Package 'monitoR'

May 9, 2026

Type Package

Title Acoustic Template Detection in R

Version 1.2

Date 2025-04-10

Maintainer Sasha D. Hafner <sasha.hafner@bce.au.dk>

Depends R (>= 2.10), tuneR, methods

Imports graphics, grDevices, stats, utils

Suggests fftw, parallel, RODBC, knitr

Description Acoustic template detection and monitoring database interface. Create, modify, save, and use templates for detection of animal vocalizations. View, verify, and extract results. Upload a MySQL schema to a existing instance, manage survey metadata, write and read templates and detections locally or to the database.

License GPL-2

URL https://github.com/jonkatz2/monitor

VignetteBuilder knitr

NeedsCompilation no

Author Sasha D. Hafner [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0955-0327>),
Jon Katz [aut],
Jerome Sueur [aut] (seewave package author (code from Fourier transform used in monitoR)),
Thierry Aubin [aut] (seewave package author (code from Fourier transform used in monitoR)),
Caroline Simonis [aut] (seewave package author (code from Fourier transform used in monitoR)),
Uwe Ligges [aut] (tuneR package author (code from readMP3() used in monitoR)),
Therese Donovan [ctb] (creative direction and database design support)

Repository CRAN

Date/Publication 2025-04-11 12:10:02 UTC

Contents

---

batchDetection *Batch Template Detection*

Description

These functions are used to carry out template dection for multiple template and survey files in a single call. These functions make it easy to analyze multiple survey files in a single call. They call corMatch or binMatch, followed by findPeaks and getDetections to do the work.

Usage

```
batchCorMatch(dir.template, dir.survey = ".", ext.template = "ct", ext.survey = "wav",
   templates, parallel = FALSE, show.prog = FALSE, cor.method = "pearson", warn = TRUE,
    time.source = "filename", fd.rat = 1, ...)

batchBinMatch(dir.template, dir.survey = ".", ext.template = "bt", ext.survey = "wav",
    templates, parallel = FALSE, show.prog = FALSE, warn = TRUE,
    time.source = "filename", fd.rat = 1, ...)
```

Arguments

dir.template	A file path to a directory that contains template files to be used. Only used if template is missing.
dir.survey	A file path to a directory that contains survey files to be analyzed.
ext.template	Extension of the template files.
ext.survey	Extension of the survey files.
templates	A template list–a corTemplateList object for corMatch or a binTemplateList object for binMatch. If templates is missing, all the template files in dir.template will be used instead.
parallel	If TRUE, mclapply from the **parallel** package is used for calculation of scores across all time bins for each template. This option is not available for Windows operating systems.
show.prog	If TRUE, progress will be reported during the score calculations.
cor.method	For corMatch, the method used to calculate correlation coefficients (see cor).
warn	Set to FALSE to suppress warnings about step mismatches.
time.source	The source of date and time information. filename will look in the name of the survey file (survey argument) for a date and time with format YYYY-MM-DD_HHMMSS_TimeZone. fileinfo will take the date and time from the file modification information.
fd.rat	A ratio of frame width (twice minimum peak separation) to template duration. Used by findPeaks.
...	Additional arguments to the spectro function.

Details

These functions are simple but do not provide flexibility in how results are handled. Manually writing a `for` loop is a more flexible solution.

Value

A data frame of detections, as returned by `getDetections`.

Author(s)

Sasha D. Hafner

See Also

`corMatch`, `binMatch`, `findPeaks`, `getDetections`

Examples

```
## Not run:
# Assume multiple survey files are in the subdirectory "Surveys" and templates
# are in subdirectory "Templates"
detects <- batchCorMatch("Templates", "Surveys")

# Or, to use an existing template list instead
detects <- batchCorMatch(templates = ctemps, dir.survey = "Surveys")

## End(Not run)
```

`bindEvents` *Summarize/Archive Manually Derived Sound Events*

Description

Read in a table of song event times and the corresponding `Wave` object, extract the song events, and bind them into a single `Wave` object for archiving or comparison viewing.

Usage

```
bindEvents(rec, file, by.species = TRUE, parallel = FALSE, return.times = FALSE)
```

Arguments

rec	File path to mp3 or wav file or object of class Wave
file	File path to csv file containing event times. See details.
by.species	Logical. Should each species be in its own Wave object?
parallel	Logical. FALSE will use lapply, TRUE will use mclapply.
return.times	Logical. FALSE returns only the Wave object with events. TRUE will also return a data frame with the start and end times of each event in the new Wave object linked to their original start and end times.

Details

The csv file supplied must use a standard set of column names, which can occur in any order:

name Species name

start.time Event start time, in seconds

end.time Event end time, in seconds

These column names are those supplied in an annotation file produced by viewSpec.

Value

If return.times = FALSE, an object of class Wave.
If return.times = TRUE, a list:

times	A data frame with the start and end times of events in the Wave object
wave	An object of class Wave

Author(s)

Sasha D. Hafner

See Also

viewSpec, collapseClips, bind.

Examples

```
data(survey_anno)
data(survey)

# Don't return times
events <- bindEvents( rec = survey, file = survey_anno, by.species = TRUE, parallel = FALSE,
                      return.times = FALSE)

# Return times
events <- bindEvents( rec = survey, file = survey_anno, by.species = TRUE, parallel = FALSE,
                      return.times = TRUE)
```

btnw *Black-Throated Green Warbler (*Setophaga virens*) Song*

Description

A 3 second wave recording of a Black-throated Green Warbler (*Setophaga virens*) song.

Usage

```
data(btnw)
```

Format

The format is:
Formal class 'Wave' [package "tuneR"] with 6 slots ..@ left : int [1:72001] -53 -65 -32 44 -15 -37 -5 26 26 55 ... ..@ right : num(0) ..@ stereo : logi FALSE ..@ samp.rate: int 24000 ..@ bit : int 16 ..@ pcm : logi TRUE

Source

Sound clips were recorded in Vermont, USA in 2010. Equipment was a Wildlife Acoustics SM1(TM) recorder recording in WAC0 format, converted to wave using the Wildlife Acoustics Wac2Wav (TM) converter. Recording has a sample rate of 24kHz and is 16-bit mono.

Examples

```
data(btnw)

viewSpec(btnw)
```

changeSampRate *Resample Wave objects*

Description

Downsample or upsample Wave objects by specifying either a new sample rate or matching the sample rate of a different Wave object. Optional adjustable dithering.

Usage

```
changeSampRate(wchange, wkeep = NULL, sr.new = wkeep@samp.rate, dither = FALSE,
               dith.noise = 32)
```

Arguments

wchange	Object of class Wave to resample.
wkeep	Object of class Wave to use to match sampling rate, or specify sampling rate with sr.new.
sr.new	Numerical sampling rate, if specified directly.
dither	Logical. TRUE adds gaussian dithering.
dith.noise	Adjustable dithering. If dither = TRUE, this value will be the stdev of the normally distributed noise.

Details

Both downsampling and upsampling are done by spline-fitting a curve to the waveform and resampling the resulting waveform. Artifacts from resampling are nearly guaranteed. Artifacts can be masked with dithering at a cost: dithering raises the amplitude of background noise but not signal.

Value

An object of class Wave with a modified sample rate.

Author(s)

Sasha D. Hafner, Jon Katz

See Also

downsample

Examples

```
data(survey)

survey <- changeSampRate(wchange = survey, sr.new = 24000)
```

---

collapseClips *Summarize/Archive Song Events*

Description

Read in a Wave object, extract the song events, and bind them into a single Wave object for archiving or comparison viewing.

Usage

```
collapseClips(rec, start.times, end.times, return.times = FALSE)
```

Arguments

rec	Object of class Wave or file path to wave file.
start.times	Vector of event start times, in seconds.
end.times	Vector of event end times, in seconds.
return.times	Logical. TRUE will return

Details

A stripped-down version of bindEvents, perhaps more readily applied to the output of findPeaks.

Value

If return.times = FALSE, an object of class Wave. If return.times = TRUE, a list:

times	A data frame with the start and end times of events in the wave object
wave	An object of class Wave

Author(s)

Sasha D. Hafner

See Also

viewSpec, bindEvents, bind.

Examples

```
data(survey_anno)

data(survey)

events <- collapseClips(rec = survey, start.times = survey_anno[, "start.time"],
                        end.times = survey_anno[, "end.time"], return.times = FALSE)
```

---

combineTemplates *Combine Acoustic Template Lists*

Description

Use these functions to combine any number of templates together into a larger template list. They can combine template lists that themselves contain any number of templates.

Usage

```
combineCorTemplates(...)

combineBinTemplates(...)
```

Arguments

...	Correlation *or* binary template lists (class corTemplateList or binTemplateList), or a single list of such.

Details

These functions are the only way to create template lists containing more than one template, and so should be used often. Only binTemplateList objects should be used with combineBinTemplates, and only corTemplateList objects should be used with combineCorTemplates. If you combine templates that use the same name, a suffix (.2) will be added to the later name.

Value

A TemplateList object that contains all the templates submitted to the function.

Author(s)

Sasha D. Hafner

See Also

makeCorTemplate, makeBinTemplate, templateNames

Examples

```
# First need to make some template lists to combine
# Load data
data(btnw)
data(oven)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")

oven.fp <- file.path(tempdir(), "oven.wav")

writeWave(btnw, btnw.fp)

writeWave(oven, oven.fp)

# Create four correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")

wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6),
                        name = "w2")

oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")

oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1,
                        name = "o2")
```

```
# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)
ctemps

# Binary templates are similar
# Create four templates
wbt1 <- makeBinTemplate(btnw.fp, amp.cutoff = -40, name = "w1")

wbt2 <- makeBinTemplate(btnw.fp, amp.cutoff = -30, t.lim = c(1.5, 2.1),
                        frq.lim = c(4.2, 5.6), buffer = 2, name = "w2")

obt1 <- makeBinTemplate(oven.fp, amp.cutoff = -20, t.lim = c(1, 4),
                        frq.lim = c(1, 11), name = "o1")

obt2 <- makeBinTemplate(oven.fp, amp.cutoff = -17, t.lim = c(1, 4),
                        frq.lim = c(1, 11), buffer = 2, name = "o2")

# Combine all of them
btemps <- combineBinTemplates(wbt1, wbt2, obt1, obt2)
btemps

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)

file.remove(oven.fp)
```

`compareTemplates` *Compare Performance of Templates*

Description

Provided a `detectionList` object containing results from N templates scored against the same survey with Y song events, `compareTemplates` will create a Y x N matrix to compare how each template scored each song event. If the song events are the sound clips used to create each template, `compareTemplates` may be a means of measuring overall similarity among sound events. Can be used to identify template clips that may match more than one song type.

Usage

```
compareTemplates(detection.obj, cutoff.return, cutoff.ignore, tol, n.drop = 0)
```

Arguments

`detection.obj`	Object of class `detectionList`.
`cutoff.return`	Score cutoff below which events are not returned.
`cutoff.ignore`	Score cutoff below which events are ignored.
`tol`	Tolerance (s). If a peak is within `tol` of a peak from another template, they are in the same event.
`n.drop`	Rows with this many templates or fewer will be dropped. `n.drop = 0` drops none.

Details

The matrix is created by comparing the score for each event to the average score for that event. For cases in which a template does not score an event above `cutoff` a value of `NA` is placed in the matrix for that template-event junction. Similarly, if a template scores an event above `cutoff` but is beyond `tol` of the mean of other events, it will enter the matrix as its own event and an `NA` will be placed in the matrix for the event's junctions with other templates.

Value

A list:

`times.mean`	Vector of mean times for each row of the matrix.
`times`	Matrix of times for each event detection and template.
`scores`	Matrix of scores for each event detection and template.

Note

It can be difficult to make this function do the same grouping of peaks that a human might do.

Author(s)

Sasha D. Hafner

See Also

`makeCorTemplate`, `makeBinTemplate`

Examples

```
# Load data
data(btnw)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")

writeWave(btnw, btnw.fp)

# Make three templates to compare
btnw.1 <- makeBinTemplate(clip = btnw.fp, frq.lim = c(2.75, 7), t.lim = c(.5, 2.5),
                          amp.cutoff = -20, name = -20)

btnw.2 <- makeBinTemplate(clip = btnw.fp, frq.lim = c(2.75, 7), t.lim = c(.5, 2.5),
                          amp.cutoff = -27, name = -27)

btnw.3 <- makeBinTemplate(clip = btnw.fp, frq.lim = c(2.75, 7), t.lim = c(.5, 2.5),
                          amp.cutoff = -34, name = -34)

# Combine templates
templates <- combineBinTemplates(btnw.1, btnw.2, btnw.3)

survey <- bind(btnw, btnw, btnw)
```

```
survey.fp <- file.path(tempdir(), "survey.wav")

writeWave(survey, survey.fp)

scores <- binMatch(survey = survey.fp, templates = templates, time.source = "fileinfo")

pks <- findPeaks(scores)

compareTemplates(detection.obj = pks, cutoff.return = 12, cutoff.ignore = 6, tol = 1,
                 n.drop = 0)

# Clean up
file.remove(btnw.fp)
file.remove(survey.fp)
```

---

cutWave *Extract Shorter Wave Objects from other Wave Objects*

Description

Extract shorter Wave objects from other Wave objects. Extracted wave object will be between the `from` and `to` boundaries.

Usage

```
cutWave(wave, from = NULL, to = NULL)
```

Arguments

`wave`	Object of class `Wave`.
`from`	Start extracted segment from this point, in seconds from beginning of `Wave` object.
`to`	End of extracted segment, in seconds from beginning of `Wave` object.

Details

This function is a simplified version of `cutw` from the seewave package. Its original name in the monitoR was the same (`cutw`), but has since been changed to avoid conflict for those who use both packages.

Value

An object of class `Wave`.

Author(s)

Sasha D. Hafner

Examples

```
data(survey)

event1 <- cutWave(wave = survey, from = 1.5, to = 4.75)
```

---

dbDownload *Retrieve Card-Recorder ID Values or Survey Names from a Database*

Description

Convenience functions to execute a prewritten SQL query. Wrappers for `RODBC::sqlQuery` with no additional processing.

Usage

```
dbDownloadCardRecorderID(db.name = "acoustics", uid, pwd,
                         date.deployed, date.collected,
 loc.prefix, ...)

dbDownloadSurvey(db.name = "acoustics", uid, pwd, start.date,
                 end.date, loc.prefix, samp.rate, ext, ...)
```

Arguments

`db.name`	Name of the ODBC connector data source corresponding to the acoustics database.
`uid`	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
`pwd`	Password to allow ODBC connector to connect to database, if not present in ODBC connector.
`date.deployed`, `date.collected`, `start.date`, `end.date`	Dates to filter results, as a character string formatted to your database storage; in the example we use `YYYY/MM/DD`, but be aware that you may need to include a full timestamp: `YYYY/MM/DD 00:00:00`.
`loc.prefix`	Location prefix or vector of six-character prefixes by which to filter results.
`samp.rate`	Numerical sampling rate of surveys (Hz).
`ext`	Character file extension "wav" or "mp3".
`...`	Additional arguments to `RODBC::odbcConnect`.

Details

These functions assume a database structure identical to that provided in the acoustics schema. `dbDownloadCardRecorderID` may be used to look up CardRecorderID values before uploading survey metadata; `dbDownloadSurvey` may be used to generate a table of survey names to work through for batch detection with either `corMatch` or `binMatch`. If the username and password are present in the ODBC datasource they do not need to be provided. It is possible to store only the username in the datasource and enter a password, but the reverse will not work.

Value

dbDownloadCardRecorderID returns a data frame with fields pkCardRecorderID, fldLocationNameAbbreviation, fldSerialNumber, and pkCardID. dbDownloadSurvey returns a data frame with a single field: fldSurveyName.

Note

These are convenience functions for users who are unfamiliar with SQL syntax and/or have not established an alternative front-end for their acoustics database. Users capable of doing so may find more utility and flexibility writing custom queries directly either with an alternative front-end or RODBC::sqlQuery. No processing is performed; data from the database is returned as it exists in the database.

Author(s)

Jon Katz

See Also

sqlQuery, dbDownloadTemplate, dbUploadSurvey

Examples

```
## Not run:
#If using the 'acoustics' schema verbatim:
CRs <- dbDownloadCardRecorderID(
        date.deployed = "2012/05/22",
        date.collected = "2012/05/29",
        loc.prefix = "MABI01")

surveys <- dbDownloadSurvey(
        start.date = "2012/05/22",
        end.date = "2012/05/29",
        loc.prefix = "MABI01",
        samp.rate = 24000,
        ext = "wav")

#'acoustics' schema, different database name:
CRs <- dbDownloadCardRecorderID(
        db.name = "LocalSQLdb",
        uid = "EntryOnly",
        pwd = "07H23BBM",
        date.deployed = "2012/05/22",
        date.collected = "2012/05/29",
        loc.prefix = "MABI01")

surveys <- dbDownloadSurvey(
        db.name = "LocalSQLdb",
        uid = "EntryOnly",
        pwd = "07H23BBM",
        start.date = "2012/05/22",
```

```
        end.date = "2012/05/29",
        loc.prefix = "MABI01",
        samp.rate = 24000,
        ext = "wav")

## End(Not run)
```

---

dbDownloadResult *Create* detectionList *Objects from Data Stored in a Database*

Description

This function creates detectionList objects corresponding to a specified survey and TemplateList from data available in an acoustics database.

Usage

```
dbDownloadResult(db.name = "acoustics", uid, pwd, survey, templates,
                 type, FFTwl, FFTwn, FFTovlp, ...)
```

Arguments

db.name	Name of the ODBC connector data source corresponding to the acoustics database.
uid	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
pwd	Password to allow ODBC connector to connect to database, if not present in ODBC connector.
survey	Character value, name of survey as it appears in the acoustics database
templates	object of class TemplateList or character vector of template names as they appear in an acoustics database
type	Character value in c("BIN", "COR") to filter the results for either binMatch or corMatch results, respectively
FFTwl	Filter for templates with specific FFT window lengths.
FFTovlp	Filter for templates with specific FFT window overlap.
FFTwn	Filter for templates with specific FFT window names.
...	Additional arguments to sqlQuery. For example, if the function fails on an error such as: Error in as.POSIXlt.character(x, tz, ...) : character string is not in a standard unambiguous format, adding as.is = TRUE may help circumnavigate the problem (although it will not solve the data issue!)

Details

This function allows database data to be coerced back into an object of class `detectionList`, which is useful in that data can be pulled from the database and used in functions that require `detectionList` objects such as `plot` and `showPeaks`.

The resulting `detectionList` object will be incomplete as it is missing the complete scores list, which is used to plot the scores in the second row of the above plotting functions. Hit markers are still plotted, and these can still be useful if set to `hit.marker = "points"`.

Value

An object of class `detectionList`

Author(s)

Jon Katz, Sasha D. Hafner

See Also

`detectionList`, `TemplateList`, `binMatch`, `corMatch`, `showPeaks`

Examples

```
## Not run:
#If using the 'acoustics' schema verbatim:
examp <- dbDownloadResult(
        survey = "INTV02_2011-06-25_081000_EDT.mp3",
        templates = templates, type = "BIN")

#'acoustics' schema, different database name:
examp <- dbDownloadResult(
        db.name = "LocalSQLdb",
        uid = "EntryOnly" ,
        pwd = "07H23BBM",
        survey = "INTV02_2011-06-25_081000_EDT.mp3",
        templates = templates,
        type = "BIN")
## End(Not run)
```

---

dbDownloadTemplate *Retrieve templates from an acoustics database*

Description

Download Acoustic Templates from a Database

Usage

```
dbDownloadTemplate(db.name = "acoustics", uid, pwd, type, names,
                   species, FFTwl, FFTovlp, FFTwn, ...)
```

Arguments

db.name	Name of the ODBC connector data source corresponding to the acoustics database.
uid	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
pwd	Password to allow ODBC connector to connect to database, if not present in ODBC connector.
type	Type of templates to select. Character value of either "BIN" or "COR". Some partial matching is performed to accept "bt" and "ct", for example.
names	Optional character value or vector of template names to filter selection from the database. If missing all templates matching other filters are selected.
species	Optional character value or vector of species to filter selection from the database. If missing all templates matching other filters are selected.
FFTwl	Optional character value or vector of FFT window lengths to filter selection from the database. If missing all templates matching other filters are selected.
FFTovlp	Optional character value or vector of FFT window overlap to filter selection from the database. If missing all templates matching other filters are selected.
FFTwn	Optional character value or vector of FFT window names to filter selection from the database. If missing all templates matching other filters are selected.
...	Additional arguments to odbcConnect.

Details

This function assumes a database structure identical to that provided in the acoustics schema. If the username and password are present in the ODBC datasource they do not need to be provided. It is possible to store only the username in the datasource and enter a password, but the reverse will not work.

Value

An object of class TemplateList.

Note

In the acoustics database templates are broken into components, and vectors are stored as text objects in various fields. To stay beneath the maximum download vector size of sqlQuery, extraneous characters are removed from each vector during upload; some must be re-inserted during download. Space characters are not replaced, but all amplitude values for correlation templates are sign-inverted and converted from integers to floating point decimal. All decimals were rounded to the hundredth's place during upload. These measures are sometimes insufficient and users may find it useful to increase the maximum download vector size in sqlQuery (see the vignette "MySQL_DataSources_RODBC" for further details). Large templates may take more than several seconds to download; 2-10 seconds is normal for binary point matching templates, and 5-30 seconds is normal for correlation templates.

Author(s)

Jon Katz

See Also

dbUploadTemplate

Examples

```
## Not run:
#If using the 'acoustics' schema verbatim:
btnw <- dbDownloadTemplate(
        type = "BIN",
        names= c("template1", "template2")
        FFTwl = 512,
        FFTovlp = 0,
        FFTwn = "hanning")

#'acoustics' schema, different database name:
btnw <- dbDownloadTemplate(
        db.name = "LocalSQLdb",
        uid = "EntryOnly" ,
        pwd = "07H23BBM",
        type = "COR",
        species = c("BTNW", "OVEN")
        FFTwl = 512,
        FFTovlp = 0,
        FFTwn = "hanning")
## End(Not run)
```

---

dbSchema *Upload a MySQL Database Schema to Create Tables in an Acoustics Database*

Description

Use this function to select a schema and upload it to an existing MySQL database. All tables in the schema will be created in the database.

Usage

```
dbSchema(schema, name.on.host, tables = FALSE,
         schema.name = "NOH", db.name = "acoustics", uid, pwd,
         ...)
```

Arguments

schema	File path to schema (.txt or .sql).
name.on.host	Database name on MySQL host.
tables	TRUE will return the result of sqlTables
schema.name	Current name of schema to be replaced by name.on.host

db.name	Connection name in ODBC data source.
uid	Database User ID, if not in ODBC data source.
pwd	Database Password, if not in ODBC data source.
...	Additional arguments to odbcConnect.

Details

Creating a MySQL database typically requires three steps:
1. Design/test/export schema
2. Create a MySQL instance on the host (locally or on a server)
3. Import schema to create tables, keys, and relationships

The default acoustics database schema will allow the user to skip step 1; this function will take care of step 3. The user must ensure that a database instance exists and is present in the ODBC data source list before attempting to use this function. This function was tested using a schema automatically generated using the default "forward engineer" export function in MySQL Workbench with DROP statements.

Value

If tables, a list:

upload.time	Duration of upload and processing.
tables	Description tables in the acoustics database.

Otherwise a report of the duration of upload and processing time to indicate completion.

Author(s)

Jon Katz

Examples

```
## Not run:
dbSchema(
    schema = "acoustics.sql",
    name.on.host = "acoustics",
    tables = TRUE,
    schema.name = 'myschema',
    db.name = "acoustics",
    uid = "Admin",
    pwd = "BadPassword!" )

## $upload.time
## [1] "Upload time 10.977 secs"
##
## $tables
##    TABLE_CAT TABLE_SCHEM           TABLE_NAME TABLE_TYPE
```

```
## 1     JKATZ3                  tblAnnotations      TABLE
## 2     JKATZ3                      tblArchive      TABLE
## 3     JKATZ3                         tblCard      TABLE
## 4     JKATZ3                 tblCardRecorder      TABLE
## 5     JKATZ3                    tblCovariate      TABLE
## 6     JKATZ3             tblEnvironmentalData      TABLE
## 7     JKATZ3                      tblLocation      TABLE
## 8     JKATZ3                  tblOrganization      TABLE
## 9     JKATZ3                       tblPerson      TABLE
## 10    JKATZ3                tblPersonContact      TABLE
## 11    JKATZ3                      tblProject      TABLE
## 12    JKATZ3                     tblRecorder      TABLE
## 13    JKATZ3                       tblResult      TABLE
## 14    JKATZ3                tblResultSummary      TABLE
## 15    JKATZ3                      tblSpecies      TABLE
## 16    JKATZ3                tblSpeciesPriors      TABLE
## 17    JKATZ3                      tblSurvey      TABLE
## 18    JKATZ3                     tblTemplate      TABLE
## 19    JKATZ3                tblTemplatePrior      TABLE
##                                                      REMARKS
## 1                        For annotated song events in surveys.
## 2             For archiving sound clips extracted from surveys.
## 3             This table stores information about memory cards.
## 4                Track survey, recorder, and memory card links.
## 5  Describe covariates and types of enviromental data collected.
## 6                  Non-acoustic data: environmental covariates.
## 7   Information about about locations for surveys and templates.
## 8            Store the organization name and contact info here.
## 9                    Names of people in the monitoring program.
## 10           Contact info, including Cell/Work Phone and email.
## 11   Store the names of multiple projects per organization here.
## 12         This table stores information about recording units.
## 13                   Table to store the results of findPeaks().
## 14                        Store probability of survey presence.
## 15         Store BBL codes or other 4, 6, or 8 character codes.
## 16                   Store site & species specific priors here.
## 17        This table stores attributes of the survey recording.
## 18                       Store templates and template metadata.
## 19              Store beta parameter estimates for error rates.



## End(Not run)
```

---

dbUploadAnno *Upload Spectrogram Annotations to an Acoustics Database*

Description

Spectrogram annotations from viewSpec can be uploaded to tblAnnotations in an acoustics database. Annotations can be specified as either a file path to a csv document or as a data frame. The name

of the survey to associate with the annotations must be identical to tblSurvey.fldSurveyName to properly link the annotations to the survey.

Usage

```
dbUploadAnno(annotations, survey, db.name = "acoustics", uid,
             pwd, analyst = "", ...)
```

Arguments

annotations	Either a file path to a csv file or a data frame of annotations.
survey	Name of survey annotations belong to. Must match tblSurvey.fldSurveyName
db.name	Name of the ODBC connector data source corresponding to the acoustics database.
uid	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
pwd	Password to allow ODBC connector to connect to database, if not present in ODBC connector.
analyst	Numerical key value corresponding to the user's tblPerson.pkPersonID value in the acoustics database.
...	Additional arguments to RODBC::odbcConnect.

Details

dbUploadAnno assumes a database structure identical to that provided in the acoustics schema. If the username and password are present in the ODBC datasource they do not need to be provided. It is possible to store only the username in the datasource and enter a password, but the reverse will not work. Annotations are expected to be formatted by (or as if by) viewSpec, so if another piece of software is recording the annotations the field order must be altered to match output of viewSpec.

Value

Invoked for its side effect. Successful upload is marked by a report of the upload time; unsuccessful upload will report any errors encountered.

Note

The expected field order is c("start.time", "end.time", "min.frq", "max.frq", "name"). "name" is intentionally ambiguous; it may be used to store the species code, but it is not referenced back to tblSpecies.fldSpeciesCode for verification.

Author(s)

Jon Katz

See Also

viewSpec

Examples

```
# Assumes 'MABI01_2010-05-22_054400_0_000.wav' is a survey in tblSurvey.fldSurveyName
# Assumes 'MABI01_2010-05-22_054400.csv' is a file of annotations belonging to the above survey

## Not run:
#If using the 'acoustics' schema verbatim:
dbUploadAnno(
    annotations = "MABI01_2010-05-22_054400.csv",
    survey = "MABI01_2010-05-22_054400_0_000.wav",
    analyst = 1)

#'acoustics' schema, different database name:
dbUploadAnno(
    annotations = "MABI01_2010-05-22_054400.csv",
    survey = "MABI01_2010-05-22_054400_0_000.wav",
    db.name = "LocalSQLdb",
    uid = "EntryOnly",
    pwd = "07H23BBM",
    analyst = 1)
## End(Not run)
```

---

dbUploadResult *Upload Detection Results to an Acoustics Database*

Description

Upload detection results (peaks or detections) from `findPeaks` directly to tblResult in an acoustics database.

Usage

```
dbUploadResult(detection.obj, which.one, what = "detections", db.name = "acoustics",
               uid, pwd, analysis.type, analyst = "", ...)
```

Arguments

`detection.obj`	Object of class `detectionList` containing results from `findPeaks`.
`which.one`	Results from a single template can be selected for upload, or leave blank to upload results from all templates.
`what`	Character value of either "detections" (the default; peaks above the score cutoff) or "peaks" (all peaks regardless of score cutoff).
`db.name`	Name of the ODBC connector data source corresponding to the acoustics database.
`uid`	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
`pwd`	Password to allow ODBC connector to connect to database, if not present in ODBC connector.

analysis.type	Character value identifying analysis type, in c("BIN", "COR"). Some partial matching is performed.
analyst	Numerical key value corresponding to the user's tblPerson.pkPersonID value in the acoustics database.
...	Additional arguments to RODBC::odbcConnect.

Details

dbUploadResult assumes a database structure identical to that provided in the acoustics schema. If the username and password are present in the ODBC datasource they do not need to be provided. It is possible to store only the username in the datasource and enter a password, but the reverse will not work.

The value for analyst must be present in tblPeople.pkPeopleID for upload to succeed.

Value

Invoked for its side effect, which is to insert the detection results into tblResult in an acoustics database. Successful upload is marked by a report of the upload time; unsuccessful upload will report any errors encountered.

Author(s)

Jon Katz

See Also

findPeaks, getPeaks, getDetections

Examples

```
## Not run:
## Not run, as it requires a database to receive the upload
# Load data
data(btnw)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")
writeWave(btnw, btnw.fp)
writeWave(survey, survey.fp)

# Template construction
b4 <- makeBinTemplate(
    btnw.fp,
    frq.lim = c(2, 8),
    select = "auto",
    name = "b4",
    buffer = 4,
    amp.cutoff = -31,
```

```
    binary = TRUE)

# Binary point matching
scores <- binMatch(survey = survey.fp, templates = b4, time.source = 'fileinfo')

# Isolate peaks
pks <- findPeaks(scores)

#If using the 'acoustics' schema verbatim:
dbUploadResult(detection.obj = pks, analysis.type = "BIN", analyst = 1)

#'acoustics' schema, different database name:
dbUploadResult(
    detection.obj = pks,
    which.one = "b4",
    what = "peaks",
    db.name = "LocalSQLdb",
    uid = "EntryOnly" ,
    pwd = "07H23BBM",
    analysis.type = "BIN",
    analyst = 1)
## End(Not run)
```

---

dbUploadSurvey *Upload Survey Metadata to an Acoustics Database*

Description

Upload survey metadata to tblSurvey in an acoustics database.

Usage

```
dbUploadSurvey(db.name = "acoustics", uid, pwd, survey.meta, update.query = FALSE,
               tz, ...)
```

Arguments

survey.meta	Object containing survey metadata, typically gathered in one or more invocations of fileCopyRename.
db.name	Name of the ODBC connector data source corresponding to the acoustics database.
uid	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
pwd	Password to allow ODBC connector to connect to database, if not present in ODBC connector.
update.query	Logical value to control the type of query. See Details.

tz	Time zone, if not in file names or metadata. See Details.
...	Additional arguments to RODBC::odbcConnect.

Details

dbUploadSurvey assumes a database structure identical to that provided in the acoustics schema. If the username and password are present in the ODBC datasource they do not need to be provided. It is possible to store only the username in the datasource and enter a password, but the reverse will not work.

Surveys recorded as wav files have metadata read from the header of the file automatically; these data can be uploaded to the database in a single call to dbUploadSurvey. Metadata for surveys recorded in proprietary compressed file formats cannot be gathered in the same manner; some basic metadata is gleaned from the initial transfer of the surveys from memory-card to storage drive, and the rest is read after the conversion from proprietary format to wav file. If recording in a proprietary format, normal operation would thus call for two invocations of dbUploadSurvey: the first with partial metadata, and the second as an update query to fill in the missing values. Therefore, standard use (update.query = FALSE) passes a simple INSERT INTO query to the database and parses the fields appropriately. When update.query = TRUE, the assumption is made that many of the fields in survey.meta have already been entered into the database, but some remain NULL.

If no 'fldOriginalDateModified' exists in the metadata it will be automatically generated from the date coded in the file name during fileCopyRename.

Value

Invoked for its side effect, which is to insert the detection results into tblResult in an acoustics database. Successful upload is marked by a report of the upload time; unsuccessful upload will report any errors encountered.

Note

This is a convenience function for users who are unfamiliar with SQL syntax and/or have not established an alternative front-end for their acoustics database. Users capable of doing so may find more utility and flexibility writing custom queries directly either with an alternative front-end or RODBC::sqlQuery. No processing is performed; data is uploaded to the database as it exists in the metadata object.

Author(s)

Jon Katz

See Also

fileCopyRename, mp3Subsamp

Examples

```
## Not run:
# metadata for wav files:
metadata <- fileCopyRename(
```

```
        from = '~/media/SDcard',
        to = '~/Desktop/Acoustics/Recordings',
        csv.dir = '~/Desktop/Acoustics/Results',
        loc.prefix = 'MABI01',
        ext = 'wav',
        CardRecorderID = 1,
        kaleidoscope = FALSE)

# If using the 'acoustics' schema verbatim:
dbUploadSurvey(survey.meta = metadata)

# 'acoustics' schema, different database name:
dbUploadSurvey(
    survey.meta = metadata,
    db.name = "LocalSQLdb",
    uid = "EntryOnly",
    pwd = "07H23BBM")

# metadata for wac files:
metadata <- fileCopyRename(
        from = '~/media/SDcard',
        to = '~/Desktop/Acoustics/Recordings',
        csv.dir = '~/Desktop/Acoustics/Results',
        loc.prefix = 'MABI01',
        ext = 'wac',
        CardRecorderID = 1)

# If using the 'acoustics' schema verbatim:
dbUploadSurvey(survey.meta = metadata)

# 'acoustics' schema, different database name:
dbUploadSurvey(
    survey.meta = metadata,
    db.name = "LocalSQLdb",
    uid = "EntryOnly",
    pwd = "07H23BBM")

# After converting wac files to wav files use update.query = TRUE:
new.metadata <- fileCopyRename(
        from = '~/Desktop/Acoustics/Recordings',
        to = '~/Desktop/Acoustics/Surveys',
        csv.dir = '~/Desktop/Acoustics/Results',
        loc.prefix = 'MABI01',
        ext = 'wav',
        CardRecorderID = 1,
        metadata.only = TRUE)

# If using the 'acoustics' schema verbatim:
dbUploadSurvey(survey.meta = new.metadata, update.query = TRUE)

# 'acoustics' schema, different database name:
dbUploadSurvey(
    survey.meta = new.metadata,
```

```
    db.name = "LocalSQLdb",
    uid = "EntryOnly",
    pwd = "07H23BBM",
    update.query = TRUE)
## End(Not run)
```

---

dbUploadTemplate *Upload Acoustic Templates to a Database*

Description

Upload a binary point matching or correlation template list containing one or more templates to tblTemplate in an acoustics database. One or more templates may be indexed by name or position from the template list for upload.

Usage

```
dbUploadTemplate(templates, which.one, db.name = "acoustics", uid , pwd, analyst,
locationID = "", date.recorded = "", recording.equip = "", species.code,
type, ...)
```

Arguments

templates	TemplateList object of class binTemplateList or corTemplateList to upload.
which.one	Indexing option for individual templates within the TemplateList object. Indexing may be by name or numerical position. If missing, all templates within the list are uploaded.
db.name	Name of the ODBC connector data source corresponding to the acoustics database.
uid	User ID to allow ODBC connector to connect to database, if not present in ODBC connector.
pwd	Password to allow ODBC connector to connect to database, if not present in ODBC connector.
analyst	Numerical key value corresponding to the user's tblPerson.pkPersonID value in the acoustics database.
locationID	Numerical key value corresponding to the location's tblLocation.pkLocationID value in the acoustics database.
date.recorded	Dates template clip was recorded, in a recognizable POSIX format: YYYY/MM/DD.
recording.equip	Equipment used to record template clip.
species.code	Character value corresponding to the species' tblSpecies.fldSpeciesCode value in the acoustics database; usually a 4, 6, or 8-character code. Codes not in the database will return a cryptic error and cause upload to fail.
type	Character value identifying template type, in c("BIN", "COR"). Some partial matching is performed.
...	Additional arguments to RODBC::odbcConnect.

Details

dbUploadTemplate assumes a database structure identical to that provided in the acoustics schema. If the username and password are present in the ODBC datasource they do not need to be provided. It is possible to store only the username in the datasource and enter a password, but the reverse will not work.

The following must be true for upload to succeed: The value for analyst must be present in tblPeople.pkPeopleID The value for locationID must be present in tblLocation.pkLocationID the value for species.code must be present in tblSpecies.fldSpeciesCode

Value

This function is invoked for its side effect, which is to insert the template list into tblTemplate in an acoustics database. Successful upload is marked by a report of the upload time; unsuccessful upload will report any errors encountered.

Note

In the acoustics database templates are broken into components, and vectors are stored as text objects in various fields. Ultimately templates must be downloaded again to be used; to stay beneath the maximum download vector size of sqlQuery, extraneous characters are removed from each vector during upload. All amplitude values for correlation templates are sign-inverted and converted from floating point decimal to integers, and all decimals are rounded to the hundredth's place before upload; after upload all spaces, new-line, and carriage return characters are removed. Removal of these characters is usually the most time-consuming part of the upload process, and the console will report "cleaning up" while this is taking place. These measures sometimes inadequately trim character count, and users may find it necessary to increase the maximum download vector size in sqlQuery (see the vignette "MySQL_DataSources_RODBC" for further details). Large templates may take more than several seconds to upload; 2-5 seconds is normal for binary point matching templates, and 5-20 seconds is normal for correlation templates.

Author(s)

Jon Katz

See Also

dbDownloadTemplate

Examples

```
# Template construction
## Not run:
data(btnw)
b4 <- makeBinTemplate(
    "btnw.wav",
    frq.lim = c(2, 8),
    select = "auto",
    name = "b4",
    buffer = 4,
```

```
    amp.cutoff = -31,
    binary = TRUE)

\dontrun{
#If using the 'acoustics' schema verbatim:
dbUploadTemplate(
    templates = b4,
    analyst = 1,
    locationID = "MABI01",
    date.recorded = "2012/05/22",
    recording.equip = "SM2",
    species.code = "BTNW",
    type = "BIN")

#'acoustics' schema, different database name:
dbUploadTemplate(
    templates = b4,
    which.one = 1,
    db.name = "LocalSQLdb",
    uid = "EntryOnly",
    pwd = "07H23BBM",
    analyst = 1,
    locationID = "MABI01",
    date.recorded = "2012/05/22",
    recording.equip = "SM2",
    species.code = "BTNW",
    type = "BIN")}

## End(Not run)
```

`detectionList-class` *Class* `"detectionList"`

Description

These objects contain information on template detections, as well as (almost) all the information contained in `templateScores` These objects represent the final result of the template detection process. Various functions exist for working with these objects. Information on the detections alone can be extracted with `getDetections`.

Objects from the Class

Objects can be created by calls of the form `new("detectionList", ...)`. However, these objects should always be created by applying the `findPeaks` to `templateScores` objects. There are other functions the exist for modifying existing `detectionList` objects, including `showPeaks`, and the combination of `templateCutoff` and `findDetections`.

Slots

`survey.name`: Object of class `"character"`. The name of the survey file, or `"A Wave object"` if the survey was not read in from a file.

`survey`: Object of class `Wave`. The survey data, as a `"Wave"` object.

`survey.data`: Object of class `list`. A named list, with one element for each template. Each element contains data from a Fourier transform of the original survey: `amp` is a matrix of amplitudes (frequency by time, r by column), `t.bins` is a numeric vector with the values of the time bins (left-aligned–first bin is always 0.0), and `frq.bins` is a numeric vector with the values of the frequency bins (top-aligned–last bin is always the upper limit). There is a separate element for each template because each template may use different parameters for the Fourier transform (see `Template`).

`templates`: Object of class `list`. A named list of templates, which is identical to the original `TemplateList` used for template matching. This template list can be extracted with `getTemplates`.

`scores`: Object of class `list`. A named list, with one element for each template. Each element is a data frame with three columns: `date.time` is the absolute time of the score, `time` is the relative time of the score (relative to the survey start), and `score` is the score. Times are based on the center of the template, and so `time` will not correspond to values in `t.bins` in the `survey.data` above if the template spans an even number of time bins.

`peaks`: Object of class `list`. A named list, with one element for each template. Each element is a data frame that contains information on peaks that were found. The first three columns are identical to those in the `scores` data frames (above) (but of course only contain those values that were identified as peaks). The fourth column is logical and indicates whether the peak was also a detection.

`detections`: Object of class `list`. A named list, with one element for each template. Each element is a data frame that contains information on detections. The columns are identical to those in the `scores` data frames (above) (but of course only contain those values that were identified as detections (i.e., peaks with a score above the `score.cutoff`).

Methods

show `signature(object = "detectionList")`: ...

summary `signature(object = "detectionList")`: ...

Author(s)

Sasha D. Hafner

See Also

`findPeaks`, `getDetections`, `templateCutoff`, `templateScores`

Examples

```
showClass("detectionList")
```

eventEval *Evaluate Detected Events with Known Event Sources and Times*

Description

Evaluate whether the detected events are True +, True -, False +, or False - detections by comparing the results to a table of events with known sources and times (such as annotations from viewSpec). Events to evaluate may be either directly from an object of class detectionList, a csv file or data frame resulting from a call to getPeaks or getDetections, or a data frame downloaded from an acoustics database. A value for score.cutoff must be supplied to distinguish between True + and False -, even if assessing all peaks.

Usage

```
eventEval(detections, what = "detections", which.one, standard,
score.cutoff = 11, tol = 1)
```

Arguments

detections	An object of class detectionList, a csv file, or data frame containing detection results. See Details.
what	If a detectionList object is supplied for detections the character value of either "detections" (default; all peaks above the score cutoff) or all "peaks" may be selected.
which.one	If the detection process involved multiple templates only one may be selected for evaluation. Value can be either character (identifying the template name), or numerical (identifying the position in names(detections['template']). See Details.
standard	The "standard" is the results from annotation with viewSpec (i.e. Gold Standard) containing the source and time of each event. Can be a data frame or a file path to a csv file.
score.cutoff	If no template is supplied a score.cutoff can be supplied to evaluate false negatives.
tol	Numeric value for tolerance, with units seconds. If a detected event is within this value (actually +/- 0.5 x tol), the events are assumed to co-occur and be of the same origin.

Details

Little checking is performed to ensure that evaluation is possible based on the values for detections and standard. The standard must contain the fields c("start.time", "end.time", "min.frq", "max.frq", "name"). Objects are assumed to be from an acoustics database if they contain the fields c("fldTime", "fldScore", "fldTemplateName"). Data frames are assumed to be objects formerly of class detectionList if they contain the fields c("time", "score", "template").

Results from only one template from one survey may be evaluated in each call to eventEval.

Value

The detections data frame with an outcome field appended.

Note

`eventEval` performs the evaluation by merging the detections and standard data frames, ordering by time, and checking to see which rows occur within a value of `tol` to the row above. True + are defined as a detected event that co-occurrs in time with an event from the standard AND scores above or equal to the score.cutoff. Such an event that scores below the score.cutoff is classified as a False -. False - events may also be the product of an event from the standard failing to co-occur with any detected events. True - events don't co-occur with any standard events, and False + events similarly don't co-occur with standard events but score above or equal to the score.cutoff.

Author(s)

Jon Katz

See Also

The function `timeAlign` operates similarly, but rather than evaluate a set of detections against a standard it merges detections from multiple templates and retains only the co-occurring detections with the highest scores.

Examples

```
# Load data
data(btnw)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")
writeWave(btnw, btnw.fp)
writeWave(survey, survey.fp)

# Make a template
btemp <- makeBinTemplate(btnw.fp, frq.lim = c(2, 8), select = "auto", name = "btnw1", buffer =
                         4, amp.cutoff = -31, binary = TRUE)

# Binary point matching
scores <- binMatch(survey = survey.fp, templates = btemp, time.source = "fileinfo")

# Isolate peaks
pks <- findPeaks(scores)

# Evaluate peaks
data(survey_anno)

survey_anno <- survey_anno[survey_anno['name'] == 'BTNW', ]  # Extract the "BTNW" rows

peaks <- getPeaks(pks)
```

```
eval <- eventEval(detections = peaks, standard = survey_anno, score.cutoff = 15)
```

`extract-methods` *Indexing (Extraction) Methods for* **monitoR** *Package*

Description

These methods can be used to index detection list (`detectionList`), template lists (`TemplateList`), and template scores (`templateScores`) objects. Indexing is analogous to indexing a vector–with single square brackets, and character (template name) or integer (template position) values.

Methods

`signature(x = "detectionList")` Index by name or position of template(s).

`signature(x = "TemplateList")` Index by name or position of template(s).

`signature(x = "templateScores")` Index by name or position of template(s).

`fileCopyRename` *Copy and Rename Sound Files from Portable Media*

Description

Collects a variety of metadata about recordings that will be acoustic surveys and encodes the date modified into the file name. Copies files between directories to move them for an SD card to a hard disk, for example.

Usage

```
fileCopyRename(files, from = ".", to, csv.dir = to, csv.name, loc.prefix, ext,
rec.tz = NA, hours.offset = 0, CardRecorderID = NA, kaleidoscope = TRUE,
split.channels = FALSE, metadata.only = FALSE, full.survey.names = FALSE,
rename = TRUE, copy = TRUE)
```

Arguments

`files`	Optional vector of mp3, WAC, or WAV files to extract surveys from.
`from`	Directory containing mp3, WAC, or WAV recordings to extract survey from; required only if `files` is missing.
`to`	Directory where surveys will be placed after extraction.
`csv.dir`	Directory where csv file of survey metadata will be saved; defaults to the `to` directory.

csv.name	Name to save csv file of metadata, character value ending in .csv
loc.prefix	Character value identifying the location at which the recording was made. Will be used in the file name (see Details) and the csv file name. Must be in tblLocation.fldLocationName in the acoustics database.
ext	three-characters. The file extension defining the type of files to move, rename, and collect metadata on. Typically in c("wav", "wac")
rec.tz	Time zone for which the recordings were made (optional). Needed if different from the time zone setting of the operating system, when times will be adjusted to the 'correct' time zone. See details.
hours.offset	Hours to offset the modification time. Minimally useful when the recorder clock was set incorrectly. Use not at all, or if you must, with caution.
CardRecorderID	Numeric key value from tblCardRecorder.pkCardRecorderID, which links the recorder that made the recording with the location it was recorded.
kaleidoscope	Logical. If ext = "wac" files must be converted to .wav in Kaleidoscope. Setting to TRUE anticipates the renaming by Kaleidoscope.
split.channels	Logical. If ext = "wac" files must be converted to .wav in Kaleidoscope. Setting to TRUE anticipates further renaming by Kaleidoscope.
metadata.only	Logical. If ext = "wac" files must be converted to .wav before metadata can be collected; this argument typically is used in the second pass to collect the metadata.
full.survey.names	Logical. TRUE will use the full file path for the survey name in the resulting metadata table. In those cases the full path name will be stored in the database as well. Useful for coping with nested or disparate survey directories.
rename	Logical. FALSE will disable renaming.
copy	Logical. FALSE will disable file copying.

Details

The file name is where two important pieces of metadata are encoded: the location (as the location prefix) and the date and time of recording (as the date modified of the original file). The detection functions corMatch binMatch are capable of using this data as a time reference. Time zone management is tricky; if recordings were made in a different time zone than the OS running fileCopyRename, specify the correct time zone for the recordings with the rec.tz argument. Unexpected results are possible, as time zone abbreviations in general use may not match those in the Internet Assigned Numbers Authority tz database. The most reliable way to specify time zone is to use the full name, most quickly seen using OlsonNames, and also found on wikipedia: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones. Metadata cannot be read for non-wave recordings, so typically a first function call is used to encode the location prefix and date modified into the file name and move it from the portable media, and a second function call with metadata.ony = TRUE is used after conversion to wave format to fill in the missing metadata. The full.survey.names argument is designed to permit the batch processing of sound files saved in different directories.

Value

A data frame of metadata about the surveys. Contains column names "fldOriginalDateModified", "fldOriginalRecordingName", "fldSurveyName", "fldRecordingFormat", "fkCardRecorderID", "fldSurveyLength", "fldSampleRate", "fldBitsperSample", and "fldChannels". Column names reflect the assumption that this data will become a catalog of surveys stored in the database.

Author(s)

Jon Katz

References

Time zone conversion assisted by a post on David Smith's Revolutions blog, June 02, 2009: https://blog.revolutionanalytics.co
time-zones.html

See Also

mp3Subsamp

Examples

```
## Not run:
# Not run because it will create a file in user's working directory
data(survey)

writeWave(survey, "survey.wav")

meta <- fileCopyRename(
            files = "survey.wav",
            to = getwd(),
            csv.name = "sampleMeta.csv",
            loc.prefix = "MABI06",
            ext = "wav",
            CardRecorderID = 1)

# If your recorder's clock is set to GMT but your OS is not:
altmeta <- fileCopyRename(
               files = "survey.wav",
               to = getwd(),
               csv.name = "sampleMeta.csv",
               loc.prefix = "MABI06",
               ext = "wav",
               rec.tz = "GMT",
               CardRecorderID = 1)

file.remove("survey.wave")
## End(Not run)
```

`findPeaks` *Find Score Peaks and Detections in a* `templateScores` *Object*

Description

This function accepts `templateScores` objects and returns information on all score peaks and those peaks that are considered detections.

Usage

```
findPeaks(score.obj, fd.rat = 1, frame, parallel = FALSE)
```

Arguments

`score.obj`	A `templateScores` object, produced by `corMatch` or `binMatch`.
`fd.rat`	A ratio of frame width (twice minimum peak separation) to template duration.
`frame`	If you want the same frame width for templates with varying duration, specify a value directly. `fd.rate` will be ignored if `frame` is specified.
`parallel`	Set to `TRUE` for parallel processing using `mclapply`. This option is not available for Windows operating systems.

Details

The `findPeaks` function translates raw scores from template matching to detection information, by finding peaks in the score data, and determining which peaks, if any, exceed the score cutoffs specified in the templates (see the two functions for making templates, `makeBinTemplate` and `makeCorTemplate` and `templateCutoff` for more details on cutoffs).

Value

An S4 object of class `templateScores`, with the following slots:

`survey.name`	The file path to the survey that the scores apply to.
`survey`	The actual survey as a `Wave` object.
`survey.data`	A named list with one element per template. Each element is a named list with time-domain results for the survey.
`templates`	The templates (an S4 object of class `corTemplateList` or `binTemplateList`) used to calculate the scores.
`scores`	A named list with an element for each template. Each element contains the scores for an individual template.
`peaks`	A named list with peak information (as a data frame) for each template.
`detections`	A named list with detection information (as a data frame) for each template.

Author(s)

Sasha D. Hafner and Jon Katz

See Also

makeCorTemplate, makeBinTemplate, corMatch, binMatch, getDetections, getPeaks

Examples

```
# Load data
data(btnw)
data(oven)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)
writeWave(survey, survey.fp)

# Correlation example
# Create two correlation templates
wct <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w")

oct <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o")

# Combine them
ctemps <- combineCorTemplates(wct, oct)

# Calculate scores
cscores <- corMatch(survey.fp, ctemps)

# Finally, find peaks and detections
cdetects <- findPeaks(cscores)

cdetects

plot(cdetects)

# plotting help:
method?plot('detectionList')

# Binary example
## Not run:
# Not run because of the time required (maybe 2-5 seconds) Create two templates
wbt <- makeBinTemplate(btnw.fp, amp.cutoff = -30, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6),
                       buffer = 2, name = "w")

obt <- makeBinTemplate(oven.fp, amp.cutoff = -20, t.lim = c(1, 4), frq.lim = c(1, 11),
                       name = "o")
```

```
# Combine them
btemps <- combineBinTemplates(wbt, obt)

# Calculate scores
bscores <- binMatch(survey.fp, btemps)

# Finally, find peaks and detections
bdetects <- findPeaks(bscores)

bdetects

plot(bdetects)

## End(Not run)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
file.remove(survey.fp)
```

`getDetections` *Extract Detections or Peaks from a* `detectionList` *Object*

Description

These functions return detection and peak timing and scores from a `detectionList` object for one or more templates used to create the object.

Usage

```
getDetections(detection.obj, which.one = names(detection.obj@detections), id = NULL,
              output = "data frame")

getPeaks(detection.obj, which.one = names(detection.obj@detections), id = NULL,
         output = "data frame")
```

Arguments

`detection.obj`	The `detectionList` object.
`which.one`	The name(s) of the template(s) for which results should be returned. Character vector.
`id`	Additional information that will be added as an additional column in the returned data frame(s). By default, no column is added. Length-one vector.
`output`	Type of output, can be `"data frame"` or `"list"`. List output contains a single element (a data frame) for each template.

Details

The `id` argument is for adding an identifying "tag" to the output. This could be useful when, e.g., extracting detections for multiple surveys and then combining all results into a single data frame.

Value

A data frame with up to six (seven for `getPeaks`) columns: `id` (from the `id` argument) (optional), template name (`template`), date and time (`date.time`, relative `time` (relative to the recording start), `score`, and verification results (`true`) (only present if the `detectionList` contains verification results from `showPeaks`). Or, a list with a separate data frame for each template. For `getPeaks`, there is also a `detection` column, with `TRUE` when a peak has been identified as a detection.

Author(s)

Sasha D. Hafner

See Also

`findPeaks`

Examples

```
# Load data
data(btnw)
data(oven)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)
writeWave(survey, survey.fp)

# Correlation example
# Create two correlation templates
wct <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w")
oct <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o")

# Combine both of them
ctemps <- combineCorTemplates(wct, oct)

# Calculate scores
cscores <- corMatch(survey.fp, ctemps)

# Find peaks
cdetects <- findPeaks(cscores)

# Finally, get detections
getDetections(cdetects)
```

```
# If list is preferred
getDetections(cdetects, output = "list")

# For select templates
getDetections(cdetects, which.one = 1)
getDetections(cdetects, which.one = "w")

# Or for all peaks
getPeaks(cdetects)
getPeaks(cdetects, output = "list")
getPeaks(cdetects, which.one = 1)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
file.remove(survey.fp)
```

`getTemplates` *Extract a Template List*

Description

Use this function to extract template lists from `templateScores` or `detectionList` objects.

Usage

```
getTemplates(object, which.ones = names(object@templates))
```

Arguments

`object`	The `templateScores` or `detectionList` object that contains the templates that are to be extracted.
`which.ones`	Which templates should be included? A character vector of templates names, or an integer vector. Default is all templates.

Details

This function would typically be used to extract and save a complete set of templates from a `detectionList` object if `templateCutoff` has been used to modify the template list after scores were calculated. `getTemplates` could also be used to extract a subset of templates present in a template list, but indexing with square brackets is an easier approach.

Value

A template list of class `corTemplateList` or `binTemplateList`.

Author(s)

Sasha D. Hafner

See Also

makeCorTemplate, makeBinTemplate, templateCutoff, templateComment

---

makeTemplate *Make an Acoustic Template*

Description

Functions for creating a spectrogram cross-correlation template or a binary point matching template for later use in identification of acoustic signals. A template is made by manually or automatically selecting cells within a Fourier-transformed representation (a spectrogram) of an audio recording.

Usage

```
makeCorTemplate(clip, t.lim = NA, frq.lim = c(0, 12), select = "auto", dens = 1,
                score.cutoff = 0.4, name = "A", comment = "", spec.col = gray.3(),
                  sel.col = ifelse(dens == 1, "#99009975", "orange"),
                  wl = 512, ovlp = 0, wn = "hanning", write.wav = FALSE, ...)

makeBinTemplate(clip, t.lim = NA, frq.lim = c(0, 12), select = "auto", binary = TRUE,
                buffer = 0, dens = 1, score.cutoff = 12, name = "A", comment = "",
              amp.cutoff = "i", shift = "i", high.pass = -Inf, spec.col = gray.3(),
                  bin.col = c("white", "black"),
                  quat.col = c("white", "gray40", "gray75", "black"),
                  sel.col = c("orange", "blue"), legend.bg.col = "#2E2E2E94",
                  legend.text.col = "black", wl = 512, ovlp = 0, wn = "hanning",
                  write.wav = FALSE, ...)
```

Arguments

clip	A file path to one wav or mp3 file, or a Wave object (but see 'Details' for this case). Or, for makeBinTemplate only, a list or vector of two such objects. Character vector or list.
t.lim	Time limits of the spectrogram plot or template itself, or a list of exactly two such vectors. Length two numeric vector.
frq.lim	Frequency limits of spectrogram plot or template. Length two numeric vector.
select	How should points be selected? Options are "cell", "rectangle", "auto". Length one character vector.
binary	Should plot be binary? Length one logical vector.
buffer	The size of a buffer (in number of time by frequency bins) around "on" points for select = "rectangle" and select = "auto" for makeBinTemplate. Bins within the buffer will not be included as "on" or "off" points. Length one integer vector.
dens	Approximate density of points included with select = "rectangle" and select = "auto" as a fraction of 1.0. Length one numeric vector.

score.cutoff	The numeric value set for the score.cutoff element of the resulting template. This value will determine which peaks qualify as detections when the resulting template is used in a complete detection analysis. Length one numeric vector.
name	The name of the template, which will be associated with the template. To change the name of an existing template, see templatenames. Length one character vevetor.
comment	Comment that will be saved with the template. See templateComment.
amp.cutoff	Amplitude cutoff for creating a binary plot. Length one numeric vector or else "i" for interactive selection.
shift	When two clips are used, the forward shift for the second clip, in time bins. Length one integer vector, or "i" for interactive.
high.pass	High-pass filter value. All amplitudes below this frequency will be set to the minimum.
spec.col	A color palette function for the spectrogram when binary = FALSE.
bin.col	Colors for the spectrogram when binary = TRUE. Length two character vector: bin.col[1] for cells below the cutoff, bin.col[2] for cells above the cutoff.
quat.col	Colors for the spectrogram when using two clips. Length four character vector: bin.col[1] for cells below the cutoff for both clips, bin.col[2] for cells above the cutoff for clip 1 only, bin.col[3] for cells above the cutoff for clip 2 only, bin.col[4] for cells above the cutoff for both clips.
sel.col	The color for displaying selected cells.
legend.bg.col	The color of the legend background.
legend.text.col	Legend text color.
wl	The wl argument sent to the spectro function.
ovlp	The ovlp argument sent to the spectro function.
wn	The wn argument sent to the spectro function.
write.wav	If clip is a Wave object, should it be written to file in the working directory? If FALSE, it will instead be written to a temporary file. See details.
...	Additional arguments to spectro.

Details

makeCorTemplate is used for making correlation templates, while makeBinTemplate is used to make binary point matching templates. makeBinTemplate can be used with one or two recordings (clip argument). If the clip argument is a Wave object, the functions will write the object(s) to a wav file(s), in the working directory if the write.wav argument is TRUE, otherwise as a temporary file. This behavior extends from an early intent to link original recordings with templates while keeping the templates small. To use templates produced with these functions, see corMatch or binMatch. To combine template lists, see combineCorTemplates or combineBinTemplates.

Value

An S4 object of class corTemplateList (returned by makeCorTemplate) or binTemplateList (returned by makeBinTemplate).

Author(s)

Sasha D. Hafner and Jon Katz

References

Mellinger, DK, Clark, CW. 1997. Methods for automatic detection of mysticete sounds. *Marine and Freshwater Behaviour and Physiology* **29**, 163-181.

Towsey M, Planitz, B, Nantes, A, Wimmer, J, Roe, P. 2012. A toolbox for animal call recognition. *Bioacoustics* **21**, 107-125.

See Also

corMatch, binMatch, templateNames, templateCutoff

Examples

```
# Load example Wave objects
data(btnw)
data(oven)

# Use a Wave object directly to make a template
## Not run:
# Not run because it will create a file in user's working directory with write.wav = TRUE
wct1 <- makeCorTemplate(btnw, name = "w1", write.wav = TRUE)
wct1

## End(Not run)

# For traceability, better to use acoustic files
# Here, first write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Use default arguments except for name
wct1 <- makeCorTemplate(btnw.fp, name = "w1")

# Specify time and frequency limits to focus on a smaller area
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")

# For finer control, see options for select argument, e.g.,
## Not run:
# Not run because requires user interaction
wct3 <- makeCorTemplate(btnw.fp, select = "cell", name = "w3")
wct4 <- makeCorTemplate(btnw.fp, select = "rectangle", name = "w4")

## End(Not run)

# Use a different recording--different species here
oct1 <- makeCorTemplate(oven.fp, name = "o1", t.lim = c(1, 4), frq.lim = c(1, 11))
```

```
# Reduce cell density
oct2 <- makeCorTemplate(oven.fp, name = "o2", t.lim = c(1, 4), frq.lim = c(1, 11),
                        dens = 0.1)

# Binary templates are similar
# By default, amplitude cutoff is interactively set
## Not run:
wbt1 <- makeBinTemplate(btnw.fp, name = "w1")

## End(Not run)

# Or specify cutoff directly
wbt1 <- makeBinTemplate(btnw.fp, amp.cutoff = -40, name = "w1")

# Specify time and frequency limits to focus on a smaller area in spectrogram, and add a
# buffer
## Not run:
wbt2 <- makeBinTemplate(btnw.fp, amp.cutoff = -30, t.lim = c(1.5, 2.1),
                        frq.lim = c(4.2, 5.6), buffer = 2, name = "w2")

## End(Not run)

# For finer control, see options for select argument, e.g.,
## Not run:
# Not run because it requires user input to select cells for the template
wbt3 <- makeBinTemplate(btnw.fp, amp.cutoff = -40, t.lim = c(0.5, 2.5),
                        frq.lim = c(1, 11), select = "cell", name = "w3")

wbt4 <- makeBinTemplate(btnw.fp, amp.cutoff = -40, t.lim = c(0.5, 2.5),
                       frq.lim = c(1, 11), select = "rectangle", buffer = 3, name = "w4")

## End(Not run)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)

# TemplateList plotting help:
method?plot('TemplateList')
```

---

monitoR *Automated Acoustic Monitoring–overview and examples*

Description

monitoR contains functions for template matching, template construction, spectrogram viewing and annotation, and direct MySQL database connectivity. This package offers two fully-supported template matching algorithms: binary point matching and spectrogram cross-correlation. The direct database connection facilitates efficient data management when batch processing as well as template

storage and sharing. It supplies a database schema that is useful for managing recorders in the field as well as functions for reading metadata from sound files when they are copied from external media.

Details

For an introduction to the package see the vignette. For some introductory examples, see 'Examples' below.

Acknowledgments

A Fourier transformed is used in the **monitoR** package to transform time-domain acoustic data to frequency-domain data (i.e., the data displayed in the spectrograms used to produce templates). The `spectro` function used in our package is a pared-down version of a function of the same name in Jerome Sueur's excellent package **seewave**. To use `spectro`, the **seewave** functions `dBweight`, `ftwindow`, `hamming.w` and other window functions, and `stft` are from **seewave**. The function `readMP3` is modified from Uwe Ligges' package **tuneR**. And several other **tuneR** functions are used directly from the **tuneR** package. Without **seewave** and **tuneR** this project would have gotten off to a much slower start.

Generous funding for this work was provided by the National Park Service, the U.S. Geological Survey, and the National Phenology Network.

Disclaimer

"Although this software program has been used by the U.S. Geological Survey (USGS), no warranty, expressed or implied, is made by the USGS or the U.S. Government as to the accuracy and functioning of the program and related program material nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the USGS in connection therewith."

Functions in monitoR

Create a MySQL database (`dbSchema`), to which survey metadata, templates and metadata, and results can be sent. Copy sound files from external media (`fileCopyRename`) and upload the metadata to the database (`dbUploadSurvey`). View and interactively annotate sound files of any length (`viewSpec`). Download a table of surveys from the database (`dbDownloadSurvey`), construct a template (`makeBinTemplate` or `makeCorTemplate`), detect/score events in a survey (`binMatch`, `corMatch`), apply a threshold to the scores (`findPeaks`), send the results to the database (`dbUploadResult`).

Author(s)

Sasha D. Hafner <sdh11@cornell.edu> and Jon Katz <jonkatz4@gmail.com>, with code for the Fourier transform from the seewave package (by Jerome Sueur, Thierry Aubin, and Caroline Simonis), and code for the readMP3 function from the tuneR package (by Uwe Ligges).

Maintainer: Sasha D. Hafner <sdh11@cornell.edu>

References

Ligges, Uwe. 2011. tuneR: Analysis of music. https://r-forge.r-project.org/projects/tuner/

Sueur J, Aubin, T, Simonis, C. 2008. Seewave: a free modular tool for sound analysis and synthesis. *Bioacoustics* **18**, 213-226.

Towsey M, Planitz, B, Nantes, A, Wimmer, J, Roe, P. 2012. A toolbox for animal call recognition. *Bioacoustics* **21**, 107-125.

Examples

```
# View spectrograms
data(survey)
viewSpec(survey)

# Annotate features
## Not run:
# Not run because it is interactive and a file is written to user's working directory
viewSpec(survey, annotate = TRUE)

# View previous annotations
data(survey_anno)

write.csv(survey_anno, "survey_anno.csv", row.names = FALSE)

viewSpec(survey, annotate = TRUE, anno = "survey_anno.csv", start.time = 5)

## End(Not run)

# Load example Wave object
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")

writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)
writeWave(survey, survey.fp)

# Correlation example
# Create two correlation templates
wct <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w")

oct <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o")

# Combine them
ctemps <- combineCorTemplates(wct, oct)

# Calculate scores
cscores <- corMatch(survey.fp, ctemps)

# Find peaks and detections
cdetects <- findPeaks(cscores)
```

```
## Not run:
# Not run because it takes a second to draw the plot
# View results
plot(cdetects, hit.marker = "points")

# Interactively inspect individual detections
# Not run because it is interactive
cdetects <- showPeaks(cdetects, which.one = "w1", flim = c(2, 8), point = TRUE,
                      scorelim = c(0, 1), verify = TRUE)

## End(Not run)
```

`mp3Subsamp` *Extract Short Surveys from Longer mp3 Recordings*

Description

Extract short surveys from longer mp3 recordings without decoding and re-encoding. Collects metadata about surveys for upload to an acoustic database and renames files with original date modified. Timing options are one or more surveys per hour starting at the beginning time of the recording or one survey per hour starting on each hour.

Usage

```
mp3Subsamp(files, from = ".", to, csv.dir = to, csv.name, duration = 600,
           mins.between = 50, index = "hour", loc.prefix, CardRecorderID = NA,
           kbps = 128, samp.rate = 44100, channels = 2, split = TRUE)
```

Arguments

`files`	Optional vector of mp3 file paths to extract surveys from.
`from`	Directory containing mp3 recordings to extract survey from; required only if `files` is missing.
`to`	Directory where surveys will be placed after extraction.
`csv.dir`	Directory where csv file of survey metadata will be saved; defaults to the `to` directory.
`csv.name`	Name assigned to csv file of metadata (character value ending in .csv).
`duration`	Duration of surveys to extract (numeric, units = 'seconds'). Defaults to 600 seconds (10 minutes).
`mins.between`	Number of minutes to skip between surveys (numeric). If `index = "hour"`, the value for `mins.between + duration * 60` (duration converted to minutes) equals the repeat period. Defaults to 50 minutes, for a 60 minute repeat period.
`index`	Character value indicating whether to take the first survey at the next hour in the recording (identified based on file date modified) or simply from the start of the recording. In `c("hour", "time0")`. Defaults to `"hour"`.

`loc.prefix`	Six characters identifying the location at which the recording was made. Will be used in the file name (see Details) and the csv file name. Must be in *tblLocation.fldLocationName* in the acoustics database.
`CardRecorderID`	Numeric key value from *tblCardRecorder.pkCardRecorderID*, which links the recorder that made the recording with the location it was recorded.
`kbps`	Numeric value for mp3 bitrate. Common values are `c(64, 128, 160, 192, 224, 256, 320)`. Must match the bitrate set by the recording device.
`samp.rate`	Numeric value for mp3 sample rate. Common values are `c(22050, 44100, 48000)`. Must match the sample rate set by the recording device.
`channels`	Numeric value for number of audio channels in mp3 file. Both "Stereo" and "Joint Stereo" are 2-channel recordings. "Mono" is a 1-channel recording.
`split`	Logical. The default `TRUE` will send the call to mp3splt to subsample the surveys; `FALSE` will generate metadata only.

Details

This function calls mp3splt, a third party library that must be installed separately from https://mp3splt.sourceforge.net/. This function supplants `fileCopyRename` as a file copying function and a metadata collection tool when using the acoustic database.

The survey file names produced will be of the form PREFIX_YYYY-mm-dd_HHMSS.mp3. Surveys from the same location can be linked by the location prefix and differentiated by different modification dates.

Value

Data frame with metadata about the surveys. Metadata includes: the date modified (fldOriginalDateModified), the original recording name (fldOriginalRecordingName), the new survey name (fldSurveyName), the recording format (fldRecordingFormat), the value for pkCardrecorderID (fkCardRecorderID), the duration of each survey (fldSurveyLength), the sample rate (fldSampleRate), the bit depth (fldBitsperSample), and the number of channels (fldChannels).

Note

`dbUploadSurvey` assumes a database structure identical to that provided in the acoustics schema.

Author(s)

Jon Katz

See Also

See `fileCopyRename` to move wave files and prepare metadata for the database; `dbUploadSurvey` to upload the survey metadata to the acoustics database.

Examples

```
# Specify individual files, 10 minutes every hour from the file start:
## Not run: metadata <- mp3Subsamp(files = '~/media/SDcard/MA01.mp3', to = '~/Desktop/Acoustics/Recordings',
csv.dir = '~/Desktop/Acoustics/Results', index = "time0", loc.prefix = 'MABI01', CardRecorderID = 1
## End(Not run)

# 10 minute surveys at the top of every hour, from an entire SD card:
## Not run: metadata <- mp3Subsamp(from = '~/media/SDcard', to = '~/Desktop/Acoustics/Recordings',
csv.dir = '~/Desktop/Acoustics/Results', loc.prefix = 'MABI01', CardRecorderID = 1
## End(Not run)

# 5 minute surveys every 30 minutes starting at the top of every hour, from an entire SD card:
## Not run: metadata <- mp3Subsamp(from = '~/media/SDcard', to = '~/Desktop/Acoustics/Recordings',
csv.dir = '~/Desktop/Acoustics/Results', duration = 300, mins.between = 25, loc.prefix = 'MABI01',
CardRecorderID = 1
## End(Not run)
```

---

oven *Ovenbird (*Seiurus aurocapilla*) Song*

Description

A 3 second wave recording of an Ovenbird (*Seiurus aurocapilla*) song.

Usage

```
data(oven)
```

Format

The format is:
Formal class 'Wave' [package "tuneR"] with 6 slots ..@ left : int [1:120001] 84 170 281 142 129 55 120 181 126 178 ... ..@ right : num(0) ..@ stereo : logi FALSE ..@ samp.rate: int 24000 ..@ bit : int 16 ..@ pcm : logi TRUE

Source

Sound clips were recorded in Vermont, USA in 2010. Equipment was a Wildlife Acoustics SM1(TM) recorder recording in WAC0 format, converted to wave using the Wildlife Acoustics Wac2Wav (TM) converter. Recording has a sample rate of 24kHz and is 16-bit mono.

Examples

```
data(oven)
viewSpec(oven)
```

plot-methods *Methods for the* plot *Function*

Description

Plotting acoustic templates and template scores

Usage

```
## S4 method for signature 'TemplateList,ANY'
plot(x, which.one = names(x@templates), click = FALSE,
ask = if(length(which.one)>1) TRUE else FALSE, spec.col = gray.3(), on.col = '#FFA50075',
off.col = '#0000FF75', pt.col = '#FFA50075', line.col = 'black')

## S4 method for signature 'detectionList,ANY'
plot(x, flim = c(0, 12), scorelim,
which.one = names(x@templates), box = TRUE, spec.col = gray.2(), t.each = 30,
hit.marker = 'lines',
color = c('red', 'blue', 'green', 'orange', 'purple', 'pink', 'darkgreen', 'turquoise',
'royalblue', 'orchid4', 'brown', 'salmon2'), legend = TRUE, all.peaks = FALSE,
ask = if(dev.list() == 2) TRUE else FALSE)
```

Arguments

x	A template list (TemplateList object) or detection list (detectionList object).
which.one	Names of templates to be plotted.
click	Set to TRUE to see values of locations on plot by mouse clicks.
ask	Set to FALSE to eliminate pause between plots.
spec.col	Color ramp for spectrogram.
on.col	Color for "on" points (binary templates only).
off.col	Color for "off" points (binary templates only).
pt.col	Color for template points (correlation templates only).
line.col	Color for lines if click = TRUE.
flim	Frequency limits for plot.
scorelim	Score limits for plot.
box	If TRUE boxes are plotted in spectrogram for each detection.
t.each	Duration shown in each individual plot (s).
hit.marker	Type of marker used to show detections in score plot. Can be "lines" or "points".
color	Colors used for individual templates.
legend	Show legend?
all.peaks	Indicate location of all peaks?

Author(s)

Sasha D. Hafner

See Also

makeCorTemplate, makeBinTemplate

Examples

```
## Not run:
# Not run because of the time required (maybe 5-10 seconds)
# Also some plot calls require user input by default
# Load data
data(btnw)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")
writeWave(btnw, btnw.fp)
writeWave(survey, survey.fp)

# Create a template list
ctemp1 <- makeCorTemplate(btnw.fp, name = "w1")
ctemp2 <- makeCorTemplate(btnw.fp, t.lim = c(0.5, 2.5), frq.lim = c(1, 10), dens = 0.1, name = "w2")
ctemps <- combineCorTemplates(ctemp1, ctemp2)

# Then it can be plotted like this
plot(ctemps)

# Next call is not useful for template w1 but good for w2:
plot(ctemps, pt.col = "red")

# Can plot just one template
plot(ctemps, which.one = 2, pt.col = "red")
plot(ctemps, which.one = "w2", pt.col = "red")

# And to check values
plot(ctemps, which.one = 1, click = TRUE)

# To plot detections, let's create some
cscores <- corMatch(survey.fp, ctemps)
cdetects <- findPeaks(cscores)

# And to plot them:
plot(cdetects)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(survey.fp)
```

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

`readMP3` *Read MP3 Files into a Wave Object*

Description

A variation of the MP3 file reader supplied in `tuneR`. Reads MP3 files in as 16bit PCM data stored in a Wave object.

Usage

```
readMP3(filename, from, to)
```

Arguments

`filename`	Filename of MP3 file.
`from`	Seconds to begin reading, measured from beginning of file. See details.
`to`	Seconds to end reading, measured from beginning of file. See details.

Details

The bare bones MP3 file reader supplied in `tuneR` reads the entire file in. When the user installs the third party software mp3splt and libmp3splt, this variant will allow `from` and `to` to be specified, and mp3splt will attempt to read in the MP3 segment without first decoding the file. Because mp3splt will cut the MP3 file at frame boundaries the `from` and `to` arguments are necessarily only guiding values; actual values may differ.

Value

An object of class `Wave`.

Note

If mp3splt is not installed a prompt will suggest falling back on the version from tuneR.

Author(s)

Jon Katz

References

mp3splt is documented at http://mp3splt.sourceforge.net/mp3splt_page/home.php.

See Also

`readMP3`, `readWave`

Examples

```
## Not run:
# Assume myMP3 is an MP3 file with a duration of at least 60 seconds:
readMP3 (filename = "myMP3.mp3", from = "30", to = "60")
## End(Not run)
```

---

readTemplates *Read Acoustic Templates from a Local Disk*

Description

Read single templates stored on a local disk, or read in entire directories of templates.

Usage

```
readBinTemplates(files = NULL, dir = ".", ext = "bt", parallel = FALSE)
readCorTemplates(files = NULL, dir = ".", ext = "ct", parallel = FALSE)
```

Arguments

`files`	Optional named vector of file names. See details.
`dir`	Name of directory to read files from. Default is working directory.
`ext`	Extension of files that should be read in. Files in `dir` without this extension will be skipped. Not necessary if `files` is provided.
`parallel`	Logical. `TRUE` uses mclapply, otherwise lapply is used.

Details

These functions can be used in three different ways, in both cases combing all templates read in into a single template list. By specifying a character vector of file names for `files`, they will read in the named files, and assign names based on file names. If `files` is a named vector, the vector names will be used in the resulting template list. Finally, if `files` is not provided, the functions will read in all saved templates with the extension `ext`.

Value

An object of class `TemplateList` containing either binary point templates or spectrogram cross-correlation templates.

Author(s)

Sasha D. Hafner

See Also

`writeBinTemplates`, `writeCorTemplates`

Examples

```
# Load data
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Correlation example
# Create one correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")
oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")
oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o2")

# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)

## Not run:
# Write ctemps to a directory "templates"
writeCorTemplates(ctemps, dir = "templates")

# Read in all correlation templates in a directory "templates"
ctemps <- readCorTemplates(dir = "templates")

# Read in two specific files
ctemps <- readCorTemplates(files = c("o1.ct", "o2.ct"), dir = "templates")

# Read in two specific files, and give them names
ctemps <- readCorTemplates(files = c(oven1 = "o1.ct", oven2 = "o2.ct"), dir = "templates")

## End(Not run)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
```

---

show-methods *Methods for the* show *and* summary *Functions*

Description

These methods are used for viewing template lists and other objects. For all types of objects documented here, show and summary will produce identical results.

Methods

`signature(object = "binTemplateList")` Displays a summary of `binTemplateList` objects.

`signature(object = "corTemplateList")` Displays a summary of `corTemplateList` objects.

`signature(object = "TemplateList")` Displays a summary of `TemplateList` objects.

`signature(object = "detectionList")` Displays a summary of `detectionList` objects.

`signature(object = "templateScores")` Displays a summary of `templateScores` objects.

Author(s)

Sasha D. Hafner

See Also

`makeCorTemplate`, `makeBinTemplate`

Examples

```
# Load data
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Correlation example
# Create two correlation templates
wct <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w")
oct <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o")

# Combine them
ctemps <- combineCorTemplates(wct, oct)

# Then for a quick summary:
ctemps

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
```

showPeaks *View or Verify Detections or Peaks*

Description

Use this function to view a spectrogram and score plot of detections or peaks. In its simplest usage, showPeaks will show all detections within for the first template within the detection list object, one after the other. With the verify option (verify = TRUE), the user can tag detections or peaks as TRUE or FALSE, and these results will be saved in an updated detection list object.

Usage

```
showPeaks(detection.obj, which.one = names(detection.obj@templates)[1], fd.rat = 4,
frame = fd.rat * detection.obj@templates[[which.one]]@duration, id = 1:nrow(pks),
t.lim, flim = c(0, 20), point = TRUE, ask = if (verify) FALSE else TRUE,
scorelim = NULL, verify = FALSE, what = "detections", box = TRUE,
player = "play", spec.col = gray.3(), on.col = '#FFA50075', off.col = '#0000FF75',
pt.col = '#FFA50075')
```

Arguments

detection.obj	A detection list object (detectionList).
which.one	Which template should be shown? Identify by name or position. Length-one integer or character vector.
fd.rat	Ratio of plot frame (time duration of plots) to template duration.
frame	Or, specify the plot frame (x limits of plots) instead of fr.rate. Length-one numeric vector.
id	Use to specify which peaks or detections will be shown. Integer vector.
t.lim	Or, to view only those detections or peaks within a certain time range, specify it here. Length-two numeric vector.
flim	Frequency limits (y axis limits) for the spectrogram. Length-two numeric vector.
point	If TRUE, plot points to show detection or peak locations.
ask	The setting of the par setting ask. Default value is based on other arguments, and should suffice in most cases.
scorelim	Score limits (y axis limits) for the score plot.
verify	If TRUE, R will prompt user to identify whether detections are TRUE
what	Should all peaks (what = "peaks") or just detections (what = "detections") be shown?
box	If TRUE plot a box around detections in the spectrogram. Box boundaries are based on template duration and frequency limits. Can also be set to "template" to see the template points plotted over the detection.
player	If verify = TRUE, the user will have the option to play the detection or peak. This argument is the command used for starting the player. See Details.

spec.col	A vector of colors for the spectrogram.
on.col	Colors for the on points of a binary point template, if box = "template". Default is #RRGGBBAA, where AA is the transparency.
off.col	Colors for the off points of a binary point template, if box = "template". Default is #RRGGBBAA, where AA is the transparency.
pt.col	Colors for the points of a correlation template, if box = "template". Default is #RRGGBBAA, where AA is the transparency.

Details

Note that almost all of the arguments have a default value.
The default audio player, "play", is the shell command for SoX, the multi-OS media player. Windows will detect the file type and use the default media player with "start", or you can specify one (such as Windows Media Player) with "start wmplayer.exe". On Ubuntu try Rhythmbox ("rhythmbox"), and on Mac OS try afplay ("afplay").

Value

NULL, invisibly, or, if verify = TRUE, an updated detection list object (detectionList).

Author(s)

Sasha D. Hafner

See Also

findPeaks, plot-methods

Examples

```
# Load data
data(btnw)
data(oven)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
survey.fp <- file.path(tempdir(), "survey2010-12-31_120000_EST.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)
writeWave(survey, survey.fp)

# Correlation example
# Create two correlation templates
wct <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w")
oct <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o")

# Combine them
ctemps <- combineCorTemplates(wct, oct)
```

```
# Calculate scores
cscores <- corMatch(survey.fp, ctemps)

# Find peaks and detections
cdetects <- findPeaks(cscores)
cdetects

# Interactively inspect individual detections, no return value
## Not run:
# Not run because user input is required
showPeaks(detection.obj = cdetects, which.one = "w", flim = c(2, 8), point = TRUE,
          scorelim = c(0, 1))

# Interactively verify individual detections, return adds verification field
cdetects <- showPeaks(detection.obj = cdetects, which.one = "w", flim = c(0, 20),
                      point = TRUE, scorelim = c(0, 1), verify = TRUE)

## End(Not run)
```

---

specCols *Color Vectors for Spectrograms*

Description

Functions to generate a selection of color vectors for spectrograms based on existing color vectors for images in **grDevices**. Vectors are reversed relative to their parent (i.e. numerical sequences progress from 1 to 0 rather than 0 to 1).

Usage

```
gray.1(n = 30)
gray.2(n = 30)
gray.3(n = 30)
rainbow.1(n = 15)
topo.1(n = 12)
```

Arguments

n	A vector of desired color levels between 1 and 0; one indicates high amplitude ("black", "red", or "blue") and zero indicates low amplitude ("white", "purple", or "tan").

Details

The n argument will divide the vector into n color levels.

Value

A vector of colors.

Author(s)

Jon Katz, Sasha D. Hafner

References

Based on the color palettes from **grDevices**, and loosely on those used in **seewave**

See Also

gray, rainbow, topo.colors, terrain.colors

Examples

```
spec.test <- function(mat, spec.col) image(z = t(mat), col = spec.col)

mat <- matrix(1:30, ncol = 6, byrow = TRUE)

spec.test(mat = mat, spec.col = gray.1())
spec.test(mat = mat, spec.col = gray.2())
spec.test(mat = mat, spec.col = gray.3())
spec.test(mat = mat, spec.col = rainbow.1())
spec.test(mat = mat, spec.col = topo.1())

## Not run:
# Colors are defined as:
gray.1 <- function(n = 30) gray(seq(1, 0, length.out = n))
gray.2 <- function(n = 30) gray(1-seq(0, 1, length.out = n)^2)
gray.3 <- function(n = 30) gray(1-seq(0, 1, length.out = n)^3)
rainbow.1 <- function(n = 15) rev(rainbow(n))
topo.1 <- function(n = 12) rev(topo.colors(n))
## End(Not run)
```

---

survey *Sample Acoustic Survey (Short)*

Description

A composite wave file 23.5 seconds long containing 3 black-throated green warbler (*Setophaga virens*) songs (at 1.8, 10.5, and 21.6 seconds) and 4 ovenbird (*Seiurus aurocapilla*) songs (at 5.8, 9.1, 14.8, and 22.0 seconds). The ovenbird song at 14.8 seconds is considerably lower amplitude than the others.

Usage

```
data(survey)
```

Format

The format is:
Formal class 'Wave' [package "tuneR"] with 6 slots ..@ left : int [1:564000] 135 192 230 163 158 230 289 277 249 280 ... ..@ right : num(0) ..@ stereo : logi FALSE ..@ samp.rate: int 24000 ..@ bit : int 16 ..@ pcm : logi TRUE

Source

Sound clips were recorded in Vermont, USA in 2010. Equipment was a Wildlife Acoustics SM1(TM) recorder recording in WAC0 format, converted to wave using the Wildlife Acoustics Wac2Wav (TM) converter. Recording has a sample rate of 24kHz and is 16-bit mono.

Examples

```
data(survey)
viewSpec(survey)
```

---

survey_anno *Annotations for* survey

Description

Data frame containing annotations for the data file survey.

Usage

```
data(survey_anno)
```

Format

The format is: 'data.frame': 7 obs. of 5 variables: $ start.time: num 1.06 4.21 7.55 9.85 13.84 ... $ end.time : num 2.59 7.41 10.7 11.06 15.85 ... $ min.frq : num 3.61 2.58 2.63 3.88 2.82 ... $ max.frq : num 6.35 9.54 9.33 6.25 6.39 ... $ name : Factor w/ 2 levels "BTNW", "OVEN": 1 2 2 1 2 2 1

Details

These annotations can be plotted onto the spectrogram by loading them in with the anno argument of viewSpec.

Examples

```
## Not run:
# View annotations
data(survey)
data(survey_anno)
write.csv(survey_anno, "survey_anno.csv", row.names = FALSE)
viewSpec(survey, annotate = TRUE, anno = "survey_anno.csv")

## End(Not run)
```

Template-class *Class* "Template"

Description

A template is an object with acoustic information (frequency, time, and amplitude) on an animal volcalization. Objects of class "corTemplate" are correlation templates, which contain quantitative data on amplitude. Objects of class "binTemplate" are binary templates, which contain only qualitative data on amplitude: only whether the it is high ("on" cells) or low ("off") cells. The class "Template" is a virtual class, and both types of templates have this class. Templates are always stored as part of a TemplateList, either a corTemplateList or a binTemplateList.

Objects from the Class

Objects can be created by calls of the form new("corTemplate", ...) or new("binTemplate", ...). However, users should not work directly with objects of this class, but only with corTemplateList or binTemplateList, which can be created as described in the documentation for TemplateList.

Slots

clip.path: Object of class character. The file path of the original recording used to create the template.

samp.rate: Object of class integer. The sample rate of the recording.

pt.on: Object of class matrix (binTemplate class only). A two-dimensional matrix with time (column 1) and frequency (column 2) bins for "on" points. Bin locations are relative to the first bin ("on" or "off"), which has a value of 1.

pt.off: Object of class matrix (binTemplate class only). A two-dimensional matrix with time (column 1) and frequency (column 2) bins for "off" points. Bin locations are relative to the first bin ("on" or "off"), which has a value of 1.

pts: Object of class "matrix" (corTemplate class only). A two-dimensional matrix with time (column 1) and frequency (column 2) bins, and amplitude (column 3).

t.step: Object of class numeric. Time step between time bins (sec).

frq.step: Object of class numeric. Frequency step between frequency bins (kHz).

n.t.bins: Object of class integer. Total number of time bins in the template.

first.t.bin: Object of class numeric. Time of the first time bin in the original recording (sec).

n.frq.bins: Object of class integer. Total number of frequency bins.

duration: Object of class numeric. Template duration (sec).

frq.lim: Object of class numeric. Frequency limits (kHz).

wl: Object of class integer. Value of argument wl used in the spectro function call when the template was created.

ovlp: Object of class integer. Value of argument ovlp used in the spectro function call when the template was created.

`wn`: Object of class `character`. Value of argument `wn` used in the `spectro` function call when the template was created.

`score.cutoff`: Object of class `numeric`. The cutoff that will be used to identify detections when this template is used.

Extends

Classes `corTemplate` and `binTemplate` extend `Template`, directly.

Methods

No methods defined with these classes in the signature. But see `TemplateList`.

Author(s)

Sasha D. Hafner

See Also

`binTemplateList`, `corTemplateList`, `TemplateList`

Examples

```
showClass("Template")

showClass("corTemplate")

showClass("binTemplate")
```

---

templateComment *Query or Set Template Cutoffs*

Description

Use this function to add or check comments to templates within template lists (`corTemplateList` or `binTemplateList` objects), scores (`templateScores` objects), or detection list (`detectionList` objects).

Usage

```
templateComment(object)
templateComment(object) <- value
```

Arguments

`object`	A binary or correlation template list (class `binTemplateList` or `corTemplateList`).
`value`	A character vector with the new comment.

Details

`templateComment` is an accessor function and `templateComment <-` is a replacement function. For replacement, the `value` object should be as long as the number of templates in `object` (or the number selecting via indexing) unless it is a named vector (see Examples).

Value

For extraction, a numeric vector of the same length as `object` with comments. For replacement, the updated object.

Author(s)

Sasha D. Hafner

See Also

`templateNames`, `templateCutoff`, `getTemplates`

Examples

```
# Load data
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Create four correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")
oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")
oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o2")

# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)
ctemps

# Add a comment for two templates
templateComment(ctemps) <- c(w1 = "This is the best template so far.",
                             o1 = "Should we drop the lowest syllable?")

# Add a default comment also
templateComment(ctemps) <- c(w1 = "This is the best template so far.",
                             o1 = "Should we drop the lowest syllable?",
                             default = "These templates have not been tested.")
# View comments
templateComment(ctemps)
```

```
# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
```

`templateCutoff` *Query or Set Template Cutoffs*

Description

Use this function to check or change the values of score cutoff in template lists (`corTemplateList` or `binTemplateList` objects), scores (`templateScores` objects), or detections list (`detectionList` objects).

Usage

```
templateCutoff(object)
templateCutoff(object) <- value
```

Arguments

`object`	A binary or correlation template list (class `binTemplateList` or `corTemplateList`).
`value`	A numeric vector with the new score cutoff.

Details

`templateCutoff` is an accessor function and `templateCutoff <-` is a replacement function. For replacement, the `value` object should be as long as the number of templates in `object` (or the number selecting via indexing) unless it is a named vector (see Examples).

Value

For extraction, a numeric vector of the same length as `object` with score cutoffs. For replacement, the updated object.

Author(s)

Sasha D. Hafner

See Also

`templateNames`, `templateComment`

Examples

```
# Load data
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Create four correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")
oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")
oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o2")

# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)
ctemps

# Check cutoffs
templateCutoff(ctemps)

# Change all like this
templateCutoff(ctemps) <- c(0.35, 0.35, 0.35, 0.35)
# or this
templateCutoff(ctemps) <- c(default = 0.35)

# Change select ones like this
templateCutoff(ctemps) <- c(o1 = 0.45, o2 = 0.45)
# or this
templateCutoff(ctemps)[c(3, 4)] <- 0.45

# Could combine these two steps
templateCutoff(ctemps) <- c(default = 0.35, o1 = 0.45, o2 = 0.45)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
```

TemplateList-class *Class* "TemplateList"

Description

A template is an object with acoustic information (frequency, time, and volume) on an animal vocalization. In monitoR, all templates are stored within a template list, which has the (virtual) class `TemplateList`. Because the structure of the two types of templates differs slightly (see `Template`),

there are actually two classes for template lists: `corTemplateList` and `binTemplateList`, and the virtual class `TemplateList` (which includes both types of template lists) is used to define most methods.

Objects from the Class

Objects can be created by calls of the form `new("corTemplateList", ...)` or `new("binTemplateList", ...)`. However, objects should always be created with the template-creation functions `makeCorTemplate` or `makeBinTemplate`, or else by reading from a file using `readCorTemplates` or `readBinTemplates`. There are also functions for modifying existing template lists or extracting template lists from other objects.

Slots

`templates`: Object of class `"list"` A list of either `corTemplate` or `binTemplate` objects.

Extends

Classes `corTemplateList` and `binTemplateList` extend the virtual class `TemplateList`, directly.

Methods

show `signature(object = "corTemplateList")`: ...

summary `signature(object = "corTemplateList")`: ...

show `signature(object = "binTemplateList")`: ...

summary `signature(object = "binTemplateList")`: ...

plot `signature(x = "TemplateList", y = "ANY")`: ...

Note

For details on the structure of the actual templates, see `Template`.

Author(s)

Sasha D. Hafner

See Also

`Template`, `combineBinTemplates`, `templateCutoff`, `templateComment`, `getTemplates`, `plot-methods`, `[-methods`

Examples

```
showClass("TemplateList")

showClass("corTemplateList")

showClass("binTemplateList")
```

templateMatching *Calculate Spectrogram Template Matching Scores*

Description

These functions are used to calculate spectrogram template matching scores between a set of templates and an acoustic survey using spectrogram cross correlation (corMatch) or binary point matching (binMatch).

Usage

```
corMatch(survey, templates, parallel = FALSE, show.prog = FALSE, cor.method = "pearson",
      time.source = "filename", rec.tz = NA, write.wav = FALSE, quiet = FALSE, ...)

binMatch(survey, templates, parallel = FALSE, show.prog = FALSE, time.source = "filename",
        rec.tz = NA, write.wav = FALSE, report.amp = FALSE, quiet = FALSE, ...)
```

Arguments

survey	A file path to a wav or mp3 recording, or a Wave object. The survey is the acoustic survey that you want to make detections within.
templates	A template list–a corTemplateList object for corMatch or a binTemplateList object for binMatch.
parallel	If TRUE, mclapply is used for calculation of scores across all time bins for each template. This option is not available for Windows operating systems.
show.prog	If TRUE, progress will be reported during the score calculations.
cor.method	For corMatch, the method used to calculate correlation coefficients (see ?cor).
time.source	The source of date and time information. filename will look in the name of the survey file (survey argument) for a date and time with format YYYY-MM-DD_HHMMSS_TimeZone. "fileinfo" will take the date and time from the file modification information. See details.
rec.tz	Time zone for which the recordings were made (optional). Needed if different from the time zone setting of the operating system, when times will be adjusted to the 'correct' time zone. See details.
write.wav	If survey is a Wave object, should it be written to file? If FALSE, functions will return an error.
report.amp	If TRUE, binMatch will return the mean "on" and "off" amplitudes as well as their difference (the score). See details.
quiet	Use TRUE to suppress status updates to the console. Does not suppress messages or warnings.
...	Additional arguments to the spectro function.

Details

Scores are refereced by both the time elapsed since the beginning of the recording and the time of day on the date the recording was made. For times derived from the date modified of the recording file (`time.source = "fileinfo"`) to be accurate the sound file must not have been edited (no samples added or removed) since its original creation. File copying and duplication (as from removeable media to a storage drive) should not affect the date modified, although the creation date will be reset. Date modified values are stored in the time zone when they were recorded but will be translated to the current time zone when read, which may result in errors due to daylight savings changes or when recorded surveys are shared across time zones. Time zone management is tricky; if recordings were made in a different time zone than the operating system running `fileCopyRename`, you can specify the correct time zone for the recordings with the `rec.tz` argument. Unexpected results are possible, as time zone abbreviations in general use may not match those in the Internet Assigned Numbers Authority tz database. The most reliable way to specify time zone is to use the full name, most quickly seen using `OlsonNames`, and also found on Wikipedia: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones. Times derived from a date-time value encoded in the file name (`time.source = "filename"`) are more stable in regard, and are automatically created with either `fileCopyRename` or `mp3Subsamp`.
Binary point matching scores each time frame by computing the difference between the mean amplitude in the "on" cells and the mean amplitude in the "off" cells. The resulting score can be a rough estimate of signal:noise.

Value

An S4 object of class `templateScores`, with the following slots:

`survey.name`	The file path to the survey that the scores apply to.
`survey`	The actual survey as a `Wave` object.
`survey.data`	A named list with one element per template. Each element is a named list with time-domain results for the survey.
`templates`	The templates (an S4 object of class `corTemplateList` or `binTemplateList`) used to calculate the scores.
`scores`	A named list with an element for each template. Each element contains the scores for an individual template.
`time`	A character vector containing information on the run time.

Note

Cross-correlation values are not normalized.

Note

For examples, see `findPeaks` and `getDetections`.

Author(s)

Sasha D. Hafner and Jon Katz

References

Mellinger, D. K. and C. W. Clark. 1997. Methods for automatic detection of mysticete sounds. *Marine and Freshwater Behaviour and Physiology*. **29**, 163-181.

Towsey, M., B. Planitz, A. Nantes, J. Wimmer, and P. Roe. 2012. A toolbox for animal call recognition. *Bioacoustics-the International Journal of Animal Sound and Its Recording* **21**, 107-125.

See Also

makeCorTemplate, makeBinTemplate, findPeaks, getDetections, getPeaks, fileCopyRename, mp3Subsamp

---

templateNames *Names of Templates*

Description

Functions to check or change the names of templates within an acoustic template list.

Usage

```
templateNames(object)
templateNames(object) <- value
```

Arguments

object	An acoustic template list, i.e., a corTemplateList or binTemplateList object.
value	A character vector of names. May be named.

Details

This function is analogous to the function names.

Value

For names, NULL or a character vector of the same length as object. For names <- , the updated template list, i.e., the original template list with only the names changed.

Author(s)

Sasha D. Hafner

See Also

makeCorTemplate, makeBinTemplate, templateComment, templateCutoff

Examples

```
# Load data
data(btnw)
data(oven)
data(survey)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Create four correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")
oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")
oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o2")

# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)
ctemps

# To check template names
templateNames(ctemps)

# Change the first two
templateNames(ctemps)[1:2] <- c("warbler 1", "warbler 2")

# Change all
templateNames(ctemps) <- c("a", "b", "c", "d")

# To check template names
templateNames(ctemps)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
```

---

templatePath *Song clip path of Templates*

Description

Functions to check or change the song clip path of templates within an acoustic template list.

Usage

```
templatePath(object)
templatePath(object) <- value
```

Arguments

object	An acoustic template list, i.e., a corTemplateList or binTemplateList object.
value	A character vector of paths. May be named.

Details

This function works in the same way as the function names. No check is performed to ensure that the specified path is valid.

Value

For filePath, NULL or a character vector of the same length as object. For filePath <- , the updated template list, i.e., the original template list with only the clip.path values changed.

Author(s)

Sasha D. Hafner

See Also

makeCorTemplate, makeBinTemplate, templateComment, templateCutoff, templateNames,

Examples

```
# Load data
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Create four correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")
oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")
oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o2")

# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)
ctemps

# To check paths
templatePath(ctemps)

# Change the first two
templatePath(ctemps)[1:2] <- c("~/templates/btnw.wav", "~/templates/btnw.wav")

# Clean up (only because these files were created in these examples)
```

```
file.remove(btnw.fp)
file.remove(oven.fp)
```

---

templateScores-class *Class* "templateScores"

Description

These objects contain template scores, which indicate how well templates match a single survey recording, with a value for each time bin. Additionally, all the objects which were used to create these scores are also saved within the objects. Objects of this class represent an intermediate step in the template detection process–detections need to be found in the scores using findPeaks.

Objects from the Class

Objects can be created by calls of the form new("templateScores", ...). However, they should always be created with the corMatch or binMatch function.

Slots

survey.name: Object of class character. The name of the survey file, or "A Wave object" if the survey was not read in from a file.

survey: Object of class Wave. The survey data, as a "Wave" object.

survey.data: Object of class list. A named list, with one element for each template. Each element contains data from a Fourier transform of the original survey: amp is a matrix of amplitudes (frequency by time), t.bins is a numeric vector with the values of the time bins (left-aligned–first bin is always 0.0), and frq.bins is a numeric vector with the values of the frequency bins (top-aligned–last bin is always the upper limit). There is a separate element for each template because each template may use different parameters for the Fourier transform (see Template).

templates: Object of class list. A named list of templates, which is identical to the original TemplateList used for template matching. This template list can be extracted with getTemplates.

scores: Object of class list. A named list, with one element for each template. Each element is a data frame with three columns: date.time is the absolute time of the score, time is the relative time of the score (relative to the survey start), and score is the score. Times are based on the center of the template, and so time will not correspond to values in t.bins in the survey.data above if the template spans an even number of time bins.

time: Object of class character. Information on the time corMatch or binMatch took to run. The first element is the run time (s), and the second element is "real-time factor" (survey length divided by the run time).

Methods

show signature(object = "templateScores"): ...

summary signature(object = "templateScores"): ...

Author(s)

Sasha D. Hafner

See Also

findPeaks, detectionList

Examples

```
showClass("templateScores")
```

`timeAlign` *Condense Detections or Peaks from Multiple Templates*

Description

Condense detections or peaks from a number of templates (of the same detection type); events that occur within an adjustable time buffer of one another are assumed to be duplicate detections. In such cases the event with the highest score is saved. Functions with detections for a single species or multiple species.

Usage

```
timeAlign(x, what = "detections", tol = 1)
```

Arguments

`x`	An object of class `detectionList`, a single data frame of detections, or list of either file paths to a csv file or of data frames.
`what`	Character, in `c("detections", "peaks")`. Detections are peaks above a score cutoff. Peaks are all peaks. Required only if `x` is of class `detectionList`
`tol`	Numeric value for tolerance, with units seconds. If a detected event is within this value (actually +/- 0.5`tol`), the events are assumed to co-occur and be of the same origin. A somewhat arbitrary value (like epsilon), but should be less than 2/3 the template duration.

Details

If input is an object of class `detectionList`, a single data frame, or list of either file paths or data frames. Must be called for each survey.

Value

Returns a single data frame of detections (the input `x`) with duplicated events removed, leaving only the event that had the highest score.

Note

Events are assumed to be duplicated if they co-occur within a time duration of `tol`, but they are only compared to the event above and below when ordered by time. Events with similar times can be spuriously discarded if `tol` is set larger than the separation of unrelated peaks. Excessive deletion of events may also occur if the value for `tol` is set larger than the duration of the template. Note that in this function `tol` specifies seconds, whereas in `findPeaks` `tol` specifies a ratio.

Author(s)

Jon Katz

See Also

The function `eventEval` operates similarly, but rather than merge detection results from multiple templates it compares them to known events and reports the True +, True -, False +, and False - rates.

Examples

```
## Not run:
# Not run because it will create files in user's working directory
data(survey)
data(btnw)

writeWave(btnw, "btnw.wav")

btnw2 <- cutw(survey, from = 0.75, to = 3)

writeWave(btnw2, "btnw2.wav")

# Template construction
btnw1 <- makeBinTemplate(
        "btnw.wav",
        frq.lim = c(2, 8),
        select = "auto",
        name = "btnw1",
        buffer = 4,
        amp.cutoff = -31,
        binary = TRUE)

btnw2 <- makeBinTemplate(
        "btnw2.wav",
        frq.lim = c(2, 8),
        select = "auto",
        name = "btnw2",
        buffer = 4,
        amp.cutoff = -24,
        binary = TRUE)

# Join templates
btnw <- combineBinTemplates(btnw1, btnw2)
```

```
# Binary point matching
scores <- binMatch(survey = survey, templates = btnw, time.source = 'fileinfo')

# Isolate peaks
pks <- findPeaks(scores)

# View detections
getDetections(pks)

# Compare to output of timeAlign
timeAlign(pks)
## End(Not run)
```

---

viewSpec *Interactively View and Annotate Spectrograms*

Description

Interactively page through short or long spectrograms of wav or mp3 files or Wave objects. Extract short or long wave files, play audio while viewing spectrogram, and annotate sounds in the spectrogram. Load annotations from csv files for viewing.

Usage

```
viewSpec(clip, interactive = FALSE, start.time = 0,
    units = "seconds", page.length = 30,
    annotate = FALSE, anno, channel = "left",
    output.dir = getwd(), frq.lim = c(0, 12), spec.col = gray.3(),
    page.ovlp = 0.25, player = "play", wl = 512, ovlp = 0,
    wn = "hanning", consistent = TRUE,
    mp3.meta = list(kbps = 128, samp.rate = 44100, stereo = TRUE),
    main = NULL, ...)
```

Arguments

clip	File path to wav file, mp3 file, or wave object. See Details.
interactive	Logical. FALSE displays the first 30 seconds (or more, if page.length is increased) of a spectrogram. TRUE enables the options to page through spectrograms, zoom in time and frequency, play, extract segments, and annotate. See Details.
start.time	Time in file to start reading.
units	Units for start.time. Available units are c("seconds", "minutes", "hours") Defaults to "seconds".
page.length	Duration of page length to view, in seconds. Can be repeatedly halved and doubled within the function.

annotate	Logical, to allow sounds to be highlighted and named on the spectrogram. See Details.
anno	Character, file path to csv containing annotations. Read in only if annotate = TRUE.
channel	Character value in c("left", "right", "both"). Stereo recordings may be viewed as single channel or multi-channel spectrograms. See Details.
output.dir	File path to directory where extracted clips and annotations will be saved, if other than the current working directory.
frq.lim	Initial frequency limits to spectrogram, in kHz. Accepts a 2 element vector. Can be adjusted from within the function.
spec.col	Color (or grayscale) gradient to apply to the spectrogram. See Details.
page.ovlp	Numeric value between 0 and 1. Proportion of page.length to overlap when moving to a new page.
player	Character value specifying an audio player to play the portion of the file corresponding to the visible spectrogram.
wl	Numeric value specifying number of samples per window in the Fourier Transform. Accepts powers of 2: c(128, 256, 512, 1024, 2048)
ovlp	Numeric value specifying window overlap in the Fourier Transform. Specified as a percent between 0 and 99.
wn	Character value specifying window function in the Fourier Transform. Defaults to "hanning"; "hamming" is also implemented.
consistent	Logical, offers a method of maintaining color gradient map from page to page. See Details.
mp3.meta	List of metadata used when paging through mp3 files using mp3splt. kbps is the compression rate, samp.rate is the sample rate, and stereo is logical where TRUE represents both stereo and JntStereo.
main	Optional character object with which to name the spectrogram. If NULL the file name will be used if possible.
...	Additional arguments to spectro

Details

When interactive = TRUE, during the function session the console will display a command menu that prints commands to scroll or nudge to the next/previous page, zoom in/out in the time axis (by halving or doubling the page.length), play the page, save the page as a wave file, change spectrogram parameters (e.g. frq.lim, start.time, wl, ovlp, etc), or quit. An option not presented on-screen is "i" to identify the RMS amplitude in a selected portion of the spectrogram.

viewSpec relies on the WaveIO functions in tuneR, with some modifications. Seeking in wave files and wave objects is accurate to the nearest sample, but the decoding required for mp3 files is "bare bones". Users can install the software mp3splt which will allow seeking in mp3 files very similar (albeit slightly less accurate) to that that exists for wave files. When using mp3splt a short mp3 file the duration of each page is extracted from the clip file or object and saved to the working

directory for each new page.

When annotation is set to TRUE the default is to start a new annotation file, unless a csv file containing annotations is specified with the argument anno. Annotation adds the option to annotate to the console command menu, and annotations can be made after typing "a" into the console and pressing enter. Annotation is accomplished by selecting first the upper-left corner of a bounding box around an event in the spectrogram followed by the lower-right corner; after the selection is complete the console will prompt to name the annotation. At a minimum the first annotation must be named, but subsequent annotations will recycle the previous name if a new one is not provided. When in annotation mode the console menu is not shown; instructions for annotation are displayed instead. To exit annotation mode right-click an appropriate number of times, and the console command menu will return. One or more annotations can be deleted by typing "d" in the console after the command menu is displayed, then bounding all annotations to delete in the same manner as if creating a new annotation. Annotations are saved when the command to exit the function is initiated ("q"). Occasionally unrecognized commands may cause the function to exit before annotations can be saved; to guard against losing annotations in such an event, annotations are auto-saved to a file called "TMPannotations.csv" in the working directory, from where they can be retrieved until written over during the next session. Annotation is only possible in one channel per function invocation. The channel will revert to "left" if annotate = TRUE and channel = "both".

Spectrogram colors are adjustable, and users may opt to create their own gradients for display. A few are provided with monitoR including gray.1, gray.2, gray.3, rainbow.1, and topo.1, all of which are based on existing R colors. The gradient is mapped to the values in the spectrogram each time the page is loaded. In gray.2, for example, this means that every page will display the highest dB value as black and the lowest value as white. The highest dB value likely changes from page to page, which can result in successive pages being displayed with wildly different color values. Setting consistent = TRUE (the default) offers a way to minimize this effect, as it artificially weights a single cell in the lower-left corner with a value of 0 dB, which is usually mapped to a black. Under normal circumstances this artificially black cell will not be noticed, but at high magnification it may stand out as erroneous, in which case setting consistent = FALSE may be warranted.

Spectrograms of existing Wave objects are titled with the first argument of the call, which is assumed to be clip.

The default audio player, "play", is the shell command for SoX, the multi-OS media player. Windows will detect the file type and use the default media player with "start", or you can specify one (such as Windows Media Player) with "start wmplayer.exe". On Ubuntu try Rhythmbox ("rhythmbox"), and on Mac OS try afplay ("afplay").

Value

A spectrogram plot. Certain options invoked during the function may write new wave or csv files to the working directory.

Note

The time axis is presented with a fair amount of rounding. It becomes progressively more accurate as the zoom level increases.

Author(s)

Jon Katz, Sasha D. Hafner

See Also

dbUploadAnno

Examples

```
data(survey)
viewSpec(survey)

## Not run:
# Start a new annotation file
viewSpec(survey, annotate = TRUE)

# View previous annotations
data(survey_anno)
write.csv(survey_anno, "survey_anno.csv", row.names = FALSE)
viewSpec(survey, interactive = TRUE, annotate = TRUE, anno = "survey_anno.csv", start.time = 5)

# Disable consistent spectrograms
viewSpec(survey, interactive = TRUE, annotate = TRUE, page.length = 10, consistent = FALSE)

## End(Not run)
```

---

writeTemplates *Write Acoustic Templates to Text Files*

Description

These functions write all templates within a template list to text files within a specified directory.

Usage

```
writeCorTemplates(..., dir = ".", ext = "ct", parallel = FALSE)
writeBinTemplates(..., dir = ".", ext = "bt", parallel = FALSE)
```

Arguments

...	One or more template lists.
dir	A file path to the directory where the files should be saved. If it doesn't exist, the function will create it. By default, the working directory.
ext	The file extension used for the new file(s).
parallel	Set to TRUE to use mclapply from the parallel package to speed up the call for large template lists (not available for Windows operating systems).

Details

For correlation templates (class corTemplateList) use writeCorTemplates, and use writeBinTemplates for binary templates (class linkS4class{binTemplateList}). To write only some of the templates in a list to file, use indexing ([-methods).

Value

NULL, invisibly.

Author(s)

Sasha D. Hafner

See Also

makeCorTemplate, makeBinTemplate, readBinTemplates, readCorTemplates

Examples

```
# Load data
data(btnw)
data(oven)

# Write Wave objects to file (temporary directory used here)
btnw.fp <- file.path(tempdir(), "btnw.wav")
oven.fp <- file.path(tempdir(), "oven.wav")
writeWave(btnw, btnw.fp)
writeWave(oven, oven.fp)

# Create four correlation templates
wct1 <- makeCorTemplate(btnw.fp, name = "w1")
wct2 <- makeCorTemplate(btnw.fp, t.lim = c(1.5, 2.1), frq.lim = c(4.2, 5.6), name = "w2")
oct1 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), name = "o1")
oct2 <- makeCorTemplate(oven.fp, t.lim = c(1, 4), frq.lim = c(1, 11), dens = 0.1, name = "o2")

# Combine all of them
ctemps <- combineCorTemplates(wct1, wct2, oct1, oct2)

# To write ctemps to a directory "templates"
## Not run:
# Not run because it will write files outside of user's temporary directory
writeCorTemplates(ctemps, dir = "templates")

## End(Not run)

# Clean up (only because these files were created in these examples)
file.remove(btnw.fp)
file.remove(oven.fp)
```

Index