function `query`.

Format

Attributed data.frame, one row per segment descriptor.

Data frame columns are:

- labels: sequenced labels of segment concatenated by '->'
- start: onset time in milliseconds
- end: offset time in milliseconds
- session: session name
- bundle: bundle name
- level: level name
- type: type of "segment" row: 'ITEM': symbolic item, 'EVENT': event item, 'SEGMENT': segment

Additional hidden columns:

- utts: utterance name (for compatibility to emusegs class)
- db_uuid: UUID of emuDB
- startItemID: item ID of first element of sequence
- endItemID: item ID of last element of sequence
- sampleStart: start sample position
- sampleEnd: end sample position
- sampleRate: sample rate

Attributes:

- database: name of emuDB
- query: Query string
- type: type ('segment' or 'event') (for compatibility to emusegs class)

Details

Each row shows the annotation label sequence, the start and end position in time, session and bundle names, level name and type. Additionally the row contains the UUID of the emuDB, the ID's of start and end elements and the corresponding start and end position as sample count and the sample rate. These columns are not printed by default. The print method of emuRsegs hides them. To print all columns of a segment list object use the print method of `data.frame`. For example to print all columns of an emuRsegs segmentlist `sl` type: `print.data.frame(sl)` Though the segment descriptors have references to the annotations, the label and sample/time position information is not updated if any of them change. The values of the segment list may get invalid if the the database is modified. A segment may consist only of one single element, in this case start and end ID are equal. An emuR segment list is the default result of `query` and can be used to get track data using `get_trackdata`. The emuRsegs class inherits emusegs and hence `data.frame`

See Also

`query`,`get_trackdata`,emusegs

`emuRtrackdata` *emuR track data object*

Description

A emuR track data object is the result of `get_trackdata` if the `resultType` parameter is set to `"emuRtrackdata"` or the result of an explicit call to `create_emuRtrackdata`. Compared to the `trackdata` object it is a sub-class of a `data.frame` which is meant to ease integration with other packages for further processing. It can be viewed as an amalgamation of a `emuRsegs` and a `trackdata` object as it contains the information stored in both objects.

Format

The `data.frame` has the following columns:

$sl_rowIdx column to indicate `emuRsegs` row index that the value belongs to

$labels - $sampleRate duplicated information of `emuRsegs` row entries

$times_rel relative time stamps of sample values in milliseconds

$times_orig absolute time stamps of sample values in milliseconds

$T1 - $TN actual data values (e.g. formant values / F0 values / DFT values / ...)

Note that $labels - $sampleRate as well as $T1 - $TN (where the N in TN is to be read as the n-th T value) refer to multiple columns of the object.

Methods

The following methods are implemented for emuRtrackdata objects:

cut Function to extract a `emuRtrackdata` object from an emuRtrackdata at a single time point or between two times

See Also

get_trackdata, create_emuRtrackdata

trackdata

`engassim` *Segment list of a sequence of syllable final n or N preceding k or g , isolated words single speaker, Australian English female from database epgassim.*

Description

An EMU dataset

Format

segmentlist

`engassim.epg` *EPG-compressed trackdata from the segment list engassim*

Description

An EMU dataset

Format

trackdata object

`engassim.l` *Vector of phonetic labels from the segment list engassim: nK = nk,ng , sK = sk,sg*

Description

An EMU dataset

Format

vector of phonetic labels

engassim.w *Vector of word labels from the segment list engassim.*

Description

An EMU dataset

Format

vector of word labels

epgai *Electropalatographic contact indices*

Description

epgai(), epgci(), epgdi() return the anteriority index, the centrality index, the dorsopalatal index respectively as a trackdata object or a vector

Usage

```
epgai(epgdata, weights = c(1, 9, 81, 729, 4921))
```

Arguments

epgdata	An eight-columned EPG-compressed trackdata object, or an eight columned matrix of EPG-compressed trackdata, or a 3D palatographic array that is the output of palate()
weights	A vector of five values that are applied to EPG rows 1-5 respectively in epgai(). A vector of four values that are applied to columns 1 and 8, to columns 2 and 7, columns 3 and 6, columns 4 and 5 respectively. Defaults to the values given in Recasens & Pallares (2001).

Details

These are exact implementations of the formulae for calculating the EPG anteriority, EPG centrality, and EPG dorsopalatal indices as described in Recasens & Pallares (2001).

Value

These functions return a trackdata object if they are applied to an eight-columned EPG-compressed trackdata object, otherwise a one-columned matrix.

Author(s)

Jonathan Harrington

References

GIBBON, F. AND NICOLAIDIS, K. (1999). Palatography. In W.J. Hardcastle & N. Hewlett (eds). Coarticulation. (pp. 229-245). Cambridge University Press: Cambridge.

RECASENS, D. & PALLARES, M. (2001) Coarticulation, assimilation and blending in Catalan consonant clusters. Journal of Phonetics, 29, 273-301.

See Also

epgcog epggs palate

Examples

```
#  Anteriority index: trackdata
ai <- epgai(coutts.epg)
#  Dorsopalatal index, one-columned matrix
di <- epgdi(dcut(coutts.epg, 0.5, prop=TRUE))
# Next to examples: Centrality  index, one-columed matrix
ci <- epgci(palate(coutts.epg))
ci <- epgci(palate(dcut(coutts.epg, 0.5, prop=TRUE)))
```

---

epgcog *Electropalatographic centre of gravity*

Description

Calculate the centre of gravity in palatographic data.

Usage

```
epgcog(
  epgdata,
  weights = seq(7.5, 0.5, by = -1),
  rows = 1:8,
  columns = 1:8,
  row1 = NULL
)
```

Arguments

epgdata	An eight-columned EPG-compressed trackdata object, or an eight columned matrix of EPG-compressed trackdata, or a 3D palatographic array that is the output of palate()
weights	A vector of 8 values that are applied to EPG rows 1-8 respectively. Defaults to 7.5, 7.0, 6.5...0.5.

rows	Calculate EPG-COG over selected row number(s). rows = 5:8, columns = 3:6 is an implementation of posterior centre of gravity, as defined by Gibbon & Nicolaidis (1999,p. 239). See examples below.
columns	Calculate EPG-COG over selected column number(s).
row1	an optional single valued numeric vector to allow a separate weighting of the electrodes in row1. For example, if row1=4/3, then all the electrodes in row1 are multiplied by that value, before EPG-COG is calculated. Defaults to NULL (no weighting).

Details

The centre of gravity is a key function in palatographic research and gives an value per palate that is indicative of the overall location of contacts along the anterior-posterior dimension. The formula is an implementation of the ones discussed in Hardcastle et al. (1991), Gibbon et al (1993), and Gibbon & Nicolaidis (1999).

Value

These functions return a trackdata object if they are applied to an eight-columned EPG-compressed trackdata object, otherwise a one-columned matrix.

Author(s)

Jonathan Harrington

References

GIBBON, F., HARDCASTLE, W. and NICOLAIDIS, K. (1993) Temporal and spatial aspects of lingual coarticulation in /kl/ sequences: a cross-linguistic investigation. Language & Speech, 36, 26t1-277.

GIBBON, F. AND NICOLAIDIS, K. (1999). Palatography. In W.J. Hardcastle & N. Hewlett (eds). Coarticulation. (pp. 229-245). Cambridge University Press: Cambridge.

HARDCASTLE, W, GIBBON, F. and NICOLAIDIS, K. (1991) EPG data reduction methods and their implications for studies of lingual coarticulation. Journal of Phonetics, 19, 251-266.

See Also

epgai epgsum palate

Examples

```
#  COG: trackdata
cog <- epgcog(coutts.epg)
#  cog, one-columned matrix
cog <- epgcog(dcut(coutts.epg, 0.5, prop=TRUE))
# posterior cog for Fig. 10.5, p. 239 in Gibbon & Nicolaidis (1999)
r = array(0, c(8, 8, 2))
r[6,c(1, 8),1] <- 1
```

```
r[7,c(1, 2, 7, 8), 1] <- 1
r[8, ,1] <- 1
r[4, c(1, 2, 8), 2] <- 1
r[5, c(1, 2, 7, 8), 2] <- 1
r[6, c(1, 2, 3, 7, 8), 2] <- 1
r[7:8, , 2] = 1
class(r) <- "EPG"
epgcog(r, rows=5:8, columns=3:6)
```

---

epggs *Plot a grey-scale image of palatographic data.*

Description

The function plots a grey-scale image of palatographic data such that the greyness in cell r, c is in proportion to the frequency of contacts in cells of row r and columns c of all palatograms in the object passed to this function.

Usage

```
epggs(
  epgdata,
  gscale = 100,
  gridlines = TRUE,
  gridcol = "gray",
  gridlty = 1,
  axes = TRUE,
  xlab = "",
  ylab = "",
  ...
)
```

Arguments

epgdata	An eight-columned EPG-compressed trackdata object, or an eight columned matrix of EPG-compressed trackdata, or a 3D palatographic array that is the output of palate()
gscale	a single valued numeric vector that defines the granularity of the greyscale. Defaults to 100.
gridlines	if TRUE (default) grid lines over the palatographic image are drawn are drawn.
gridcol	color of grid
gridlty	A single-valued numeric vector that defines the linetype for plotting the grid.
axes	TRUE for show axes, FALSE for no axes
xlab	A character vector for the x-axis label.
ylab	A character vector for the y-axis label.
...	graphical parameters can be given as arguments to 'epggs'.

Details

The function plots a grey-scale image of up to 62 values arranged over an 8 x 8 grid with columns 1 and 8 unfilled for row 1. If cell row r column c is contacted for all palatograms in the object that is passed to this function, the corresponding cell is black; if none of of the cells in row r column c are contacted, then the cell is white (unfilled).

Author(s)

Jonathan Harrington

See Also

epgai epgcog epgplot palate

Examples

```
# greyscale image across the first two segments 'just relax'
# with title
epggs(coutts.epg[1:2,], main="just relax")

# as above but with dotted gridlines in blue
epggs(coutts.epg[1:2,], main="just relax", gridlty=2, gridcol="blue")

# as the first example, but with greyscale set to 2
epggs(coutts.epg[1:2,], 2, main="just relax")

# get palatograms for "S" from the polhom.epg database
temp = polhom.l == "S"
# greyscale image of all "S" segments at their temporal midpoint
epggs(dcut(polhom.epg[temp,], 0.5, prop=TRUE))

# greyscale image of all "S" segments from their onset to offset
epggs(polhom.epg[temp,])

# the same but derived from palates
p <- palate(polhom.epg[temp,])
epggs(p)
```

---

epgplot *Plot palatographic data*

Description

Function to plot palatograms from EPG compressed objects or from a 3D-palatographic array that is output from palate().

Usage

```
epgplot(
  epgdata,
  select = NULL,
  numbering = "times",
  gridlines = TRUE,
  mfrow = NULL,
  col = 1,
  mar = c(0.8, 0.1, 0.8, 0.1),
  xlim = NULL
)
```

Arguments

epgdata	An eight-columned EPG-compressed trackdata object, or an eight columned matrix of EPG-compressed trackdata, or a 3D palatographic array that is the output of palate()
select	A vector of times. Palatograms are plotted at these times only. Note: this argument should only be used if epgdata is temporally contiguous, i.e. the entire trackdata object contains palatograms at successive multiple times of the EPG sampling frequency. (as in coutts.epg$ftime). Defaults to NULL, in which case palatograms are plotted for all times available in epgdata.
numbering	Either "times" (default), or logical TRUE, or a character vector of the same length as the number of segments in epgdata. In the default case, the times at which the palatograms occur are printed above the palatograms. If logical TRUE, then the palatograms are numbered 1, 2, ... number of segments and this value is printed above the palatograms. If a character vector, then this must be the same length as the number of segments in epgdata.
gridlines	if TRUE (default) grid lines over the palatogram are drawn.
mfrow	By default, the function tries to work out a sensible number of rows and columns for plotting the palatograms. Otherwise, this can be user-specified, in which case mfrow is a vector of two integer numeric values.
col	specify a colour for plotting the filled EPG cells.
mar	A numerical vector of the form 'c(bottom, left, top, right)' which gives the number of lines of margin to be specified on the four sides of the plot. The default in this function is c(0.8, 0.1, 0.8, 0.1). (The default in the R plot() function is c(5, 4, 4, 2) + 0.1.
xlim	A numeric vector of two time values over which the epgdata should be plotted. Note: this argument should only be used if epgdata is temporally contiguous, i.e. the entire trackdata object contains palatograms at successive multiple times of the EPG sampling frequency. (as in coutts.epg$ftime). Defaults to NULL (plot all time values).

Details

The function plots 62 values arranged over an 8 x 8 grid with columns 1 and 8 unfilled for row 1. When there is a contact (1), the corresponding rectangle of the grid is filled otherwise the rectangle

is empty.

Author(s)

Jonathan Harrington

See Also

epgai epgcog epggs palate

Examples

```
epgplot(polhom.epg[10,])

# as above but between times 1295 ms and 1330 ms
epgplot(polhom.epg[10,], xlim=c(1295, 1330))

# the same as above, but the data is first
# converted to a 3D palatographic array
p <- palate(polhom.epg[10,])
epgplot(p, xlim=c(1295, 1330))

# plot palatograms 2 and 8
epgplot(p[,,c(2, 8)])

# as above but
# no gridlines, different colour, numbering rather than times
epgplot(p[,,c(2, 8)], gridlines=FALSE, col="pink", numbering=TRUE)

# as above but with a user-specified title

epgplot(p[,,c(2, 8)], gridlines=FALSE, col="pink", numbering=c("s1", "s2"))

# plot the palatograms in the second
# segment of coutts.epg that are closest in time
# to 16377 ms and 16633 ms
epgplot(coutts.epg[2,], c(16377, 16633))
```

---

epgsum *Sum contacts in palatograms.*

Description

The function calculates EPG contact profiles, i.e. sums active or inactive electrodes optionally by row and/or column in palatographic data.

Usage

```
epgsum(
  epgdata,
  profile = c(1, 3),
  inactive = FALSE,
  rows = 1:8,
  columns = 1:8,
  trackname = "EPG-sum"
)
```

Arguments

epgdata	An eight-columned EPG-compressed trackdata object, or an eight columned matrix of EPG-compressed trackdata, or a 3D palatographic array that is the output of palate()
profile	A numeric vector of one or two values. The options are as follows. c(1,3) and c(1) sum the contacts by row, but the latter outputs the summation in the rows. c(2,3) and c(2) sum the contacts by column, but the latter outputs the summation in the columns. (see also rows and columns arguments and the examples below for further details).
inactive	a single element logical vector. If FALSE (the default), then the active electrodes (i.e, 1s) are summed, otherwise the inactive electrodes (i.e., 0s) are summed.
rows	vector of rows to sum
columns	vector of columns to sum
trackname	single element character vector of the name of the track (defaults to "EPG-sum")

Details

Contact profiles are standard tools in electropalatographic analysis. See e.g., Byrd (1996) for details.

Value

These functions return a trackdata object if they are applied to an eight-columned EPG-compressed trackdata object, otherwise a one-columned matrix.

Author(s)

Jonathan Harrington

References

BYRD, D. (1996). Influences on articulatory timing in consonant sequences. Journal of Phonetics, 24, 209-244.

GIBBON, F. AND NICOLAIDIS, K. (1999). Palatography. In W.J. Hardcastle & N. Hewlett (eds). Coarticulation. (pp. 229-245). Cambridge University Press: Cambridge.

See Also

epgai epgcog epggs palate

Examples

```
# Trackdata object of the sum of contacts in the 1st segment of polhom.epg
epgsum(polhom.epg[1,])
# as above, but the summation is in rows 1-3 only.
epgsum(polhom.epg[1,], rows=c(1:3))
# as epgsum(polhom.epg[1,]), except sum the inactive electrodes in columns 3-6.
epgsum(polhom.epg[1,], columns=3:6, inactive=TRUE)
# Obtain compressed EPG-trackdata object for the 1st four segments of polhom.epg
# at the temporal midpoint
mid <- dcut(polhom.epg[1:4,], .5, prop=TRUE)
# sum of contacts in these four palatograms.
epgsum(mid)
# gives the same result as the previous command.
p <- palate(mid)
# sum the contacts in the palatograms.
epgsum(p)
# as above, but show the separate row summmations.
epgsum(p, 1)
# as above, but show the separate column summmations.
epgsum(p, 2)
# sum of the contacts in rows 1-4 showing the separate row summations.
epgsum(p, 1, rows=1:4)
# sum of the contacts in rows 1-4 showing the separate column summations.
epgsum(p, 2, rows=1:4)
# sum of the contacts in columns 3-6  showing the separate row summations.
epgsum(p, 1, columns=3:6)
# sum of the contacts in columns 3-6  showing the separate column summations.
epgsum(p, 2, columns=3:6)
```

---

eplot *Plot ellipses for two-dimensional data (DEPRECATED see below)*

Description

The function plots ellipses for different categories from two-dimensional data. DEPRECATED as this function does not play well with with the new resultType = "tibble" of `get_trackdata()`. See https://ips-lmu.github.io/The-EMU-SDMS-Manual/recipe-plottingSnippets.html for an alternative plotting routines using ggplot2.

Usage

```
eplot(
  x,
  labs,
  chars,
  formant = FALSE,
  scaling = "linear",
  prob = 0.95,
  nsdev = NULL,
  dopoints = FALSE,
  doellipse = TRUE,
  centroid = FALSE,
  axes = TRUE,
  xlim,
  ylim,
  col = TRUE,
  lty = FALSE,
  lwd = NULL,
  ...
)
```

Arguments

x	A two-columned matrix of data
labs	An optional vector of labels, parallel to 'data'
chars	An optional vector of labels, parallel to 'data'. If this argument is specified these labels will be plotted rather than the labels in 'labs'.
formant	If TRUE) then the data is negated and the axes are switched so that, for formant data, the plot is made with decreasing F2 on the x-axis and decreasing F1 on the y-axis.
scaling	Either "mel" or "bark" for mel or bark scaling of the data
prob	A single numeric vector greater than zero and less than 1 representing the confidence interval of the ellipse contours. Defaults to 0.95
nsdev	Defines the length of the major and minor axes of the ellipses in terms of the standard deviation of the data and overrides the prob argument.
dopoints	If TRUE) character labels (from 'labs' or 'chars') are plotted for each data point
doellipse	If TRUE, ellipses are drawn on the plot. If FALSE, no ellipses are drawn and, if 'dopoints' is also FALSE, 'centroids' is set to TRUE
centroid	One label for each ellipse is drawn
axes	If TRUE axes are drawn on the plot.
xlim	A vector of two numeric values giving the range of the x-axis.
ylim	A vector of two numeric values giving the range of the y-axis.
col	If colour is TRUE) the ellipses and labels will be plotted in different colours

lty	If linetype is TRUE) the ellipses will be plotted with different linetypes. This is useful for plots that will be printed.
lwd	A code passed to the lwd argument in plotting functions. 'lwd' can be either a single element numeric vector, or its length must be equal to the number of unique types in labs. For example, if lwd=3 and if labs = c("a", "b", "a", "c"), then the output is c(3, 3, 3, 3). Alternatively, if lwd = c(2,3,1), then the output is c(2, 3, 2, 1) for the same example. The default is NULL in which case all lines are drawn with lwd=1
...	graphical options par

Author(s)

Jonathan Harrington, Steve Cassidy

See Also

dcut

Examples

```
data(vowlax)
data <- cbind(vowlax.df$F1,vowlax.df$F2)
phonetic = vowlax.l
word = vowlax.word

eplot(data, phonetic)


eplot(data, phonetic, form=TRUE, main="F1 x F2 plane", centroid=TRUE)
eplot(data, phonetic, form=TRUE, main="F1 x F2 plane", dopoints=TRUE)
eplot(data, phonetic, form=TRUE, main="F1 x F2 plane in Bark",
      dopoints=TRUE, scaling="bark")
eplot(data, phonetic, form=TRUE, main="F1 x F2 plane in Bark b/w with linetype",
      col=FALSE, lty=TRUE, dopoints=TRUE, scaling="bark")
eplot(data, phonetic, form=TRUE, main="F1 x F2 plane",
      doellipse=FALSE, dopoints=TRUE)
eplot(data, phonetic, form=TRUE, dopoints=TRUE,
      prob=0.5, main="F1 x F2 plane, 50% confidence intervals")
eplot(data, phonetic, form=TRUE, dopoints=TRUE,
      nsdev=2, main="F1 x F2 plane, 2 standard deviations")


temp <- phonetic %in% c("a", "O")
eplot(data[temp,], phonetic[temp], form=TRUE,  main="F1 x F2 [A] and [O] only", centroid=TRUE)


temp <- phonetic=="O"
eplot(data[temp,], phonetic[temp], word[temp], form=TRUE,
      dopoints=TRUE, main="[O] only showing word labels")
```

`euclidean` *Find the inter-euclidean distance for a data matrix*

Description

Finds the inter-euclidean distance for a data matrix

Usage

```
euclidean(data, m = 1, n = ncol(data))
```

Arguments

`data`	A vector or matrix of numerical data.
`m`	The first column of data to be used in the distance calculation.
`n`	The last column of data to be used in the distance calculation.

Value

Calculates the euclidean distance between successive rows of the matrix based on columns m:n.

See Also

steady

Examples

```
euclidean(cbind(c(1,2,3,4), c(2,3,2,2)))
```

expand_labels *Label each data sample*

Description

Labels each data sample

Usage

```
expand_labels(indvals, labs)
```

Arguments

indvals	Index component of a trackdata object as returned by frames, or track.
labs	A label vector parallel to indvals.

Value

Returns a vector of labels, one for each row in the data matrix that corresponds to indvals.

See Also

frames, track

export_BPFCollection *Exports an emuDB into a BAS Partitur File (BPF) Collection*

Description

This function exports an emuDB into the BAS Partitur File format, with one BPF file per bundle. The user must pass a list of matching label names and BPF keys. **Important:** The BPF format does not support explicit hierarchies with more than three levels. Hence, you will probably lose information when exporting complex hierarchies.

Usage

```
export_BPFCollection(
  handle,
  targetDir,
  extractLevels,
  refLevel = NULL,
  verbose = TRUE,
  newLevels = NULL,
  newLevelClasses = NULL,
  copyAudio = FALSE
)
```

Arguments

handle	handle to the emuDB
targetDir	directory where the BPF collection should be saved
extractLevels	list containing the names of labels (not levels!) that should be extracted, and their matching BPF keys, e.g. extractLevels = list(SampleRate="SAM", Text="ORT", Phonemes="SAP")
refLevel	optional name of level (not label!) used as reference for symbolic links. If NULL (the default), a link-less BPF collection is created.
verbose	display infos, warnings and show progress bar
newLevels	optional vector containing names of levels in the BPF collection that are not part of the standard BPF levels. See http://www.bas.uni-muenchen.de/forschung/Bas/BasFormatseng.html#Partitur_tiersdef for details on standard BPF levels.
newLevelClasses	optional vector containing the classes of levels in the newLevels vector as integers. Must have the same length and order as newLevels.
copyAudio	if true, audio files are copied to the new BPF collection

See Also

export_TextGridCollection

---

export_seglistToTxtCollection

Exports a segment list to txt collection

Description

Extract the media file (usually .wav file) snippets that correspond to the segments of a segment list (see result of a query) and save them to separate files and write the corresponding labels into a .txt file. Further, the segmentlist is also stored to the target directory (as a .csv file).

Usage

```
export_seglistToTxtCollection(emuDBhandle, seglist, targetDir)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
seglist	tibble, emuRsegs or emusegs object obtained by querying a loaded emuDB
targetDir	target directory to store

`export_TextGridCollection` *Export annotations of emuDB to TextGrid collection*

Description

Exports the annotations of an emuDB to a TextGrid collection (.TextGrid and .wav file pairs). To avoid naming conflicts and not to loose the session information, the session structure of the database is kept in place (i.e. the TextGrid collection will have sub-folders that are named as the sessions were). Due to the more complex annotation structure modeling capabilities of the EMU-SDMS system, this export routine has to make several compromises on export which can lead to information loss. So use with caution and at own risk as reimporting the exported data will mean that not all information can be recreated! The main compromises are:

- If a MANY_TO_MANY relationship between two levels is present and two items from the parent level are linked to a single item on the child level, the concatenated using the '->' symbol. An example would be: the annotation items containing the labels 'd' and 'b' of the parent items are merged into a single annotation item and their labels are Phoneme level are linked to 'db' on the Phonetic level. The generated Phoneme tier then has a segment with the start and end times of the 'db' item and contains the labels 'db' (see for example the bundle 0000_ses/msajc010_bndl of the ae_emuDB).
- As annotations can contain gaps (e.g. incomplete hierarchies or orphaned items) and do not have to start at time 0 and be the length of the audio file this export routine pads these gaps with empty segments.

Usage

```
export_TextGridCollection(
  emuDBhandle,
  targetDir,
  sessionPattern = ".*",
  bundlePattern = ".*",
  attributeDefinitionNames = NULL,
  timeRefSegmentLevel = NULL,
  verbose = TRUE
)
```

Arguments

`emuDBhandle`	emuDB handle object (see load_emuDB)
`targetDir`	directory where the TextGrid collection should be saved
`sessionPattern`	A regular expression pattern matching session names to be exported from the database
`bundlePattern`	A regular expression pattern matching bundle names to be exported from the database

attributeDefinitionNames
: list of names of attributeDefinitions that are to be exported as tiers. If set to NULL (the default) all attribute definitions will be exported as separate tiers.

timeRefSegmentLevel
: parameter passed into query function. (set time segment level from which to derive time information. It is only necessary to set this parameter if more than one child level contains time information and the queried parent level is of type ITEM.)

verbose
: Show progress bars and further information

See Also

load_emuDB

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

## Export all levels
export_TextGridCollection(ae, "/path/2/targetDir")


## End(Not run)
```

---

fapply *Function that applies a function to an EMU spectral object*

Description

Applies a function to an EMU spectral object.

Usage

```
fapply(specdata, fun, ..., power = FALSE, powcoeffs = c(10, 10))
```

Arguments

specdata
: A matrix or trackdata object of class spectral

fun
: A function to be applied.

...
: Optional arguments to fun

power
: A single element logical vector. If TRUE, convert specdata to power values i.e. apply the function to a * specdata ^b or a * specdata$data ^b where a and b powcoeffs defined below.

powcoeffs A 2 element numeric vector for converting dB values to power values. Defaults to a = 10 and b = 10. See power.

Details

fapply performs a similar operation to apply except that it is specifically designed for handling EMU spectral objects.

Value

If the output has the same dimensions has the input, then an object of the same dimensionality and class is returned. Otherwise it may be a vector or matrix depending on the function that is applied.
...

Warning

The function can be very slow if applied to a large trackdata object. In this case, it may be faster to use a for-loop with the desired function around $data

Author(s)

Jonathan Harrington

See Also

apply by.trackdata

Examples

```
# mean value per spectrum, input is a spectral matrix
m <- fapply(vowlax.dft.5, sapply, FUN=mean)
# as above but after converting dB to powers before
# applying the function
m <- fapply(vowlax.dft.5, sapply, FUN=mean, power=TRUE)
# spectral range
r <- fapply(vowlax.dft.5, range)
# spectral moments applied to a trackdata object
# m is a four-dimensional trackdata object
m <- fapply(fric.dft, moments)
# 1st 3 DCT coefficients calculated in a spectral matrix
# d is a 3-columned matrix
d <- fapply(vowlax.dft.5, dct, 3)
# dct-smooth with 10 coefficients. d2 is spectral matrix
d2 <- fapply(vowlax.dft.5, dct, 10, TRUE)
# dct-smooth a trackdata object with 10 coefficients
d3 <- fapply(fric.dft[1:4,], dct, 10, TRUE)
```

frames *frames*

Description

Get frames from trackdata objects

Usage

```
frames(trackdata)
```

Arguments

trackdata	an object of class trackdata

Value

Data frames from the input object.

Author(s)

Jonathan Harrington

See Also

trackdata

frames.time *Find the time and position of a data element.*

Description

Finds the time and position of a data element.

Usage

```
frames.time(dataset, datanum)
```

Arguments

dataset	A dataset returned by track or frames.
datanum	An integer, an index into the data component of dataset.

Details

The dataset returned from `track` or `frames` consists of a matrix of data (the `data` component) and two index components (`index` and `ftime`). The data for all segments is concatenated together in `$data`. This function can be used to find out which segment a particular row of `$data` corresponds to.

Value

The segment number which contains the element `datanum` of `dataset$data`.

See Also

track, frames

`freqtoint` *Function to find the column number corresponding to frequencies of a spectral object*

Description

Find the column number corresponding to frequencies of a spectral object.

Usage

```
freqtoint(trackdata, j)
```

Arguments

`trackdata`	A spectral object
`j`	A vector of frequencies

Details

This function is used in conjunction with object oriented programming of EMU spectral objects. It should not in general be called from inside a function. Its principal use is to determine the column number(s) corresponding to frequencies for spectral trackdata objects or spectral matrices or the element number for spectral vectors.

Author(s)

Jonathan Harrington

Examples

```
freqtoint(fric.dft,1000:2000)
# all frequencies except 1000-2000
freqtoint(vowlax.dft.5, -(1000:2000))
# all frequencies except 1000 Hz
freqtoint(e.dft, -1000)
# the d.c. offset - i.e. column 1
freqtoint(vowlax.dft.5, 0)
# all freqs except the d.c. offset - i.e. not column 1
freqtoint(vowlax.dft.5, -1)
```

`fric` *Segment list of word-medial s or z one male speaker of Standard North German, read speech from database kielread.*

Description

An EMU dataset

Format

segmentlist

`fric.dft` *Spectral trackdata object from the segment list fric.*

Description

An EMU dataset

Format

trackdata object

fric.l — *Vector of labels from the segment list fric*

Description

An EMU dataset

Format

vector of labels

fric.w — *Vector of word labels from the segment list fric.*

Description

An EMU dataset

Format

vector of word labels

get.time.element — *Get data for a given time*

Description

Gets data for a given time

Usage

```
get.time.element(timeval, dataset)
```

Arguments

timeval	A time in milliseconds
dataset	A trackdata object as returned by track.

Value

The element number of trackdata$data corresponding to time

See Also

track, frames

`get_trackdata` *Get trackdata from loaded emuDB*

Description

Extract trackdata information from a loaded emuDB that corresponds to the entries in a segment list.

Usage

```
get_trackdata(
  emuDBhandle,
  seglist = NULL,
  ssffTrackName = NULL,
  cut = NULL,
  npoints = NULL,
  onTheFlyFunctionName = NULL,
  onTheFlyParams = NULL,
  onTheFlyOptLogFilePath = NULL,
  onTheFlyFunction = NULL,
  resultType = "tibble",
  consistentOutputType = TRUE,
  verbose = TRUE
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`seglist`	`tibble`, `emuRsegs` or `emusegs` object obtained by `query`ing a loaded emuDB
`ssffTrackName`	The name of track that one wishes to extract (see `list_ssffTrackDefinitions` for the defined ssffTracks of the emuDB). If the parameter `onTheFlyFunctionName` is set, then this corresponds to the column name af the AsspDataObj (see `wrassp::wrasspOutputInfos[[` and `wrasspOutputInfos` - NOTE: `library(wrassp)` might be necessary to access the `wrasspOutputInfos` object without the `wrassp::` prefix). If the parameter `onTheFlyFunctionName` is set and this one isn't, then per default the first track listed in the `wrassp::wrasspOutputInfos` is chosen (`wrassp::wrasspOutputInfos[[onTheFlyF` `get_trackdata` has so called constant track names that are always available for every emuDB. The constant track names are: • "MEDIAFILE_SAMPLES": refers to the audio sample values specified by the "mediafileExtension" entry of the DBconfig.json
`cut`	An optional cut time for segment data, ranges between 0 and 1, a value of 0.5 will extract data only at the segment midpoint.
`npoints`	An optional number of points to retrieve for each segment or event. For segments this requires the `cut` parameter to be set; if this is the case, then data is extracted around the resulting cut time. For events data is extracted around the

event time. If npoints is an odd number, the samples are centered around the cut-time-sample; if not, they are skewed to the right by one sample.

`onTheFlyFunctionName`
: Name of wrassp function that will perform the on-the-fly calculation (see `?wrassp` for a list of all the signal processing functions wrassp provides)

`onTheFlyParams`
: A `pairlist` of parameters that will be given to the function passed in by the `onTheFlyFunctionName` parameter. This list can easily be generated by applying the `formals` function to the on-the-fly function name and then setting the according parameter one wishes to change.

`onTheFlyOptLogFilePath`
: Path to optional log file for on-the-fly function

`onTheFlyFunction`
: pass in a function pointer. This function will be called with the path to the current media file. It is required that the function returns a tibble/data.frame like object that contains a column called `frame_time` that specifies the time point of each row. `get_trackdata` will then extract the rows belonging to the current segment. This allows users to code their own function to be used with `get_trackdata` and allows for most data formats to be used within an emuDB.

`resultType`
: Specify class of returned object. Either `"emuRtrackdata"`, `"trackdata"` or `"tibble"` == the default (see `trackdata`, `emuRtrackdata` and `tibble` for details about these objects).

`consistentOutputType`
: Prevent converting the output object to a `data.frame` depending on the `npoint` and `cut` arguments (only applies to output type "trackdata"). Set to `FALSE` if the following legacy `emu.track` output conversion behaviour is desired: If the `cut` parameter is not set (the default) an object of type `trackdata` is returned. If `cut` is set and `npoints` is not, or the seglist is of type event and npoints is not set, a `data.frame` is returned (see the `consistentOutputType` to change this behaviour).

`verbose`
: Show progress bars and further information

Details

This function utilizes the wrassp package for signal processing and SSFF/audio file handling. It reads time relevant data from a given segment list (`emuRsegs` or `emusegs`), extracts the specified trackdata and places it into a trackdata object (analogous to the deprecated `emu.track`). This function replaces the deprecated `emu.track` function. Note that an warning is issued if the bundles in the `emuRsegs` or `emusegs` object have in-homogeneous sampling rates as this could lead to inconsistent/erroneous `trackdata`, `emuRtrackdata` or `tibble` result objects. For more information on the structural elements of an emuDB see the signal data extraction chapter of the EMU-SDMS manual (`https://ips-lmu.github.io/The-EMU-SDMS-Manual/chap-sigDataExtr.html`).

Value

object of type specified with `resultType`

See Also

formals, wrasspOutputInfos, trackdata, emuRtrackdata

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# query loaded "ae" emuDB for all "i:" segments of the "Phonetic" level
sl = query(emuDBhandle = ae,
           query = "Phonetic == i:")

# get the corresponding formant trackdata
td = get_trackdata(emuDBhandle = ae,
                   seglist = sl,
                   ssffTrackName = "fm")

# get the corresponding F0 trackdata
# as there is no F0 ssffTrack defined in the "ae" emuDB we will
# calculate the necessary values on-the-fly
td = get_trackdata(emuDBhandle = ae,
                   seglist = sl,
                   onTheFlyFunctionName = "ksvF0")


## End(Not run)
```

---

import_mediaFiles *Import media files to emuDB*

Description

Import new recordings (media files) to emuDB and create bundles. Looks for files with the defined mediafile extension of the emuDB (see `mediaFileExtension` in vignette `emuDB`) in `dir` or in subdirectories thereof (interpreted as sessions), for each mediafile create a bundle directory named as the basename of the mediafile in the specified session, and copies the mediafile into the bundle. If not already present, adds 'OSCI' and 'SPEC' perspectives to the emuDB config file.

Usage

```
import_mediaFiles(emuDBhandle, dir, targetSessionName = "0000", verbose = TRUE)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
dir	directory containing mediafiles or session directories
targetSessionName	name of session in which to create the new bundles
verbose	display infos & show progress bar

Examples

```
## Not run:
## Add mediafiles from directory

 import_mediaFiles(myEmuDB,dir="/data/mymedia/")


## End(Not run)
```

---

is.spectral *Function to test whether the object is of class "spectral"*

Description

Returns TRUE or FALSE depending on whether the object is of class "spectral"

Usage

```
is.spectral(dat)
```

Arguments

dat	An R object

Value

A single element logical vector: TRUE or FALSE

Author(s)

Jonathan Harrington

See Also

as.spectral

Examples

```
is.spectral(vowlax.dft.5)
is.spectral(fric.dft)
is.spectral(fric.dft$data)
is.spectral(vowlax.dft.5[1,])
is.spectral(fric.dft[1,1])
```

`is.trackdata` *Test whether an object is an Emu trackdata object*

Description

Test whether an object is an Emu trackdata object

Usage

```
is.trackdata(object)
```

Arguments

`object`	A data object to be tested

Value

Returns TRUE if the argument is a trackdata object.

See Also

`get_trackdata`

`isol` *Segment list of vowels in a d d context isolated word speech, one male speaker of Australian English from database isolated.*

Description

An EMU dataset

Format

segmentlist

isol.fdat *Trackdata of formants from the segment list isol*

Description

An EMU dataset

Format

trackdata object

isol.l *Vector of vowel phoneme labels from the segment list isol*

Description

An EMU dataset

Format

vector of vowel phoneme labels

label *Get labels / utterances from segment list*

Description

label: extracts the labels from the segment list. utt: extracts the utterances from the segment list.

Usage

```
label(segs)
```

Arguments

segs	segment list

Value

label / utterance vector

Author(s)

Jonathan Harrington

See Also

segmentlist start end

Examples

```
data(dip)
#dip is a segment list - first ten segments only
dip[1:10,]


#extract labels from the segment list
dips.labs = label(dip)
dips.labs
```

---

linear *Perform linear time normalisation on trackdata.*

Description

Performs linear time normalisation on trackdata.

Usage

```
linear(dataset, n = 20)
```

Arguments

dataset	A trackdata object as returned by track.
n	The number of points (samples) required for each segment.

Details

The data for each segment is normalised using the approx function.

Value

A new trackdata object where the data for each segment has the same number (n) of samples.

See Also

approx

`list_bundles` *List bundles of emuDB*

Description

List all bundles of emuDB or of particular session.

Usage

```
list_bundles(
  emuDBhandle,
  session = NULL,
  sessionPattern = ".*",
  bundlePattern = ".*"
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`session`	optional session (depricated!)
`sessionPattern`	A regular expression pattern matching session names to be searched for in the database. Note: "_ses$" is appended to this RegEx automatically
`bundlePattern`	A regular expression pattern matching bundle names to be searched for in the database. Note: "_bndl$" is appended to this RegEx automatically

Value

data.frame object with columns session and name of bundles

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# list bundles of session "0000" of ae emuDB
list_bundles(emuDBhandle = ae,
             session = "0000")


## End(Not run)
```

`list_files` *List files of emuDB*

Description

List files belonging to emuDB. For more information on the structural elements of an emuDB see `vignette{emuDB}`.

Usage

```
list_files(
  emuDBhandle,
  fileExtension = ".*",
  sessionPattern = ".*",
  bundlePattern = ".*"
)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`fileExtension`	file extension of files
`sessionPattern`	A (RegEx) pattern matching sessions to be searched from the database
`bundlePattern`	A (RegEx) pattern matching bundles to be searched from the database

Value

file paths as character vector

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# list all files of ae emuDB
list_files(emuDBhandle = ae)

# list all files of ae emuDB in bundles ending with '3'
list_files(emuDBhandle = ae, bundlePattern=".*3$")


## End(Not run)
```

list_sampleRates *List sample rates of media and annotation (_annot.json) files*

Description

List sample rates of media and annotation (_annot.json) files

Usage

```
list_sampleRates(emuDBhandle, sessionPattern = ".*", bundlePattern = ".*")
```

Arguments

emuDBhandle	emuDB handle object (see load_emuDB)
sessionPattern	A regular expression pattern matching session names to be searched from the database
bundlePattern	A regular expression pattern matching bundle names to be searched from the database

Value

tibble with the columns

- session
- bundle
- sample_rate_annot_json
- sample_rate_media_file

session, b

list_sessions *List sessions of emuDB*

Description

List session names of emuDB

Usage

```
list_sessions(emuDBhandle, sessionPattern = ".*")
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
sessionPattern	A regular expression pattern matching session names to be searched for in the database. Note: "_ses$" is appended to this RegEx automatically

Value

data.frame object with session names

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# list all sessions of ae emuDB
list_sessions(emuDBhandle = ae)


## End(Not run)
```

`load_emuDB` *Load emuDB*

Description

Function loads emuDB into its cached representation and makes it accessible from within the current R session by returning a emuDBhandle object

Usage

```
load_emuDB(
  databaseDir,
  inMemoryCache = FALSE,
  connection = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`databaseDir`	directory of the emuDB
`inMemoryCache`	cache the loaded DB in memory
`connection`	pass in DBI connection to SQL database if you want to override the default which is to use an SQLite database either in memory (`inMemoryCache = TRUE`) or in the emuDB folder. This is intended for expert use only!
`verbose`	be verbose
`...`	additional parameters

Details

In order to access an emuDB from R it is necessary to load the annotation and configuration files to an emuR internal database format. The function expects a emuDB file structure in directory `databaseDir`. The emuDB configuration file is loaded first. On success the function iterates through session and bundle directories and loads found annotation files. The parameter `inMemoryCache` determines where the internal database is stored: If `FALSE` a database cache file in `databaseDir` is used. When the database is loaded for the first time the function will create a new cache file and store the data to it. On subsequent loading of the same database the cache is only updated if files have changed, therefore the loading is then much faster. For this to work the user needs write permissions to `databaseDir` and the cache file. The database is loaded into a volatile in-memory database if `inMemoryCache` is set to `TRUE`.

Value

emuDB handle object

Examples

```
## Not run:
## Load database ae in directory /homes/mylogin/EMUnew/ae
## assuming an existing emuDB structure in this directory

ae = load_emuDB("/homes/mylogin/EMU/ae")

## Load database ae from demo data

# create demo data in temporary directory
create_emuRdemoData(dir = tempdir())
# build base path to demo emuDB
demoDatabaseDir = file.path(tempdir(), "emuR_demoData", "ae_emuDB")

# load demo emuDB
ae = load_emuDB(demoDatabaseDir)


## End(Not run)
```

---

locus *Calculate locus equations for two-dimensional data*

Description

The function plots a locus equation and returns associated statistical information.

Usage

```
locus(
  target,
  onset,
  labels.vow = NULL,
  yxline = TRUE,
  plotgraph = TRUE,
  axes = TRUE,
  ...
)
```

Arguments

target	a numerical vector typically of F2 values at the vowel target
onset	a numerical vector typically of the same length as target of F2 values at the vowel onset
labels.vow	an optionally character vector for plotting labels at the points (target, onset) of the same length as target
yxline	optionally plot the line target = onset. Defaults to True.
plotgraph	a logical vector for specifying whether the data should be plotted. Defaults to True.
axes	A logical vector indicating whether the axes should be plotted
...	graphical options par

Details

A locus equation is a straight line regression fitted with lm() in which the F2- values typically at the vowel onset are regressed on those of the target. The slope can be used to give an indication of target-on-onset coarticulatory influences.

The best estimate of the locus frequency is where the locus equation bisects the line target = onset.

Value

A list containing regression diagnostics of the function lm() that can be accessed with summary() and the estimated locus frequency in $locus. A plot of values in the onset x target plane with superimposed locus equation and line onset=target.

Author(s)

Jonathan Harrington

Examples

```
 # calculate an F2-locus equation for initial [d]
# preceding lax vowels produced by female speaker "68".
```

```
# the onset is taken at the vowel onset; the
# vowel target is taken at the vowel's temporal midpoint.

# identify initial "d" of speaker "68"
temp <- vowlax.left == "d" & vowlax.spkr == "68"
# get the F2 value at the vowel's temporal midpoint
targ <- dcut(vowlax.fdat[temp,2], .5, prop=TRUE)
# F2 value at the vowel's acoustic onset.
on <- dcut(vowlax.fdat[temp,2], 0, prop=TRUE)

# locus equation plot
result <- locus(targ, on, vowlax.l[temp])
# statistical diagnostics of the regression line (locus equation)
summary(result)
# intercept and slope
result$coeff
# best estimate of the locus frequency, i.e. the
# point of bisection of on = TRUEarg with the regression line
result$locus
```

`mahal` *Classify using Mahalanobis distance*

Description

Classifies using Mahalanobis distance

Usage

```
mahal(data, train)
```

Arguments

`data`	A vector or matrix of data
`train`	A Gaussian model generated by `train`.

Details

The `model` argument contains the mean and inverse covariance matrix (or standard deviation if the data is one-dimensional) for each class in the training set as well as the class labels. This function calculates the Mahalanobis distance of each row of `data` from each class mean and assigns the label of the closest mean to that row. The result is a vector of labels corresponding to the rows of `data`.

The Mahalanobis distance between a data point and a class is the Euclidean distance between the point and the class mean divided by the covariance matrix for the class. This means that classes with large covariances will attract data points from a larger area than those with small covariances.

Value

A label vector with one element per row of `data`

References

O'Shaughnessy, D. Speech Communication (Addison-Wesley: Reading, MA. 1987)

See Also

train

`mahal.dist` *Calculate mahalanobis distances*

Description

Calculates mahalanobis distances

Usage

```
mahal.dist(data, train, labels = NULL)
```

Arguments

`data`	A matrix of numerical data points.
`train`	A gaussian model as returned by the `train` function.
`labels`	A vector of labels..

Details

The `train` function finds the centroids and covariance matrices for a set of data and corresponding labels: one per unique label. This function can be used to find the mahalanobis distance of every data point in a dataset to each of the class centroids. The columns of the resulting matrix are marked with the label of the centroid to which they refer. The function `mahal` should be used if you want to find the closest centroid to each data point.

Value

A matrix of distances with one column for every class (label) in the gaussian model.

See Also

train, mahal, bayes.lab, bayes.dist

make.emuRsegs *Make emuDB segment list*

Description

Make emuDB segment list

Usage

```
make.emuRsegs(dbName, seglist, query, type)
```

Arguments

dbName	name of emuDB
seglist	segment list data.frame
query	query string
type	type of list elements

make.seglist *Make an Emu segment list from the various components*

Description

This is the appropriate way to make an Emu segment list and ensure that it has all of the required components.

Usage

```
make.seglist(labels, start, end, utts, query, type, database)
```

Arguments

labels	A character vector of labels for each segment
start	A vector of start times
end	A vector of end times
utts	A character vector of utterance names
query	A query string
type	segment or event
database	The database name associated with the segment list

Details

An Emu segment list is the result of a query to a speech database (see query) and has one row per matching segment or event from the query. Each row lists the label, start and end times (in milliseconds) and utterance name for the segment. This information is used by get_trackdata and other functions to extract data corresponding to these segments.

In order to ensure the proper format for segment lists and to ensure against future changes to the format, make.seglist should be used whenever you wish to create a segment list. Another function, modify.seglist can be used to change some part of an existing segment list. The functions label.emusegs, start.emusegs, end.emusegs and utt.emusegs can be used to access the different columns of the segment list.

Value

An Emu segment list.

Author(s)

Steve Cassidy

See Also

modify.seglist, label.emusegs

Examples

```
l <- c("A", "B", "C")
s <- 1:3
e <- 2:4
u <- c("u1", "u1", "u1")
segs <- make.seglist(l, s, e, u, "Fake Query", "segment", "fake")
segs
## summary gives an overview of the data in the segment list
summary(segs)


# The following should be TRUE
label(segs) == l
dur(segs) == s
end(segs) == e
utt(segs) == u
emusegs.database(segs) == "fake"
emusegs.type(segs) == "segment"
emusegs.query(segs) == "Fake Query"

# segment durations should all be 1
dur(segs) == c(1,1,1)
```

makelab *Write out ESPS-style label files*

Description

Writes out separate ESPS-label files for each utterance to a specified directory.

Usage

```
makelab(vectimes, uttname, dir, extn = "xlab", labels = NULL)
```

Arguments

vectimes	a vector of times
uttname	a character vector of the same length as vectimes giving the utterance name associated with each element of vectimes
dir	a character specifying the directory
extn	a character specifying the extension of the resulting files. Defaults to xlab
labels	either a single character vector or a character vector the same length as vectimes. Defaults to "T"

Value

ESPS-style label files are written out to the directory of the user's choice. One ESPS-label file is created for each utterance containing all time values for that utterance.

Author(s)

Jonathan Harrington

Examples

```
#first two segments (for the whole example) of segmentlist vowlax
vowlax[1:2,]

#format track of vowlax
vowlax.fdat[1:2,]

#Formant values of the midpoint of the segment
vowlax.fdat.5 = dcut(vowlax.fdat,0.5,prop=TRUE)

#the time marks of the midpoint of the segment
times = vowlax.fdat.5[1:2,1]
times

#utterance names to the segments
utts = utt(vowlax[1:2,])
```

```
utts

#the path to save the label files to "." is the RHOME Directory
path = "."

#write the label files to path
## Not run: makelab(times, utts, path, labels="T")

#the first two segments are from the same utterance,
#thus one label file was created in the R_HOME directory
```

`matscan` *Read matrix data from a file*

Description

Reads matrix data from a file

Usage

```
matscan(file, num.cols = utils::count.fields(file)[1], what = 0, sk = 0)
```

Arguments

`file`	A filename.
`num.cols`	The number of columns of data in the file.
`what`	A template for the data elements in the file, it should be a number for numeric data (the default) or a string for string data. Note that an Splus matrix can only hold one type of data (string or numeric), for mixed types use data tables and the `read.table` function.
`sk`	The number of leading lines of the file to skip.

Details

This function has been partially superseded by the introduction of data frames and the read.table function. It is still useful however for reading data into Splus matrix objects.

Value

A matrix corresponding to the data in `file`.

See Also

read.table

`mel` *Convert Hz to the mel scale*

Description

The calculation is done using the formulae mel = 1/log(2) * (log(1 + (Hz/1000))) * 1000 where Hz is the frequency in Hz.

Usage

```
mel(a)
```

Arguments

`a`	A vector or matrix of data or a spectral object.

Details

If 'data' is a spectral object, then the frequencies are changed so that they are proportional to the mel scale and such that the mel intervals between frequencies are constant between the lowest and highest frequencies. More specifically, suppose that a spectral object has frequencies at 0, 1000, 2000, 3000, 4000 Hz. Then the corresponding frequencies extend in mel between 0 and 2321.928 mel (=4000 Hz in mels) in four equal intervals, and linear interpolation is used with the 'approx' function to obtain the dB values at those frequencies.

Value

A vector or matrix or spectral object of the same length and dimensions as data.

Author(s)

Jonathan Harrington

References

Traunmueller, H. (1990) "Analytical expressions for the tonotopic sensory scale" J. Acoust. Soc. Am. 88: 97-100.

See Also

`bark`, `plot.spectral`

Examples

```
#convert Hertz values to mel

vec <- c(500, 1500, 2500)
vec
mel(vec)


       # convert Hertz values to mel

       mel(vec)


       # convert the $data values in a trackdata object to mel
       # create a new track data object

       t1 <- dip.fdat
       t1[1]

       # convert Hertz to mel

       t1$data <- mel(t1$data)
       t1[1]

# warp the frequency axis of a spectral object such
# that it is proportional to the mel scale.

w = mel(e.dft)
oldpar = par(mfrow=c(1,2))
plot(w, type="l")


# The values of w are at equal mel intervals. Compare
# with

plot(e.dft, freq=mel(trackfreq(e.dft)))

# the latter has a greater concentration of values
# in a higher frequency range.

par(oldpar)
```

`modify.seglist` *Modify one of the components of an Emu segment list*

Description

This function can be used to modify one of the parts of an Emu segment list while leaving the other parts unchanged.

Usage

```
modify.seglist(
  segs,
  labels = label.emusegs(segs),
  start = start.emusegs(segs),
  end = end.emusegs(segs),
  utts = utt.emusegs(segs),
  query = emusegs.query(segs),
  type = emusegs.type(segs),
  database = emusegs.database(segs)
)
```

Arguments

segs	A segment list to modify, a modified copy is returned
labels	A new label vector
start	A new start time vector
end	A new end time vector
utts	A new vector of utterance labels
query	A new query string to associate with the segment list
type	A new type string
database	A new database name

Details

An Emu segment list has a number of components and is stored as an R object of class emusegs. This function can be used to modify a segment list while retaining all of the proper structures.

Any new vectors passed to the function must have the same length as the segment list itself for this call to succeed.

All arguments are optional and default to not modifying the segment list if not supplied.

The original segment list is not modified, instead, a modified copy is returned.

Value

An Emu segment list.

Author(s)

Steve Cassidy

See Also

query

Examples

```
data(vowlax)
segs = vowlax
# extend the start times by 10ms
newsegs <- modify.seglist( segs, start=start(segs)+10 )

# change the associated database name
# this will affect where emu.track looks to find data
newsegs <-  modify.seglist( segs, database="notdemo" )
```

---

moments *Function to calculate statistical moments*

Description

The function calculates the first 4 moments, i.e. the mean, variance, skew, kurtosis.

Usage

```
moments(count, x, minval = FALSE)
```

Arguments

count	A vector of the observed instances per class
x	A vector of the same length as count defining the class. If missing, and if count is of class spectral, then x is equal to trackfreq(count). If x is missing and is not of class spectral, then x default to 0:(length(count)-1)
minval	If TRUE, subtract min(count) from count so that the minimum value of count is zero. This is principally used in calculating spectral moments where count is in decibels, and more generally if count contains negative values.

Details

The units of the first moment are the same as x, the units of the second moment are x^2, and the third and fourth moments are dimensionless.

Author(s)

Jonathan Harrington

References

Snedecor, G & Cochran, W. 'Statistical Methods' Iowa State Press. Wuensch,K., 2005

Examples

```
# first four moments of a vector
mom <- moments(bridge[,2])
# the above is the same as moments(bridge[,2], 0:12)
# first four moments of a spectral vector with the dB values
# reset so that the minimum dB value is 0. The d.c. offset is also
# excluded in the calculation
mom <- moments(e.dft[-1], minval=TRUE)
# the temporal skew of F1 for the 10th segment. Use
m <- moments(vowlax.fdat[10,1]$data)[3]
```

---

mu.colour *Function for specifying color, linetype, and line-widths in EMU plotting functions.*

Description

The function specifies color, linetype and linewidths in EMU plotting functions as is used mostly in calls from within plot.trackdata, plot.spectral, eplot, and dplot

Usage

```
mu.colour(labs, col = TRUE, linetype = FALSE, lwd = NULL, pch = NULL)
```

Arguments

labs	A vector of character labels
col	A code passed to the 'col' argument in plotting functions. There are four possibilities. Either logical, a character vector, or a numeric vector. In the first case, if TRUE, then a different numeric code is given for each unique label type. For example, if labs is c("a", "b", "a", "c"), then the output is c(1, 2, 1, 3). If FALSE, then for this example, the output is c(1, 1, 1, 1). In the second case, the character vector can be either a single element specifying a character, or there can be as many elements as there are unique colors. Thus if col = "red", then for the example c("a", "b", "a", "c"), the output is c("red", "red", "red", "red"). Alternatively, since there are three unique labels for this example, then the user could specify col = c("green", "red", "blue") and the output is c("green", "red", "green", "blue") if labs is c("a", "b", "a", "c"). In the third case, 'col'. can be either a single element numeric vector, or its length must be equal to the number of unique types in labs. For example, if col=3 and if labs = c("a", "b", "a", "c"),

then the output is c(3, 3, 3, 3). Alternatively, if col = c(2,3,1), then the output is c(2, 3, 2, 1) for the same example. Finally, col can be specified as a character or numeric vector that is the same length as labs, allowing the user to choose the color in which each line should be drawn. The default is col = TRUE.

`linetype` A code specifying linetypes, i.e. the values passed to lty in plotting functions.There are 2 possibilities. Either logical, a character vector, or a numeric vector. In the first case, if TRUE, then a different numeric code is given for each unique label type. For example, if labs is c("a", "b", "a", "c"), then the output is c(1, 2, 1, 3). If FALSE, then for this example, the output is c(1, 1, 1, 1). In the second case, 'linetype' can be either a single element numeric vector, or its length must be equal to the number of unique types in labs. For example, if linetype=3 and if labs = c("a", "b", "a", "c"), then the output is c(3, 3, 3, 3). Alternatively, if linetype = c(2,3,1), then the output is c(2, 3, 2, 1) for the same example. Finally, linetype can be specified as a numeric vector that is the same length as labs, allowing the user to choose the linetype in which each line should be drawn. The default is linetype=FALSE

`lwd` A code passed to the lwd argument in plotting functions. 'lwd' can be either a single element numeric vector, or its length must be equal to the number of unique types in labs. For example, if lwd=3 and if labs = c("a", "b", "a", "c"), then the output is c(3, 3, 3, 3). Alternatively, if lwd = c(2,3,1), then the output is c(2, 3, 2, 1) for the same example. The default is NULL in which case all lines are drawn with lwd=1

`pch` A code passed to the pch argument in plotting functions. Functions in the same way as lwd above

Details

Parameters are also supplied for use with the function 'legend'

Value

If it is a LISTRUE, use

`colour` A code for the color'

`linetype` A code for the linetype

`lwd` A code for the line width

`legend` A list consisting of $legend$lab, $legend$lty and $legend$lwd that specify the parameters for the 'legend' function.

...

Author(s)

Steve Cassidy, modified by Jonathan Harrington

See Also

`plot.trackdata` `dplot` `eplot` `plot.spectral`

Examples

```
# examples will be given using the above functions
# b/w but with different linetypes
eplot(vowlax.fdat.5[,1:2], vowlax.l, col=FALSE, lty=TRUE)

# user-defined colors
eplot(vowlax.fdat.5[,1:2], vowlax.l, col=c("green", "blue", "red", "orange"))

# spectral plot, user-defined colors, the last one is dotted
# and with a line-thickness of 2
plot(vowlax.dft.5[1:20,], vowlax.l[1:20],
col=c("green", "blue", "red", "orange"),
fun=mean, lty=c(1, 1, 1, 2), lwd=c(1, 1, 1, 2))

# similar but using dplot()
dplot(vowlax.fdat[1:20,2], vowlax.l,
col=c("green", "blue", "red", "orange"),
lwd=c(1, 1, 1, 2), lty=c(1, 1, 1, 2))

# the default except plot everything with a dotted line and plotting symbol 4
dplot(vowlax.fdat[,2], vowlax.l, average=TRUE, lty=2, pch=4, type="b", xlim=c(40, 60))

# the default except plot everything with a dotted line and
# with double line thickness
eplot(vowlax.fdat.5[,1:2], vowlax.l, lty=2, lwd=2)
```

---

muclass *Find common elements in vectors*

Description

Finds common elements in vectors

Usage

```
muclass(labels, class)
```

Arguments

labels	A vector of labels.
class	A label or vector of labels.

Value

A logical vector which is TRUE for each element in `labels` which matches `class` or an element of `class`.

See Also

match

Examples

```
muclass(c("a", "b", "c"), c("a", "c"))
#[1] TRUE FALSE TRUE
```

`norm` *Normalise speech data*

Description

Normalises speech data

Usage

```
norm(data, speakerlabs, type = "gerst", rescale = FALSE)
```

Arguments

`data`	A matrix of data. Can be either an n-columned matrix or a trackdata object as returned by `track`.
`speakerlabs`	A parallel vector of speaker labels.
`type`	The type of extrinsic normalisation to be performed on data. type can be `"nearey"`, `"cen"`, `"lob"`, `"gerst"` (default), for normalisation according to Nearey, centroid method, Lobanov, or Gerstman.
`rescale`	Currently only works for Lobanov normalisation. The normalised values are multiplied by the standard deviation and then the mean is added, where the standard deviation and mean are across all original speakers' unnormalised data.

Details

Types of normalisation: `"nearey"`, Nearey : Find the log of each data element and subtract from each the mean of the logarithmic data. `"cen"`, centroid: Find the mean of the data column and subtract it from each data element in that column. `"lob"`, Lobanov: Find the mean and standard deviation of the data. Subtract the mean from each data element and divide each result by the standard deviation. "gerst", Gerstman: Subtract from the data the minimum formant value then divide by the formant range.

Value

Normalised values of data are returned, having the same structure as data.

See Also

track

`normalize_length` *Normalize length of segments contained in a* `data.frame` *like object returned by* `get_trackdata`

Description

Normalize length of segments contained in a `data.frame` like object returned by `get_trackdata`

Usage

```
normalize_length(x, colNames = NULL, N = 21)
```

Arguments

`x`	data.frame like object that was generated by `get_trackdata` with the resultType set to either `emuRtrackdata` or `tibble`
`colNames`	character vector containing names of columns to normalize. If not set all data columns are normalized (T1-TN as well as other numeric columns).
`N`	specify length of normalized segments (each segment in resulting object will consist of `N` rows).

Value

data.frame like object containing the length normalized segments

See Also

`emuRtrackdata` `emuRsegs`

`palate` *Obtain a three-dimensional palatographic array*

Description

Function to calculate a three-dimensional palatographic array from.

Usage

```
palate(epgdata)
```

Arguments

epgdata	An eight-columned EPG-compressed trackdata object or an eight columned matrix of EPG-compressed trackdata.

Details

An EPG compressed trackdata object that is output from the Reading system contains eight columns of data and each row value when converted to binary numbers (after adding 1) gives the corresponding EPG contact patterns. This function does the conversion to binary values.

Value

An array of three dimensions of 8 rows x 8 columns x n segments where n is the number of segments in the trackdata object or matrix. The rows and columns are given dimension names, the dimension names of the third dimension contains the times at which the palatograms occur.

Author(s)

Jonathan Harrington

See Also

epgcog epggs epgai epgplot

Examples

```
# convert an EPG-compressed trackdata object to palatograms
p <- palate(coutts.epg)

# convert an EPG-compressed matrix to palatograms
p <- palate(dcut(coutts.epg, 0, prop=TRUE))
```

---

perform *Performance (hit rate) of a confusion matrix*

Description

Performs (hit rate) of a confusion matrix

Usage

```
perform(data)
```

Arguments

data	A confusion matrix.

Value

Calculates the accuracy (total score) of the confusion matrix, returning percentage of correct, and incorrect matches.

See Also

confusion

`plafit` *Calculate the coefficients of a parabola*

Description

Fit a second ordered polynomial to a vector of values

Usage

```
plafit(wav, fit = FALSE, n = 101)
```

Arguments

`wav`	a vector or single column matrix of numeric values to which the 2nd order polynomial is to be fitted.
`fit`	if FALSE, return the coefficients of the polynomial; if TRUE, the values of the polynomial are returned to the same length as the vector wav.
`n`	in fitting the polynomial, linear time normalisation is first applied to the input vector wav to 101 points. The polynomial is fitted under the assumption that these points extend linearly in time between t = -1 and t = 1 with t = 0 occurring at the temporal midpoint.

Details

The function fits a parabola (2nd order polynomial) following the method of van Bergem, Speech Communication, 14, 1994, 143-162. The algorithm fixes the parabola at the onset, midpoint, and offset of the vector i.e. such htat the fitted parabola and original vector have the same values at these points.

Value

The function returns the coefficients of c0, c1, c2 in the parabola y = c0 + c1t + c2t^2 where t extends between -1 and 1. The function can also be used to derive the values of the parabola as a function of time from the coefficients.

Author(s)

Jonathan Harrington

See Also

dct

Examples

```
# fit a polynomial to a segment of fundamental frequency data
plafit(vowlax.fund[1,]$data)

# return the fitted values of the polynomial
plafit(vowlax.fund[1,]$data, fit=TRUE)
```

`plot.spectral` *Plot spectra from EMU spectral objects*

Description

The function plots spectrum of any EMU spectral object.

Usage

```
## S3 method for class 'spectral'
plot(
  x,
  labs,
  ylim,
  xlim,
  col,
  lty,
  lwd,
  fun,
  freq,
  type = "l",
  power = FALSE,
  powcoeffs = c(10, 10),
  dbnorm = FALSE,
  dbcoeffs = c(0, 0),
  legend = TRUE,
  axes = TRUE,
  ...
)
```

Arguments

x	An EMU object of class 'spectral'
labs	An optional vector character labels. Must be the same length as specdata
ylim	A two-element numeric vector for the y-axis range (see 'par')
xlim	A two-element numeric vector for the x-axis range (see 'par')
col	Specify a color - see 'mu.colour')
lty	Specify a linetype - see 'mu.colour'
lwd	Specify line thickness - see 'mu.colour'
fun	An R function name e.g., mean, var, sum, etc. The function is applied separately to each category type specified in labs
freq	A numeric vector the same length as the number of columns in specdata specifying the frequencies at which the spectral data is to be plotted. If not supplied, defaults to trackfreq(specdata)
type	A single element character vector for the linetype
power	Logical. If TRUE, then specdata (or specdata$data if specdata is a trackdata object, is converted to a * specdata^b, where a and b have the values given in powcoeffs. This operation is applied before b
powcoeffs	A two-element numeric vector. Defaults to c(10, 10)
dbnorm	Logical. If TRUE, apply dB-level normalization per spectrum as defined by dbcoeffs below. Defaults to FALSE.
dbcoeffs	A two element numeric vector (x, y). The spectra are normalised in such a way that the values of each spectrum at a frequency of y are set to a dB level of x. For example, to normalise the spectrum to 10 dB at 2000 Hz, set dbnorm to TRUE and dbcoeffs to c(2000, 10)
legend	Parameters for defining the legend. See 'mu.legend' for further details
axes	A logical vector indicating whether the axes should be plotted
...	Further graphical parameters may be supplied.

Details

This function is implemented when a spectral trackdata object is called with the 'plot' function.

Note

To plot spectral data from a spectral trackdata object, then call the function explicitly with 'plot/spectral' rather than with just 'plot'

Author(s)

Jonathan Harrington

See Also

plot plot.trackdata as.spectral

Examples

```
## Not run:

plot(vowlax.dft.5[1,])

# with label types
plot(vowlax.dft.5[1:20,], vowlax.l[1:20])

# As above but averaged after converting to power ratios.
plot(vowlax.dft.5[1:20,], vowlax.l[1:20], fun=mean, power=TRUE)

# All the spectra of one segment in a trackdata object
plot(fric.dft[1,])


## End(Not run)
```

`plot.trackdata` *Produces time-series plots from trackdata*

Description

The function produces a plot as a function of time for a single segment or multiple plots as a function of time for several segments.

Usage

```
## S3 method for class 'trackdata'
plot(
  x,
  timestart = NULL,
  xlim = NULL,
  ylim = NULL,
  labels = NULL,
  col = TRUE,
  lty = FALSE,
  type = "p",
  pch = NULL,
  contig = TRUE,
  ...
)
```

Arguments

x	A trackdata object.

timestart	A single valued numeric vector for setting the time at which the trackdata should start. Defaults to NULL which means that the start time is taken from start(trackdata), i.e. the time at which the trackdata object starts.
xlim	A numeric vector of two values for specifying the time interval over which the trackdata is to be plotted. Defaults to NULL which means that the trackdata object is plotted between between the start time of the first segment and the end time of the last segment.
ylim	Specify a yaxis range.
labels	A character vector the same length as the number of segments in the trackdata object. Each label is plotted at side = 3 on the plotted at the temporal midpoint of each segment in the trackdata object. Defaults to NULL (plot no labels). Labels will only be plotted if xlim=NULL.
col	A single element logical vector. Defaults to TRUE to plot each label type in a different colour
lty	A single element logical vector. Defaults to FALSE. If TRUE, plot each label type in a different linetype
type	Specify the type of plot. See plot for the various possibilities
pch	The symbol types to be used for plotting. Should be specified as a numeric vector of the same length as there are unique label classes
contig	A single valued logical vector TRUE or FALSE. If TRUE, then all the segments of the trackdata object are assumed to be temporally contiguous, i.e. the boundaries of the segments are abutting in time and the start time of segment[j,] is the end time of segment[j-1,]. In this case, all the segments of the trackdata object are plotted on the same plot as a function of time. An example of a contiguous trackdata object is coutts.sam. contig = FALSE is when a trackdata object is non-contiguous e.g. all "i:" vowels in a database. An example of a non-contiguous trackdata object is vowlax.fdat. If contig=FALSE then each segment of the trackdata object is plotted separately.
...	the same graphical parameters can be supplied to this function as for plot e.g type="l", lty=2 etc.

Details

The function plots a single segment of trackdata as a function of time. If the segment contains multiple tracks, then these will be overlaid. If there are several temporally non-contiguous segments in the trackdata object, each segment is plotted in a different panel by specifying contig=FALSE. This function is not suitable for overlaying trackdata from more than one segments on the same plot as a function of time: for this use dplot().

Author(s)

Jonathan Harrington

See Also

plot, dplot

Examples

```
# a single segment of trackdata (F1) plotted as a function of time.
plot(vowlax.fdat[1,1])

# as above, but limits are set for the time axis.
plot(vowlax.fdat[1,1], xlim=c(880, 920))

# the the start-time of the x-axis is set to 0 ms, plot F1 and F3, lineplot
plot(vowlax.fdat[1,c(1,3)],  timestart=0, type="l")


# plot F1-F4, same colour, same plotting symbol, between 900
# and 920 ms, type is line and points plot, different linetype per track, no box
plot(vowlax.fdat[1,], col="blue", pch=20, xlim=c(900, 920), type="b", lty=TRUE, bty="n")


# F1 and F2 of six vowels with labels, separate windows
oldpar = par(mfrow=c(2,3))
plot(vowlax.fdat[1:6,1:2], contig=FALSE, labels=vowlax.l[1:6], ylab="F1 and F2",
xlab="Time (ms)", type="b", ylim=c(300, 2400))

# As above, timestart set to zero, colour set to blue, different plotting
# symbols for the two tracks
plot(vowlax.fdat[1:6,1:2], contig=FALSE, labels=vowlax.l[1:6], ylab="F1 and F2",
xlab="Time (ms)", type="b", col="blue", pch=c(1,2),  ylim=c(300, 2400), timestart=0)

# RMS energy for the utterance 'just relax said Coutts'
 plot(coutts.rms, type="l")
# as above a different colour
 plot(coutts.rms, type="l", col="pink")
# as above, linetype 2, double line thickness, no box, times reset to 0 ms
 plot(coutts.rms, type="l", col="pink", lty=2, lwd=2, bty="n", timestart=0)
# as above but plotted as non-contiguous segments, i.e one segment per panel
 par(mfrow=c(2,3))
 plot(coutts.rms, type="l", col="pink", lty=2, lwd=2, bty="n", timestart=0, contig=FALSE)
# plot with labels
 labels = label(coutts)
par(mfrow=c(1,1))
 plot(coutts.rms, labels=labels, type="l", bty="n")
# as above, double line-thickness, green, line type 3, no box,
# time start 0 ms with x and y axis labels
 plot(coutts.rms, labels=labels, type="l", lwd=2,
      col="green", lty=3,  bty="n", timestart=0, xlab="Time (ms)", ylab="Amplitude")
# as above with a different plotting symbol for the points
par(mfrow=c(2,3))
 plot(coutts.rms, labels=labels, type="b", lwd=2, col="green",
      timestart=0, bty="n", contig=FALSE, pch=20)

par(oldpar)
```

polhom — *Segment list of four Polish homorganic fricatives from database epg-polish.*

Description

An EMU dataset

Format

segmentlist

polhom.epg — *EPG-compressed trackdata from the segment list polhom*

Description

An EMU dataset

Format

trackdata object

polhom.l — *Vector of phonetic labels from the segment list polhom*

Description

An EMU dataset

Format

vector of phonetic labels

print.emuRsegs *Print emuRsegs segment list*

Description

Print emuRsegs segment list

Usage

```
## S3 method for class 'emuRsegs'
print(x, ...)
```

Arguments

x	object to print
...	additional params

print.emuRtrackdata *Print emuRtrackdata object*

Description

Print emuRtrackdata object

Usage

```
## S3 method for class 'emuRtrackdata'
print(x, ...)
```

Arguments

x	object to print
...	additional params

query *Query emuDB*

Description

Function to query annotation items/structures in an emuDB

Usage

```
query(
  emuDBhandle,
  query,
  sessionPattern = ".*",
  bundlePattern = ".*",
  queryLang = "EQL2",
  timeRefSegmentLevel = NULL,
  resultType = "tibble",
  calcTimes = TRUE,
  verbose = FALSE
)
```

Arguments

emuDBhandle	emuDB handle object (see load_emuDB)
query	string (see vignette https://ips-lmu.github.io/The-EMU-SDMS-Manual/chap-querysys.html)
sessionPattern	A regular expression pattern matching session names to be searched from the database
bundlePattern	A regular expression pattern matching bundle names to be searched from the database
queryLang	query language used for evaluating the query string
timeRefSegmentLevel	set time segment level from which to derive time information. It is only necessary to set this parameter if more than one child level contains time information and the queried parent level is of type ITEM.
resultType	type (class name) of result (either 'tibble', 'emuRsegs' or 'emusegs' (use 'emusegs' for legacy compatablility only))
calcTimes	calculate times for resulting segments (results in NA values for start and end times in emuseg/emuRsegs). As it can be very computationally expensive to calculate the times for large nested hierarchies, it can be turned off via this parameter.
verbose	be verbose. Set this to TRUE if you wish to choose which path to traverse on intersecting hierarchies. If set to FALSE (the default) all paths will be traversed (= legacy EMU behavior).

Details

Evaluates a query string of query language queryLang on an emuDB referenced by `emuDBhandle` and returns a segment list of the desired type resultType. For details of the query language please refer to the EMU-SDMS manual's query system chapter (https://ips-lmu.github.io/The-EMU-SDMS-Manual/chap-querysys.html). This function extracts a list of segments which meet the conditions given by the query string. A segment can consist of one (e.g. 's') or more (e.g. 's->t') items from the specified emuDB level. Segment objects (type 'SEGMENT') contain the label string and the start and end time information of the segment (in ms). The `tibble` return type (now the defaults) objects additionally contain sample position of start and end item. Time information of symbolic elements (type 'ITEM') are derived from linked SEGMENT levels if available. If multiple linked SEGMENT levels exist, you can specify the level with the `timeRefSegmentLevel` argument. If time and sample values cannot be derived they will be set to NA. tibbles will be ordered by the columns UUID, session, bundle and sequence index (seq_idx). Legacy emusegs lists are ordered by the columns utts and start. The query may be limited to session and/or bundle names specified by regular expression pattern strings (see regex) in parameters `sessionPattern` respectively `bundlePattern`.

Value

result set object of class resultType (default: tibble, compatible to legacy types emuRsegs and emusegs)

See Also

load_emuDB

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

## Query database ae with EQL query "[Phonetic=t -> Phonetic=s]":
## 'Find all sequences /ts/ on the level named Phonetics'.
## and store result seglist in variable segListTs

seglistTs = query(ae, "[Phonetic == t -> Phonetic == s]")

## Query database ae with EQL query "[Syllable == S ^ Phoneme == t]":
## 'Find all items 't' on the level named Phoneme that are dominated by
## items 'S' in level Syllable.'
## Return legacy Emu result type 'emusegs'

query(ae, "[Syllable == S ^ Phoneme == t]", resultType = "emusegs")

## Query 't' items on the level named Phoneme from bundles whose
## bundle names start with 'msajc07'
## and whose session names start with '00'
## (Note that here the query uses the operator '=' (meaning '==')
## which is kept for backwards compatibilty to EQL1.)
```

```
query(ae, "Phoneme = p", bundlePattern = "msajc05.*", sessionPattern = "00.*")

## End(Not run)
```

---

rad *Function to convert between Hertz and Radians*

Description

convert between Hertz and Radians

Usage

```
rad(vec, samfreq = 20000, hz = TRUE)
```

Arguments

vec	A numerical vector of frequencies in Hz or radians
samfreq	A single element numerical vector of the sampling frequency. Defaults to 20000 Hz
hz	Logical. If TRUE, convert from Hz to radians otherwise from radians to hz

Author(s)

Jonathan Harrington

See Also

help

Examples

```
# 4000 Hz in radians at a sampling frequency of 8000 Hz
rad(4000, 8000)
# pi/2 and pi/4  radians in Hz at a sampling frequency of 10000 Hz
rad(c(pi/2, pi/4), 10000, FALSE)
```

radians *Converts degrees to radians*

Description

Converts degrees to radians

Usage

```
radians(degrees)
```

Arguments

degrees	Angular measurement for conversion.

Details

There are 360 degrees or 2 * PI radians in one full rotation.

Value

Angular measurement in radians.

randomise.segs *Randomise or Reverse items in a segment list*

Description

Randomises or Reverses items in a segment list

Usage

```
randomise.segs(segs, rand = TRUE, bwd = FALSE)
```

Arguments

segs	An Emu segment list.
rand	If TRUE, randomise the order of the segment lists (default).
bwd	If TRUE, reverse the order of the segment list.

Value

A segment list containing the original elements in random or reversed order. This is useful if the segment list is to be used as the source for a set of stimuli in a perception experiment.

See Also

query

Examples

```
data(vowlax)
## assumes a database called demo is available on your system and that
## the Emu system is installed.

# all Phonetic vowels in the database
segs <- vowlax

# randomise the segment list
rsegs <- randomise.segs( segs )
```

---

rbind.trackdata *A method of the generic function rbind for objects of class trackdata*

Description

Different track data objects from one segment list are bound by combining the $data columns of the track data object by rows. Track data objects are created by get_trackdata.

Usage

```
## S3 method for class 'trackdata'
rbind(...)
```

Arguments

...	track data objects

Details

All track data objects have to be track data of the same segment list. Thus $index and $ftime values have to be identically for all track data objects. The number of columns of the track data objects must match. Thus a track data object of more than one formant and single columned F0 track data object can not be rbind()ed.

Value

A track data object with the same $index and $ftime values of the source track data objects and with $data that includes all columns of $data of the source track data objects.

Author(s)

Jonathan Harrington

See Also

rbind cbind.trackdata trackdata get_trackdata

Examples

```
data(vowlax)

#segment list vowlax - first segment only
vowlax[1,]

#F0 track data object for vowlax - first segment only
vowlax.fund[1]

#rms track data object for vowlax - first segment only
vowlax.rms[1]

#now combine both track data objects
fund.rms.lax = rbind(vowlax.fund[1:10,], vowlax.rms[1:10,])

#the combined track data object
#The first ten rows in $data keep vowlax.fund data, the 11th to last row keeps vowlax.rms data
fund.rms.lax
```

---

read.emusegs *Create an Emu segment list from a file*

Description

Create an Emu segment list from a file saved by the Emu query tools.

Usage

```
read.emusegs(file)
```

Arguments

file	The name of the file to read

Details

Reads segment lists created by programs external to R/Splus and stored in text files on disk.

Value

An Emu segment list.

Author(s)

Steve Cassidy

See Also

query

Examples

```
## create a segment list file and write it out
# seglist.txt <- "database:demo"\
# query:Phonetic=vowel\
# type:segment\
#\
# @: 3059.65 3343.65 msdjc001\
# e: 5958.55 6244.55 msdjc002\
# @u 8984.75 9288.75 msdjc003\
# A 11880.8 12184.8 msdjc004\
# E 17188.3 17366.4 msdjc005\
# ei 20315.2 20655.2 msdjc006"

## Not run: cat(seglist.txt, file="seglist.txt")

# now read it back as a segment list
## Not run: segs <- read.emusegs("seglist.txt")
## Not run: segs
## and clean up
## Not run: unlink("seglist.txt")
```

---

read_bundleList *read bundleList*

Description

read bundleList JSON file in emuDB

Usage

```
read_bundleList(emuDBhandle, name)
```

Arguments

emuDBhandle	emuDB handle object (see load_emuDB)
name	name of bundleList (excluding the _bundleList.json suffix)

Details

Read bundleList JSON file in emuDB that is stored in the databases root dir sub-dir bundleLists/

Value

tibble with the columns session, name, comment, finishedEditing

---

rename_bundles *Rename bundles in emuDB*

Description

Rename bundles of emuDB.

Usage

```
rename_bundles(emuDBhandle, bundles)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
bundles	data.frame like object with the columns • session: name of sessions containing bundle • name: name of bundle • name_new: new name given to bundle It is worth noting that session and name are the columns returned by list_bundles.

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# list bundles of session "0000" of ae emuDB
bundles = list_bundles(emuDBhandle = ae,
                       session = "0000")

# append "XXX" to bundle names and rename
bundles$name_new = paste0(bundles$name, "XXX")
rename_bundles(emuDBhandle, bundles)
```

```
## End(Not run)
```

`rename_emuDB` *Rename emuDB*

Description

Rename a emuDB. This effectively renames the folder of a emuDB the _DBconfig.json file as well as the "name" entry in the _DBconfig.json file and the _emuDBcache.sqlite file if available.

Usage

```
rename_emuDB(databaseDir, newName)
```

Arguments

`databaseDir`	directory of the emuDB
`newName`	new name of emuDB

Examples

```
## Not run:

###################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# rename ae emuDB to "aeNew"
rename_emuDB(databaseDir = "/path/2/ae_emuDB", newName = "aeNew")


## End(Not run)
```

`replace_itemLabels` *Replace item labels*

Description

Replace the labels of all annotation items, or more specifically of attribute definitions belonging to annotation items, in an emuDB that match the provided `origLabels` character vector which the corresponding labels provided by the `newLabels` character vector. The indices of the label vectors provided are used to match the labels (i.e. `origLabels[i]` will be replaced by `newLabels[i]`).

Usage

```
replace_itemLabels(
  emuDBhandle,
  attributeDefinitionName,
  origLabels,
  newLabels,
  verbose = TRUE
)
```

Arguments

emuDBhandle	emuDB handle object (see load_emuDB)
attributeDefinitionName	name of a attributeDefinition of a emuDB where the labels are to be replaced
origLabels	character vector containing labels that are to be replaced
newLabels	character vector containing labels that are to replaced the labels of origLabels. This vector has to be of equal length to the origLabels vector.
verbose	Show progress bars and further information

See Also

load_emuDB

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# replace all "I" and "p" labels with "I_replaced" and "p_replaced"
replace_itemLabels(ae, attributeDefinitionName = "Phonetic",
                       origLabels = c("I", "p"),
                       newLabels = c("I_replaced", "p_replaced"))


## End(Not run)
```

---

requery_hier *Requery hierarchical context of a segment list in an emuDB*

Description

Function to requery the hierarchical context of a segment list queried from an emuDB

Usage

```
requery_hier(
  emuDBhandle,
  seglist,
  level,
  collapse = TRUE,
  resultType = "tibble",
  calcTimes = TRUE,
  timeRefSegmentLevel = NULL,
  verbose = FALSE
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
seglist	segment list to requery on (type: emuRsegs)
level	character string: name of target level
collapse	collapse the found items in the requested level to a sequence (concatenated with ->). If set to FALSE separate items as new entries in the emuRsegs object are returned.
resultType	type of result (either 'tibble' == default or 'emuRsegs')
calcTimes	calculate times for resulting segments (results in NA values for start and end times in emuseg/emuRsegs). As it can be very computationally expensive to calculate the times for large nested hierarchies, it can be turned off via this boolean parameter.
timeRefSegmentLevel	set time segment level from which to derive time information. It is only necessary to set this parameter if more than one child level contains time information and the queried parent level is of type ITEM.
verbose	be verbose. Set this to TRUE if you wish to choose which path to traverse on intersecting hierarchies. If set to FALSE (the default) all paths will be traversed (= legacy EMU behaviour).

Details

A segment is defined as a single item or a chain of items from the respective level, e.g. if a level in a bundle instance has labels 'a', 'b' and 'c' in that order, 'a' or 'a->b' or 'a->b->c' are all valid segments, 'a->c' is not. For each segment of the input segment list seglist the function checks the start and end item for hierarchically linked items in the given target level, and based on them constructs segments in the target level. As the start item in the resulting segment the item with the lowest sequence index is chosen; for the end item that with the highest sequence index. If the parameter collapse is set to TRUE (the default), it is guaranteed that result and input segment list have the same length (for each input segment one or multiple segments on the target level was found). If multiple linked segments where found they are collapsed into a sequence of segments ('a->b->c') and if no linked items where found an NA row is inserted.

Value

result set object of class emuRsegs or tibble

See Also

query requery_seq emuRsegs

Examples

```
## Not run:

#################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

## Downward requery: find 'Phoneme' sequences of all words 'beautiful' (of level 'Text')
## Note that the resulting segments consists of phoneme sequences and have therefore
## the same length as the word segments.

sl1 = query(ae, "Text == beautiful")
requery_hier(ae, sl1, level = "Phoneme")

## Upward requery: find all word segments that dominate a 'p' on level 'Phoneme'
## Note that the resulting segments are larger than the input segments,
## because they contain the complete words.

sl1 = query(ae, "Phonetic == p")
requery_hier(ae, sl1, level = 'Text')

## Why is there a 'p' the word 'emphazised'? Requery the whole words back down to 'Phoneme' level:

requery_hier(ae, sl1, level = 'Phoneme')

## ... because of 'stop epenthesis' a 'p' is inserted between 'm' and 'f'

## Combined requery: last phonemes of all words beginning with 'an'.
## Note that we use a regular expression 'an.*' (EQL operator '=~') in the query.

sl1=query(ae, "Text =~ an.*")
requery_seq(ae, requery_hier(ae, sl1, level = 'Phoneme'), offsetRef = 'END')


## End(Not run)
```

---

requery_seq *Requery sequential context of segment list in an emuDB*

Description

Function to requery sequential context of a segment list queried from an emuDB

Usage

```
requery_seq(
  emuDBhandle,
  seglist,
  offset = 0,
  offsetRef = "START",
  length = 1,
  ignoreOutOfBounds = FALSE,
  resultType = "tibble",
  calcTimes = TRUE,
  timeRefSegmentLevel = NULL,
  verbose = FALSE
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
seglist	segment list to requery on (type: 'tibble' or 'emuRsegs')
offset	start item offset in sequence (default is 0, meaning the start or end item of the input segment)
offsetRef	reference item for offset: 'START' for first and 'END' for last item of segment
length	item length of segments in the returned segment list
ignoreOutOfBounds	ignore result segments that are out of bundle bounds
resultType	type of result (either 'tibble' == default, 'emuRsegs')
calcTimes	calculate times for resulting segments (results in NA values for start and end times in emuseg/emuRsegs). As it can be very computationally expensive to calculate the times for large nested hierarchies, it can be turned off via this boolean parameter.
timeRefSegmentLevel	set time segment level from which to derive time information. It is only necessary to set this parameter if more than one child level contains time information and the queried parent level is of type ITEM.
verbose	be verbose. Set this to TRUE if you wish to choose which path to traverse on intersecting hierarchies. If set to FALSE (the default) all paths will be traversed (= legacy EMU behaviour).

Details

Builds a new segment list on the same hierarchical level and the same length as the segment list given in seglist. The resulting segments usually have different start position and length (in terms of items of the respective level) controlled by the offset, offsetRef and length parameters. A segment here is defined as a single item or a chain of items from the respective level, e.g. if a level in a bundle instance has labels 'a', 'b' and 'c' in that order, 'a' or 'a->b' oder 'a->b->c' are all valid segments, but not 'a->c'. offsetRef determines if the position offset is referenced

to the start or the end item of the segments in the input list `seglist`; parameter `offset` determines the offset of the resulting item start position to this reference item; parameter `length` sets the item length of the result segments. If the requested segments are out of bundle item boundaries and parameter `ignoreOutOfBounds` is `FALSE` (the default), an error is generated. To get residual resulting segments that lie within the bounds the `ignoreOutOfBounds` parameter can be set to `TRUE`. The returned segment list is usually of the same length and order as the input `seglist`; if `ignoreOutOfBounds=FALSE`, the resulting segment list may be out of sync.

Value

result set object of class emuRsegs or tibble

See Also

`query` `requery_hier` `emuRsegs`

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

## Requery previous item of 'p' on level 'Phonetic'
sl1 = query(ae, "Phonetic == p")

requery_seq(ae, sl1, offset = -1)

## Requery context (adding previuos and following elements)
## of 'p' on phonetic level

requery_seq(ae, sl1, offset = -1, length = 3)

## Requery previous item of n->t sequence
sl2 = query(ae, "[Phoneme == n -> Phoneme == t]")

requery_seq(ae, sl2, offset = -1)

## Requery last item within n->t sequence

requery_seq(ae, sl2, offsetRef = 'END')

## Requery following item after n->t sequence

requery_seq(ae, sl2, offset = 1, offsetRef = 'END')

## Requery context (previous and following items) of n->t sequence

requery_seq(ae, sl2, offset = -1, length = 4)

## Requery next word contexts (sequence includes target word)
```

```
sl3 = query(ae, "Text == to")
requery_seq(ae, sl3, length = 2)

## Requery following two word contexts, ignoring segment
## sequences that are out of bundle end bounds
requery_seq(ae, sl3, length = 3, ignoreOutOfBounds = TRUE)


## End(Not run)
```

`resample_annots` *Resample annotations (*`_annot.json`*) files of emuDB*

Description

Resample all annotations (`_annot.json`) files of emuDB to a specified sample rate. It is up to the user to ensure that the samplerates of the annot.json files match those of the `.wav` files.

Usage

```
resample_annots(emuDBhandle, newSampleRate, verbose = TRUE)
```

Arguments

`emuDBhandle`	emuDB handle object (see load_emuDB)
`newSampleRate`	target sample rate
`verbose`	show progress bars and further information

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# resample
resample_annots(ae, newSampleRate = 16000)


## End(Not run)
```

`runBASwebservice_all` *Runs several BAS webservices, starting from an orthographic transcription*

Description

This function calls the BAS webservices G2P, MAUS, Pho2Syl, MINNI and (if necessary) Chunker. Starting from an orthographic transcription, it derives a tokenized orthographical word tier using the G2P tool. It also derives canonical pronunciations (in SAMPA) for the words. If at least one audio file is longer than 60 seconds, the function then calls the Chunker webservice to presegment the recordings. Subsequently, the webservice MAUS is called to derive a phonetic segmentation. A second, rough segmentation is created by running the phoneme decoder MINNI. Finally, syllabification is performed by calling Pho2Syl. **This function requires an internet connection.**

Usage

```
runBASwebservice_all(
  handle,
  transcriptionAttributeDefinitionName,
  language,
  orthoAttributeDefinitionName = "ORT",
  canoAttributeDefinitionName = "KAN",
  mausAttributeDefinitionName = "MAU",
  minniAttributeDefinitionName = "MINNI",
  sylAttributeDefinitionName = "MAS",
  canoSylAttributeDefinitionName = "KAS",
  chunkAttributeDefinitionName = "TRN",
  runMINNI = TRUE,
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

`handle`	emuDB handle
`transcriptionAttributeDefinitionName`	name of the attribute (not level!) containing an orthographic transcription.
`language`	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
`orthoAttributeDefinitionName`	attribute name for orthographic words

canoAttributeDefinitionName
: attribute name for canonical pronunciations of words

mausAttributeDefinitionName
: attribute name for the MAUS segmentation

minniAttributeDefinitionName
: attribute name for the MINNI segmentation

sylAttributeDefinitionName
: attribute name for syllable segmentation

canoSylAttributeDefinitionName
: attribute name for syllabified canonical pronunciations of words

chunkAttributeDefinitionName
: attribute name for the chunk segmentation. Please note that the chunk segmentation will only be generated if your emuDB contains audio files beyond the one minute mark.

runMINNI
: if set to TRUE (the default) the MINNI service is also run. As the MINNI service contains less languages than the others it can be useful to turn this off.

patience
: If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.

resume
: If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.

verbose
: Display progress bars and other information

Details

All necessary level, attribute and link definitions are created in the process. Note that this function will run all BAS webservices with default parameters, with four exceptions:

- Chunker: force=rescue
- G2P: embed=maus
- Pho2Syl: wsync=yes
- MAUS: USETRN=[true if Chunker was called or transcription is a segment tier, false otherwise]

If you wish to change parameters, you must use the individual runBASwebservices functions. This will also allow you to carry out manual corrections in between the steps, or to use different languages for different webservices.

See Also

Other BAS webservice functions: runBASwebservice_chunker(), runBASwebservice_g2pForPronunciation(), runBASwebservice_g2pForTokenization(), runBASwebservice_maus(), runBASwebservice_minni(), runBASwebservice_pho2sylCanonical(), runBASwebservice_pho2sylSegmental()

runBASwebservice_chunker *Creates a chunk segmentation using the webservice Chunker.*

Description

When audio input files are longer than approximately 10 minutes, alignment-based segmentation tools such as MAUS will take a long time to run. In these cases, the Chunker pre-segments the input into more digestible "chunks". As input, it requires a word tier with canonical pronunciation attributes (which can be derived by runBASwebservice_g2pForPronunciation). The resulting chunk level can be passed as input to runBASwebservice_maus. **This function requires an internet connection.**

Usage

```
runBASwebservice_chunker(
  handle,
  canoAttributeDefinitionName,
  language,
  chunkAttributeDefinitionName = "TRN",
  rootLevel = NULL,
  orthoAttributeDefinitionName = NULL,
  params = list(force = "rescue"),
  perspective = "default",
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
canoAttributeDefinitionName	name of the attribute (not level!) containing a canonical pronunciation of the words.
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
chunkAttributeDefinitionName	attribute name for the chunk segmentation
rootLevel	if provided, the new level will be linked to the root level

`orthoAttributeDefinitionName`
: if provided, chunk attributes will contain orthographic instead of SAMPA strings. Must be paired with the canonical pronunciation attributes in canoAttributeDefinitionName.

`params`
: named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.

`perspective`
: the webApp perspective that the new level will be added to. If NULL, the new level is not added to any perspectives.

`patience`
: If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.

`resume`
: If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.

`verbose`
: Display progress bars and other information

Details

Please note that the chunker output is **not** a semantically meaningful sentence or turn segmentation, meaning that it cannot be used for analyses of sentence durations and the like. By default, the chunker is called in force rescue mode. This means that the chunker is first run in its normal mode, and switches to forced chunking mode only when it fails to find chunks that are short enough for processing by MAUS. To disable the force mode completely, call this function with params=list(force="false"). To skip the normal chunking mode and go directly into forced chunking mode, use params=list(force="true").

See Also

Other BAS webservice functions: `runBASwebservice_all()`, `runBASwebservice_g2pForPronunciation()`, `runBASwebservice_g2pForTokenization()`, `runBASwebservice_maus()`, `runBASwebservice_minni()`, `runBASwebservice_pho2sylCanonical()`, `runBASwebservice_pho2sylSegmental()`

---

runBASwebservice_g2pForPronunciation

Creates canonical pronunciation attributes for a tier of tokenized orthographical words.

Description

This function calls the G2P webservice to add canonical pronunciation attributes in SAMPA (default) or IPA to a tier of tokenized orthographical words. It is usually called after tokenization with runBASwebservice_g2pForTokenization. Its output can be used as input to runBASwebservice_maus or runBASwebservice_chunker. **This function requires an internet connection.**

Usage

```
runBASwebservice_g2pForPronunciation(
  handle,
  orthoAttributeDefinitionName,
  language,
  canoAttributeDefinitionName = "KAN",
  params = list(embed = "maus"),
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
orthoAttributeDefinitionName	name of a attribute (not level!) containing orthographic words.
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
canoAttributeDefinitionName	attribute name for canonical pronunciations of words
params	named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.
patience	If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.
resume	If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.
verbose	Display progress bars and other information

Details

By default, G2P is called in MAUS embed mode. This is important if you intend to use MAUS afterwards. To disable MAUS embed mode, call this function with params=list(embed="no"). To derive IPA symbols, add outsym="ipa" to the parameter list.

See Also

Other BAS webservice functions: runBASwebservice_all(), runBASwebservice_chunker(), runBASwebservice_g2pForTokenization(), runBASwebservice_maus(), runBASwebservice_minni(), runBASwebservice_pho2sylCanonical(), runBASwebservice_pho2sylSegmental()

---

runBASwebservice_g2pForTokenization

Tokenizes an orthographic transcription.

Description

This function calls the webservice G2P to break up a transcription into tokens, or words. In addition to tokenization, G2P performs normalization of numbers and other special words. A call to this function is usually followed by a call to runBASwebservice_g2pForPronunciation. **This function requires an internet connection.**

Usage

```
runBASwebservice_g2pForTokenization(
  handle,
  transcriptionAttributeDefinitionName,
  language,
  orthoAttributeDefinitionName = "ORT",
  params = list(),
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
transcriptionAttributeDefinitionName	name of the attribute (not level!) containing an orthographic transcription.
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
orthoAttributeDefinitionName	attribute name for orthographic words

params	named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.
patience	If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.
resume	If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.
verbose	Display progress bars and other information

Details

All necessary level, link and attribute definitions are created in the process.

See Also

Other BAS webservice functions: `runBASwebservice_all()`, `runBASwebservice_chunker()`, `runBASwebservice_g2pForPronunciation()`, `runBASwebservice_maus()`, `runBASwebservice_minni()`, `runBASwebservice_pho2sylCanonical()`, `runBASwebservice_pho2sylSegmental()`

`runBASwebservice_maus` *Runs MAUS webservice to create a phonetic segmentation*

Description

This function calls the BAS webservice MAUS to generate a phonemic segmentation. It requires a word-tokenized tier with a SAMPA pronunciation, which can be generated by the function runBASwebservice_g2pForPronunciation. **This function requires an internet connection.**

Usage

```
runBASwebservice_maus(
  handle,
  canoAttributeDefinitionName,
  language,
  mausAttributeDefinitionName = "MAU",
  chunkLevel = NULL,
  turnChunkLevelIntoItemLevel = TRUE,
  params = NULL,
  perspective = "default",
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
canoAttributeDefinitionName	name of the attribute (not level!) containing the SAMPA word pronunciations. If this attribute resides on a segment level, the segment time information is used as a presegmentation. If it is an item level, no assumption is made about the temporal position of segments.
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
mausAttributeDefinitionName	attribute name for the MAUS segmentation
chunkLevel	if you have a chunk segmentation level, you can provide it to improve the speed and accuracy of MAUS. The chunk segmentation level must be a segment level, and it must link to the level of canoAttributeDefinitionName.
turnChunkLevelIntoItemLevel	if TRUE, and if a chunk level is provided, the chunk level is converted into an ITEM level after segmentation
params	named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.
perspective	the webApp perspective that the new level will be added to. If NULL, the new level is not added to any perspectives.
patience	If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.
resume	If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.
verbose	Display progress bars and other information

Details

All necessary level, link and attribute definitions are created in the process.

See Also

Other BAS webservice functions: runBASwebservice_all(), runBASwebservice_chunker(), runBASwebservice_g2pForPronunciation(), runBASwebservice_g2pForTokenization(), runBASwebservice_minni
runBASwebservice_pho2sylCanonical(), runBASwebservice_pho2sylSegmental()

runBASwebservice_minni

Creates a rough phonetic segmentation by running the phoneme decoder webservice MINNI.

Description

The MINNI phoneme decoder performs phoneme-based decoding on the signal without input from the transcription. Therefore, labelling quality is usually worse than that obtained from MAUS (runBASwebservice_maus). Contrary to MAUS however, there is no need for a pre-existing transcription.

Usage

```
runBASwebservice_minni(
  handle,
  language,
  minniAttributeDefinitionName = "MINNI",
  rootLevel = NULL,
  params = list(),
  perspective = "default",
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
minniAttributeDefinitionName	attribute name for the MINNI segmentation
rootLevel	if provided, the new level will be linked to the root level
params	named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.

perspective	the webApp perspective that the new level will be added to. If NULL, the new level is not added to any perspectives.
patience	If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.
resume	If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.
verbose	Display progress bars and other information

Details

All necessary level, link and attribute definitions are created in the process.

See Also

Other BAS webservice functions: runBASwebservice_all(), runBASwebservice_chunker(), runBASwebservice_g2pForPronunciation(), runBASwebservice_g2pForTokenization(), runBASwebservice_maus(), runBASwebservice_pho2sylCanonical(), runBASwebservice_pho2sylSegmental()

---

runBASwebservice_pho2sylCanonical

Adds syllabified word labels to a word level that already contains canonical pronunciations.

Description

This function calls the webservice Pho2Syl to add syllabified canonical pronunciation labels to a word level that already contains unsyllabified canonical pronunciation labels (as can be derived using runBASwebservice_g2pForPronunciation). **This function requires an internet connection.**

Usage

```
runBASwebservice_pho2sylCanonical(
  handle,
  canoAttributeDefinitionName,
  language,
  canoSylAttributeDefinitionName = "KAS",
  params = list(),
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
canoAttributeDefinitionName	name of the attribute (not level!) containing a canonical pronunciation of the words.
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
canoSylAttributeDefinitionName	attribute name for syllabified canonical pronunciations of words
params	named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.
patience	If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.
resume	If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.
verbose	Display progress bars and other information

See Also

Other BAS webservice functions: runBASwebservice_all(), runBASwebservice_chunker(), runBASwebservice_g2pForPronunciation(), runBASwebservice_g2pForTokenization(), runBASwebservice_maus(), runBASwebservice_minni(), runBASwebservice_pho2sylSegmental()

---

runBASwebservice_pho2sylSegmental

Creates a syllable segmentation on the basis of a phonetic segmentation.

Description

This function calls the BAS webservice Pho2Syl to create a syllable segmentation on the basis of a phonetic segmentation (created by, for example, runBASwebservice_maus). You can provide the level of your word segmentation, or of any other hierarchically dominant segmentation, via the superLevel parameter. This way, the new syllable items can be linked up into the pre-existing hierarchy. If you do not provide this input, the syllables will only be linked down to the segments.

Usage

```
runBASwebservice_pho2sylSegmental(
  handle,
  segmentAttributeDefinitionName,
  language,
  superLevel = NULL,
  sylAttributeDefinitionName = "MAS",
  params = list(wsync = "yes"),
  perspective = "default",
  patience = 0,
  resume = FALSE,
  verbose = TRUE
)
```

Arguments

handle	emuDB handle
segmentAttributeDefinitionName	name of the attribute (not level!) containing a phonetic segmentation.
language	language(s) to be used. If you pass a single string (e.g. "deu-DE"), this language will be used for all bundles. Alternatively, you can select the language for every bundle individually. To do so, you must pass a data frame with the columns session, bundle, language. This data frame must contain one row for every bundle in your emuDB. Up-to-date lists of the languages accepted by all webservices can be found here: https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help
superLevel	name of the segments' parent level (typically the word level). If set to NULL, the syllable level cannot be linked up.
sylAttributeDefinitionName	attribute name for syllable segmentation
params	named list of parameters to be passed on to the webservice. It is your own responsibility to ensure that these parameters are compatible with the webservice API (see https://clarin.phonetik.uni-muenchen.de/BASWebServices/services/help). Some options accepted by the API (e.g. output format) cannot be set when calling a webservice from within emuR, and will be overridden. If file parameters are used please wrap the file path in `httr::upload_file("/path/2/file/rules.nrul")`.
perspective	the webApp perspective that the new level will be added to. If NULL, the new level is not added to any perspectives.
patience	If a web service call fails, it is repeated a further n times, with n being the value of patience. Must be set to a value between 0 and 3.
resume	If a previous call to this function has failed (and you think you have fixed the issue that caused the error), you can set resume=TRUE to recover any progress made up to that point. This will only work if your R temporary directory has not been deleted or emptied in the meantime.
verbose	Display progress bars and other information

Details

All necessary level, link and parameter definitions are created in the process. By default, Pho2Syl is run in word synchronized mode. To override this, call this function with the parameter params=list(wsync="no").

See Also

Other BAS webservice functions: runBASwebservice_all(), runBASwebservice_chunker(), runBASwebservice_g2pForPronunciation(), runBASwebservice_g2pForTokenization(), runBASwebservice_maus(), runBASwebservice_minni(), runBASwebservice_pho2sylCanonical()

---

segmentlist *Segment list*

Description

A segment list is the result type of legacy Emu query.

Format

multi-columned matrix one row per segment

- columnlabel
- columnsegment onset time
- columnsegment offset time
- columnutterance name

See Also

query, demo.vowels

Examples

```
data(demo.vowels)

#demo.vowels is a segment list
demo.vowels
```

serve *Serve EMU database to EMU-webApp*

Description

Serves emuDB media files, SSFF tracks and annotations for EMU-webApp browser GUI http://ips-lmu.github.io/EMU-webApp/

Instructions:

Start and connect (this should happen automatically):

- Call this function to start the server.
- Start a suitable HTML5 capable Web-Browser (Google Chrome, Firefox,...).
- Navigate to the EMU-Webapp URL: http://ips-lmu.github.io/EMU-webApp/.
- Press the 'Connect' button in the EMU-webApp and connect with default URL.
- EMU-webApp loads the bundle list and the first bundles media file, SSFF tracks and annotations.

Disconnect and stop:

- Disconnect and stop the server with the 'Clear' button of the webapp or the reload button of your browser.
- The server can also be stopped by calling `stopAllServers` of the httpuv package

Hints:

- To serve only a subset of sessions or bundles use the parameters `sessionPattern` and/or `bundlePattern`.
- Use the `seglist` parameter to pass in a segment list which was generated using the `query` function. This will allow quick navigation to those segments.

Usage

```
serve(
  emuDBhandle,
  sessionPattern = ".*",
  bundlePattern = ".*",
  seglist = NULL,
  bundleListName = NULL,
  host = "127.0.0.1",
  port = 17890,
  autoOpenURL = "https://ips-lmu.github.io/EMU-webApp/?autoConnect=true",
  browser = getOption("browser"),
  useViewer = TRUE,
  debug = FALSE,
  debugLevel = 0
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
sessionPattern	A regular expression pattern matching session names to be served
bundlePattern	A regular expression pattern matching bundle names to be served
seglist	segment list to use for times anchors and session + bundle restriction (type: emuRsegs)
bundleListName	name of bundleList stored in emuDB/bundleLists subdir to send to EMU-webApp
host	host IP to listen to (default: 127.0.0.1 (localhost))
port	the port number to listen on (default: 17890)
autoOpenURL	URL passed to browseURL function. If NULL or an empty string are passed in browseURL will not be invoked.
browser	argument passed on to browser argument of browseURL (see it's documentation for details)
useViewer	Use the viewer provided by getOption("viewer") (the viewer pane when using RStudio) and host a local version of the EMU-webApp in it. This will clone the current EMU-webApp build (https://github.com/IPS-LMU/EMU-webApp/tree/gh-pages/) into the directory provided by tempdir and serve this local version. A clone will only be performed if no file.path(tempdir(), "EMU-webApp") directory is present. An alternative directory can be also set: options(emuR.emuWebApp.dir="path/to/EMU-webApp") (use if offline functionality is required).
debug	TRUE to enable debugging (default: no debugging messages)
debugLevel	integer higher values generate more detailed debug output

Details

Function opens a HTTP/websocket and waits in a loop for browser requests. Parameter host determines the IP address(es) of hosts allowed to connect to the server. By default the server only listens to localhost. If you want to allow connection from any host set the host parameter to 0.0.0.0. Please note that this might be an safety issue! The port parameter determines the port the server listens on. The host and port parameters are intended only for expert users. When started the R console will be blocked. On successful connection the server sends the session and bundle list of the database referenced by name by parameter dbName or by UUID parameter dbUUID. The Web application requests bundle data for viewing or editing. If a bundle is modified with the EMU-webApp and the save button is pressed the server modifies the internal database and saves the changes to disk. Communication between server and EMU webApp is defined by EMU-webApp-websocket-protocol version 0.0.2 (https://ips-lmu.github.io/The-EMU-SDMS-Manual/app-chap-wsProtocol.html).

Value

TRUE (invisible) if the server was started

Examples

```
## Not run:
## Load EMU database 'myDb' and serve it to the EMU-webApp (opens default HTTP/websocket port 17890)

myDb = load_emuDB("/path/to/myDb")
serve(myDb)

## End(Not run)
```

SetGetlevelCanvasesOrder

Set / Get level canvases order of emuDB

Description

Set / Get which levels of an emuDB to display as level canvases (in a given perspective of the EMU-webApp), and in what order. Level canvases refer to levels of the type "SEGMENT" or "EVENT" that are displayed by the EMU-webApp. Levels of type "ITEM" can always be displayed using the hierarchy view of the web application but can not be displayed as level canvases. For more information on the structural elements of an emuDB see `vignette{emuDB}`.

Usage

```
set_levelCanvasesOrder(emuDBhandle, perspectiveName, order)

get_levelCanvasesOrder(emuDBhandle, perspectiveName)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`perspectiveName`	name of perspective
`order`	character vector containing names of levelDefinitions

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# get level canvases order of ae emuDB
order = get_levelCanvasesOrder(emuDBhandle = ae,
                               perspectiveName = "default")

# reverse the level canvases order of ae emuDB
```

```
set_levelCanvasesOrder(emuDBhandle = ae,
                       perspectiveName = "default",
                       order = rev(order))

# get level canvases order of ae emuDB
get_levelCanvasesOrder(emuDBhandle = ae,
                       perspectiveName = "default")

## End(Not run)
```

SetGetRemoveLegalLabels *Set / Get / Remove legal labels of attributeDefinition of emuDB*

Description

Set / Get / Remove legal labels of a specific attributeDefinition of a emuDB. The legal labels are a character vector of strings that specifies the labels that are legal (i.e. allowed / valid) for the given attribute. As the EMU-webApp won't allow the annotator to enter any labels that are not specified in this array, this is a simple way of assuring that a level has a consistent label set. For more information on the structural elements of an emuDB see `vignette(emuDB)`. Note that defining legal labels for an attributeDefinition does not imply that the existing labels are checked for being 'legal' in the emuDB.

Usage

```
set_legalLabels(emuDBhandle, levelName, attributeDefinitionName, legalLabels)

get_legalLabels(emuDBhandle, levelName, attributeDefinitionName)

remove_legalLabels(emuDBhandle, levelName, attributeDefinitionName)
```

Arguments

`emuDBhandle`	emuDB handle as returned by `load_emuDB`
`levelName`	name of level
`attributeDefinitionName`	name of attributeDefinition (can be and often is the level name)
`legalLabels`	character vector of labels

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)
```

```
legalPhoneticLabels = c("V", "m", "N", "s", "t", "H", "@:", "f", "r",
                        "E", "n", "z", "S", "i:", "w", "@", "k", "I", "d",
                        "db", "j", "u:", "dH", "l", "ai", "O", "D", "o:", "v")

# set legal labels of the
# default "Phonetic" attributeDefinition of
# the "Phonetic" level of ae emuDB
set_legalLabels(emuDBhandle = ae,
                levelName = "Phonetic",
                attributeDefinitionName = "Phonetic",
                legalLabels = legalPhoneticLabels)

# get legal labels of the
# default "Phonetic" attributeDefinition of
# the "Phonetic" level of ae emuDB
get_legalLabels(emuDBhandle = ae,
                levelName = "Phonetic",
                attributeDefinitionName = "Phonetic")


# remove legal labels of the
# default "Phonetic" attributeDefinition of
# the "Phonetic" level of ae emuDB
remove_legalLabels(emuDBhandle = ae,
                   levelName = "Phonetic",
                   attributeDefinitionName = "Phonetic")


## End(Not run)
```

`SetGetSignalCanvasesOrder` *Set / Get signalCanvasesOrder of / to / from emuDB*

Description

Set / Get signalCanvasesOrder array that specifies which signals are displayed in the according perspective by the EMU-webApp. An entry in this character vector refers to either the name of an ssffTrackDefinition or a predefined string: `"OSCI"` which represents the oscillogram or `"SPEC"` which represents the spectrogram. For more information on the structural elements of an emuDB see `vignette{emuDB}`.

Usage

```
set_signalCanvasesOrder(emuDBhandle, perspectiveName, order)

get_signalCanvasesOrder(emuDBhandle, perspectiveName)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
perspectiveName	name of perspective
order	character vector containing names of ssffTrackDefinitions or "OSCI" / "SPEC"

Examples

```
## Not run:

##################################
# prerequisite: loaded ae emuDB
# (see ?load_emuDB for more information)

# get signal canvas order of the "default"
# perspective of the ae emuDB
get_signalCanvasesOrder(emuDBhandle = ae,
                        perspectiveName = "default")


## End(Not run)
```

---

shift *Function to shift the elements of a vector.*

Description

The function makes use of the function 'filter' to delay or advance a signal by k points.

Usage

```
shift(x, delta = 1, circular = TRUE)
```

Arguments

x	A numeric vector
delta	A single element numeric vector. Defines the number of points by which the signal should be shifted.
circular	Logical. If TRUE, the signal is wrapped around itself so that if delta = 1, x[n] becomes x[1]. Otherwise, if delta is positive, the same number of zeros are prepended to the signal

Details

The function makes use of the function 'filter' for linear filtering to carry out the shifting.

Value

The signal shifted by a certain number of points. ...

Author(s)

Jonathan Harrington

See Also

filter

Examples

```
vec = 1:10
shift(vec, 2)
shift(vec, -2)
shift(vec, 2, circular=FALSE)
```

Slope.test *Slope Test*

Description

Tests whether the difference between two or more regression lines is significant

Usage

```
Slope.test(...)
```

Arguments

...	this function takes any number of two column matrices. The first column is the y-data (in the case of locus equations, this is the vowel onset) and the second column is the x-data (in the case of locus equations, vowel target).

Value

The return value consists of the following components:

separate	slope, intercept, r-squared, F-ratio, "d(egrees of) f(reedom)" and "prob(ability that) line fits data" for the separate data matrices entered.
combined	F-ratio, "d(egrees of) f(reedom)", and "Probability of them being DIFFERENT" for the slope and for the intercept of the combined data.
x	the combined x-data for all the matrices.

y	the combined y-data for all the matrices.
mat	the category vectors for the combined data (consists of 1, 0 and -1).
numrows	the number of rows in each matrix.
numcats	the sum number of matrices entered.

References

see E. Pedhazur, Multiple Regression in Behavioral Research p.436-450, 496-507.

See Also

lm(), summary.lm(), pf()

---

sort.emuRsegs *Sort emuRsegs segment list by session, bundle and sample_start*

Description

Sort emuRsegs segment list by session, bundle and sample_start

Usage

```
## S3 method for class 'emuRsegs'
sort(x, decreasing, ...)
```

Arguments

x	object to sort
decreasing	NOT IMPLEMENTED!
...	additional params

---

sortmatrix *Sort matrix by label*

Description

Sorts matrix by label

Usage

```
sortmatrix(mat, labs = dimnames(mat)[[2]])
```

Arguments

mat	A mu+ segment matrix.
labs	A label vector which has the same number of columns as mat.

Value

Returns a sorted matrix by label, created from `mat`.

See Also

label, phon

`splitstring` *Split a string into words.*

Description

Splits a string into words.

Usage

```
splitstring(str, char)
```

Arguments

`str`	A string.
`char`	A character to split on

Value

A vector of strings. The original `str` is split at ever occurrence of `char` to generate a vector of strings.

Examples

```
splitstring("/home/recog/steve/foo", "/")
#[1] "home" "recog" "steve" "foo"
```

`start.emusegs` *Start and end times for EMU segment lists and trackdata objects*

Description

Obtain start and end times for EMU segment lists and trackdata objects

Arguments

`x`	a segment list or a trackdata object
`...`	due to the generic only

Details

The function returns the start and/or end times of either a segment list or a trackdata object. The former refers to the boundary times of segments, the latter the start and end times at which the tracks from segments occur. start.emusegs and end.emusegs give exactly the same output as start and end respectively.

Value

A vector of times.

Author(s)

Jonathan Harrington

See Also

`tracktimes`

Examples

```
# start time of a segment list
start(polhom)
# duration of a segment list
end(polhom) - start(polhom)
# duration from start time of segment list
# and start time of parallel EPG trackdata
start(polhom) - start(polhom.epg)
```

summary.emuDBhandle *Print summary of loaded EMU database (emuDB).*

Description

Gives an overview of an EMU database. Prints database name, UUID, base directory path, session and bundle count and informations about signal track, annotation level, attribute and link definitions.

Usage

```
## S3 method for class 'emuDBhandle'
summary(object, ...)
```

Arguments

object	emuDBhandle as returned by load_emuDB
...	additional arguments affecting the summary produced.

track.gradinfo *Calculate gradient summary information for trackdata*

Description

Calculates a number of summary measures for a trackdata object: duration, start and end data points, delta values and slope.

Usage

```
track.gradinfo(trackdata)
```

Arguments

trackdata	An Emu trackdata object as returned by get_trackdata

Details

track.gradinfo calculates a number of summary measure for the segments within a trackdata object. These are useful for data such as kinematic measures where segments might correspond to articulatory movements etc.

Measures returned are: duration, start and end data values (ie. the first and last rows of data for each segment), delta (the difference between the first and last rows of data) and slope (delta divided by the duration).

Value

A data frame with one row per segment and columns:

`duration`	Segment
`startN`	The starting value for each segment (start1 is the starting value for the first column)
`endN`	The ending value for each segment
`deltaN`	The delta value for each segment
`slopeN`	The slope value for each segment

Since the result is a data frame, the columns can be referred to by name (`result$duration`) or as matrix columns (`result[,1]`).

Author(s)

Steve Cassidy

See Also

`get_trackdata`, `dapply`

Examples

```
data(vowlax)
segs = vowlax
## fm has 4 columns
data.fm <-vowlax.fdat
## F0 has one
data.F0 <- vowlax.fund
## info.fm will have duration, 4xstart, 4xend, 4xdelta, 4xslope
info.fm <- track.gradinfo(data.fm)
## this should be true
ncol(info.fm) == 1+4+4+4+4

## info.F0 will have one of each
info.F0 <- track.gradinfo(data.F0)
## this should be true
ncol(info.F0) == 1+1+1+1+1

## plot the durations vs delta of the first formant
plot(info.F0$duration, info.fm$delta1, type="n", xlab="Duration", ylab="Delta")
text(info.fm$duration, info.fm$delta1, labels=label(segs))

## extract just the delta values from the formant info
## You need to eyeball the data to work out which columns to select
delta.fm <- info.fm[,10:13]
```

trackdata *Track data object*

Description

A track data object is the result of get_trackdata().

Format

$index a two columned matrix, each row keeps the first and last index of the $data rows that belong to one segment

$ftime a two columned matrix, each row keeps the times marks of one segment

$data a multi-columned matrix with the real track values for each segment

Methods

The following generic methods are implemented for trackdata objects.

list("Arith") "+", "-", "*", "^", "%%", "%/%", "/"

list("Compare") "==", ">", "<", "!=", "<=", ">="

list("Logic") "&", "|".

list("Ops") "Arith", "Compare", "Logic"

list("Math") "abs", "sign", "sqrt", "ceiling", "floor", "trunc", X "cummax", "cummin", "cumprod", "cumsum", "log", "log10", "log2", "log1p", "acos", "acosh", "asin", "asinh", "atan", "atanh", "exp", "expm1", "cos", "cosh", "sin", "sinh", "tan", "tanh", "gamma", "lgamma", "digamma", "trigamma"

list("Math2") "round", "signif"

list("Summary") "max", "min", "range", "prod", "sum", "any", "all"

Note

The entire data track is retrieved for each segment in the segment list. The amount of data returned will depend on the sample rate and number of columns in the track requested.

See Also

get_trackdata, demo.vowels.fm demo.all.rms

Examples

```
data(demo.vowels.fm)
data(demo.vowels)

#Formant track data for the first segment of the segment list demo.vowels
demo.vowels.fm[1]
```

trackfreq *function to find the frequencies of a spectral object*

Description

Find the frequencies of a spectral object.

Usage

```
trackfreq(specdata)
```

Arguments

specdata	A spectral object

Value

A vector of the frequencies at which the columns of a spectral matrix occur.

Author(s)

Jonathan Harrington

Examples

```
trackfreq(vowlax.dft.5)
# Frequency components between 1000 and 2000 Hz
trackfreq(vowlax.dft.5[,1000:2000])
# All frequency components of a trackdata object except the d.c. offset
trackfreq(fric.dft[,-1])
# All frequency components except the d.c. offset
# and except frequencies above 5000 Hz
trackfreq(fric.dft[,-c(1, 5000:20000)])
# Note the following syntax if the spectral object is a vector
# Frequencies 1000-3000 Hz
trackfreq(e.dft[1000:3000])
```

`tracktimes` *Get the track times from EMU trackdata objects*

Description

The function obtains the times at which track values occur.

Usage

```
tracktimes(trackdata)
```

Arguments

`trackdata`	An EMU trackdata object, or a matrix of track values obtained at a single time point using dcut()

Details

Every $data value in a trackdata object is associated with a time at which it occurs in the utterance. This function returns those times.

Author(s)

Jonathan Harrington

See Also

`start.trackdata` `end.trackdata` `start.emusegs` `end.emusegs`

Examples

```
# track time values for a trackdata object
times <- tracktimes(vowlax.fdat)
# track time values for a matrix of trackdata values
# at  the temporal midpoint
tracktimes(dcut(vowlax.fdat[1:3,], 0.5, prop=TRUE))
```

`train` *Train a Gaussian Model*

Description

Trains a Gaussian Model

Usage

```
train(x, lab = rep("x", nrow(x)))
```

Arguments

`x`	A data vector or matrix.
`lab`	A vector of labels parallel to `x`. If missing, all data is assumed to be from the same class.

Details

This function is used to train a gaussian model on a data set. The result can be passed to either the `mahal` or `bayes.lab` functions to classify either the training set (`x`) or a test set with the same number of dimensions. Train simply finds the mean and inverse covariance matrix/standard deviation for the data corresponding to each unique label in labs.

Value

A structure with the following components:

`label`	The unique labels in `lab`.
`means`	The means for each dimension per unique label.
`cov`	The combined covariance matrixes for each unique label. The matrixes are joined with `rbind`. If the input data is one-dimensional, this is just the standard deviation of the data.
`invcov`	The combined inverse covariance matrixes for each unique label. The matrixes are joined with `rbind`. If the input data is one-dimensional, this is just the reciprocal of the standard deviation of the data.

See Also

mahal, bayes.lab, mahalplot, bayes.plot

`trapply` *A method of the generic function by for objects of class 'trackdata'*

Description

A given function 'FUN' is applied to the data corresponding to each segment of data.

Usage

```
trapply(trackdata, fun, ..., simplify = FALSE, returntrack = FALSE)
```

Arguments

`trackdata`	a track data object
`fun`	a function that is applied to each segment
`...`	arguments of the function fun
`simplify`	simplify = TRUE , output is a matrix; simplify = FALSE a list is returned
`returntrack`	returntrack = FALSE , return a trackdata object

Details

trapply() applies a function iteratively to each segment of a trackdata object without the need for using a for-loop. It can be used to calculate, for example, the mean value of the data values of each segment separately. Any function that can be applied sensibly to trackdata[j]$data where j is a segment number can be used as the fun argument to trapply(). It is also possible to write your own function and use trapply() to apply it separately to each segment. Care needs to be taken in using trapply() in the following two ways. Firstly, the argument simplify=TRUE should only be set if it can be guaranteed that a vector of the same length or matrix of the same number of rows as the number of segments in the trackdata object is returned. For example, simplify=TRUE can be used in calculating the mean per segment of a trackdata object, because there will only be one value (the mean) per segment. However, simplify should be set to FALSE in calculating the range because here two values are returned per segment. Similarly use simplify=FALSE n smoothing the data in which the number of values returned per segment is different. Secondly, trapply() only applies a function to a single parameter; the function can be used to apply to a function to multi-parameter trackdata such as F1-F4, but then the function needs to be put inside apply() - see examples below.

Value

list or vector or matrix

Author(s)

Jonathan Harrington

See Also

`apply`

Examples

```
# mean f0 one value per segment
m = trapply(vowlax.fund, mean, simplify=TRUE)
# mean F1 - F4
m = trapply(vowlax.fdat, apply, 2, mean, simplify=TRUE)
# make a logical vector of any segments that have an F1 value
# between their start time and end time greater than n Hz
pfun <- function(x, n=1000) any(x > n)
# greater than 1100 Hz
temp = trapply(vowlax.fdat[,1], pfun, 1100, simplify=TRUE)
# get the F2-range per segment
r = trapply(vowlax.fdat[,2], range)
# F2-range of 20th segment
r[[20]]
# DCT-smooth F2 with 10 coeffs
# get the first 4 DCT coefficients
f2.dct = trapply(vowlax.fdat[,2], dct, 3, simplify=TRUE)
# dct-smooth F2 with the first 5 DCT coeffs
f2sm = trapply(vowlax.fdat[,2], dct, 4, TRUE,  returntrack=TRUE)
# Make new F2 trackdata such that each segment has
# F2 divided by its F2 range
pfun <- function(x) x/(diff(abs(range(x))))
newf2 = trapply(vowlax.fdat[,2], pfun, returntrack=TRUE)
```

---

update_itemsInLevel *Update items programmatically*

Description

Update items programmatically

Usage

```
update_itemsInLevel(
  emuDBhandle,
  itemsToUpdate,
  rewriteAllAnnots = TRUE,
  verbose = TRUE
)
```

Arguments

emuDBhandle	emuDB handle as returned by load_emuDB
itemsToUpdate	A data frame with the columns • session, • bundle,

- `level`,
- `start_item_seq_idx` (`start_item_seq_idx` is used instead of e.g. `seq_idx` so that the result of a `query` call can be used directly. `query` can return a sequence of items defined by `start_item_seq_idx` and `end_item_seq_idx` which have the same value if single items are returned),
- `attribute`, and
- `labels`.

None of the columns should be factors. `sequenceIndex` must be numeric (natural-valued), all other columns must be of type character.

`rewriteAllAnnots` should changes be written to file system (_annot.json files) (intended for expert use only)

`verbose` if set to `TRUE`, more status messages are printed

---

vowlax

Segment list of four lax vowels, read speech, one male and one female speaker of Standard North German from database kielread.

Description

An EMU dataset

Format

segmentlist

---

vowlax.df

Data frame of various parameters and labels from the segment list vowlax

---

Description

An EMU dataset

Format

dataframe

`vowlax.dft.5` *Spectral matrix centred at the temporal midpoint of the vowels from the segment list vowlax.*

Description

An EMU dataset

Format

spectral matrix

`vowlax.fdat` *Trackdata of formants from the segment list vowlax*

Description

An EMU dataset

Format

trackdata object

`vowlax.fdat.5` *Matrix of formant data extracted at the temporal midpoint from the segment list vowlax.*

Description

An EMU dataset

Format

matrix of formant data

`vowlax.fund` *Trackdata of fundamental frequency from the segment list vowlax*

Description

An EMU dataset

Format

trackdata object

vowlax.fund.5 — *Vector of fundamental frequency extracted at the temporal midpoint from the segment list vowlax.*

Description

An EMU dataset

Format

vector of fundamental frequency

vowlax.l — *Vector of phoneme labels from the segment list vowlax*

Description

An EMU dataset

Format

vector of phoneme labels

vowlax.left — *Vector of labels preceding the vowels from the segment list vowlax*

Description

An EMU dataset

Format

vector of phoneme labels

vowlax.right — *Vector of labels following the vowels from the segment list vowlax*

Description

An EMU dataset

Format

vector of phoneme labels

vowlax.rms — *Trackdata of RMS energy from the segment list vowlax*

Description

An EMU dataset

Format

trackdata object

vowlax.rms.5 — *Vector of RMS energy values at the temporal midpoint extracted at the temporal midpoint from the segment list vowlax*

Description

An EMU dataset

Format

vector of RMS energy values

vowlax.spkr — *Vector of speaker labels from the segment list vowlax.*

Description

An EMU dataset

Format

vector of speaker labels

vowlax.word — *Vector of word labels from the segment list vowlax.*

Description

An EMU dataset

Format

vector of word labels

wordlax.l — *Vector of word labels from segment list wordlax*

Description

For wordlax (see data(vowlax))

Format

vector of word labels

write.emusegs — *Write an Emu segment list to a file*

Description

Writes an Emu segment list to a file

Usage

```
write.emusegs(seglist, file)
```

Arguments

seglist	An Emu segment list
file	The name of a file to write the segment list into.

Value

None.

Side Effects

The segment list is written to a file in the standard format, suitable for input to gettrack or other Emu utility programs.

See Also

query

Examples

```
data(dip)
#dip a segment list - first 10 segments only
dip[1:10,]
## Not run: write.emusegs(dip, "write.emusegs.example.txt")

#The file write.emusegs.example.txt would have been written to R_HOME
## Not run: unlink("write.emusegs.example.txt")
```

`write_bundleList` *write bundleList*

Description

write bundleList JSON file to emuDB

Usage

```
write_bundleList(
  emuDBhandle,
  name,
  bundleList,
  seglist,
  updateDBconfig = TRUE,
  verbose = TRUE
)
```

Arguments

`emuDBhandle`	emuDB handle object (see load_emuDB)
`name`	name of bundleList (excluding the _bundleList.json suffix)
`bundleList`	tibble/data.frame with the columns `session`, `name`, `comment` (optional), `finishedEditing` (optional). Use list_bundles
`seglist`	segment list returned by query function. If set the `bundleList` parameter will be ignored and a bundleList will be created by collapsing the segments as timeAnchors into the `_bundleList.json`
`updateDBconfig`	if set to TRUE (the default) DBconfig will be updated with the fields
`verbose`	be verbose

- `"bundleComments": true`
- `"bundleFinishedEditing": true`

Details

Write bundleList JSON file to emuDB sub-dir `bundleLists/`

Index