Package: sequenceR (via r-universe)

January 14, 2025

Type Package

Title A Simple Sequencer for Data Sonification

Version 1.0.1

Description A rudimentary sequencer to define, manipulate and mix sound samples. The underlying motivation is to sonify data, as demonstrated in the blog <https://globxblog.github.io/>, the presentation by Renard and Le Bescond (2022, <https://hal.science/hal-03710340v1>) or the poster by Renard et al. (2023, <https://hal.inrae.fr/hal-04388845v1>).

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/benRenard/sequenceR

BugReports https://github.com/benRenard/sequenceR/issues

Depends R (>= 3.5.0)

Imports tuneR

Suggests knitr, rmarkdown, ggplot2, gganimate, tidyr, dplyr, av

RoxygenNote 7.2.3

VignetteBuilder knitr

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

RemoteUrl https://github.com/benrenard/sequencer

RemoteRef HEAD

RemoteSha eab49579fe5aa5e1f811913cf08e5dd5c45792a9

Contents

`applyDelay` *Delay effect*

Description

Apply a delay to a sound sample. See https://en.wikipedia.org/wiki/Comb_filter

Usage

```
applyDelay(sample, type = "feedforward", delayTime = 0.6, echoes = c(0.8))
```

Arguments

`sample`	soundSample object, input sample
`type`	Character string, the delay type: feedforward or feedback
`delayTime`	Numeric >0, delay time in s.
`echoes`	Numeric vector >0. The size of the vector gives the number of echoes, the values the level of each echo (generally decreases to 0).

Value

The sound sample with a delay effect

Examples

```
# example code
notes=c('E3','G3','A3','B3','D4','E4','G4')
synth=getSynth(notes)
raw=as.soundSample(play.instrument(synth,notes=notes[c(1,2,3,2,3,4,3,4,5,4,5,6,5,6,7)]))
plot(raw)
## Not run:
# All calls to function 'listen' are wrapped in \dontrun{} since
# they rely on an external audio player to listen to the audio samples.
# See ?tuneR::setWavPlayer for setting a default player.
listen(raw)
## End(Not run)
# Single echo by default
cooked=applyDelay(raw)
plot(cooked)
## Not run: listen(cooked)
# Multiple echoes
cooked=applyDelay(raw,echoes=1/(1:10))
plot(cooked)
## Not run: listen(cooked)
# Feedback-type delay
cooked=applyDelay(raw,echoes=1/(1:10),type='feedback')
plot(cooked)
## Not run: listen(cooked)
```

applyDisto *Distortion effect*

Description

Apply a distortion to a sound sample

Usage

```
applyDisto(sample, type = c("clip", "tanh"), level = 2, ..., rescale = FALSE)
```

Arguments

sample	soundSample object, input sample
type	Character string, the distortion type
level	Numeric >0, distortion level
...	other parameters passed to the distortion transfer function
rescale	Logical. If TRUE, the soundSample wave is rescaled to [-1,1]

Value

The distorted sound sample

Examples

```
# example code
raw=oscillator(freq=110,duration=0.5)
plot(raw)
dist=applyDisto(raw,type='tanh',level=5)
plot(dist)
```

applyEnvelope *Apply an envelope*

Description

Apply a volume envelope to a sound sample.

Usage

```
applyEnvelope(sample, env)
```

Arguments

sample	Sound sample object.
env	Envelope object. Envelope values should all be between 0 and 1.

Value

A sound sample object.

Examples

```
# Define the sound sample
sam <- soundSample(sin(2*pi*seq(0,0.5,1/44100)*220)) # 0.5-second A (220 Hz)
# Define the envelope
env <- envelope(t=c(0,0.03,1),v=c(0,1,0))
# Apply it
res <- applyEnvelope(sam,env)
# Compare waveforms
plot(sam,main='before')
plot(res,main='after')
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(res)
## End(Not run)
```

---

as.soundSample *Cast to a sound sample*

Description

Convert a tuneR::Wave object into a soundSample.

Usage

```
as.soundSample(w, pan = 0)
```

Arguments

w	tuneR Wave object
pan	Numeric in [-1;1], panoramic. -1 (resp. 1) only select the left (resp. right) channel of w (if the latter is stereo). 0 averages both channels

Value

An object of class 'soundSample'.

Examples

```
w <- tuneR::Wave(left=sin(2*pi*seq(0,1,,44100)*440)) # 1-second A
sam <- as.soundSample(w)
plot(sam)
```

as.Wave *Cast to a tuneR::Wave object*

Description

Convert a soundSample into a tuneR::Wave object.

Usage

```
as.Wave(x)
```

Arguments

x	sound sample object.

Value

a tuneR Wave object.

Examples

```
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)) # 1-second A (440 Hz)
w <- as.Wave(sam)
tuneR::plot(w)
```

bell *Bell sample*

Description

A ride cymbal (hit ion the bell) sound sample object

Usage

```
bell
```

Format

An object of class soundSample of length 4.

checkMaxSize *Check wave size*

Description

Check that the size of a wave does not exceed the maximum allowed size.

Usage

```
checkMaxSize(n, nmax)
```

Arguments

n	integer, size to be checked
nmax	integer, maximum allowed size

Value

nothing - just stops execution with an error message if n>nmax

checkSeqArgs *Check sequencer arguments*

Description

Check that the arguments used in sequencing functions (e.g. time, volume, pan, etc.) are valid.

Usage

```
checkSeqArgs(argList)
```

Arguments

argList	list, a named list containg the arguments

Value

nothing - just stops execution with an error message if something is invalid

disto_clip *Clip distortion*

Description

Transfer function for 'clip' distortion

Usage

```
disto_clip(x, level)
```

Arguments

x	Numeric vector in [-1,1], input signal
level	Numeric (>=0), distortion level

Value

a numeric vector containing the distorted output signal

disto_tanh *Tanh distortion*

Description

Transfer function for 'tanh' distortion

Usage

```
disto_tanh(x, level)
```

Arguments

x	Numeric vector in [-1,1], input signal
level	Numeric (>=0), distortion level

Value

a numeric vector containing the distorted output signal

envelope *Envelope constructor.*

Description

Creates a new instance of an 'envelope' object (https://en.wikipedia.org/wiki/Envelope_(music)). In this package an envelop is viewed as a curve v(t), where t is the time and v the value of the envelope. Time t is normalized between 0 and 1 so that 1 corresponds to the end of the sound sample the envelope is applied to (and 0 to its beginning). The curve is defined by a discrete set of points (t,v) (linear interpolation in between).

Usage

```
envelope(t, v)
```

Arguments

t	Numeric vector, normalized time. Vector of increasing values starting at 0 and ending at 1.
v	Numeric vector, same size as t, envelop values v(t).

Value

An object of class 'envelope'.

Examples

```
# A triangular envelop
env <- envelope(t=c(0,0.3,1),v=c(0,1,0))
# An ADSR envelope (https://en.wikipedia.org/wiki/Envelope_(music)#ADSR)
env <- envelope(t=c(0,0.1,0.3,0.8,1),v=c(0,1,0.4,0.4,0))
# An envelope that could be used for a 1-octave frequency modulation (from 440 to 220 Hz)
env <- envelope(t=c(0,1),v=c(440,220))
# An envelope that could be used for phase modulation
# (https://en.wikipedia.org/wiki/Phase_modulation)
env <- envelope(t=seq(0,1,0.01),v=(-pi/2)*sin(2*pi*4*seq(0,1,0.01)))
```

getFrequencies *Notes-to-frequencies function*

Description

Get frequencies from note names (in scientific pitch notation).

Usage

```
getFrequencies(notes, minOctave = 0, maxOctave = 8)
```

Arguments

notes	Character vector, note names.
minOctave	integer, smallest (lowest-pitched) octave
maxOctave	integer, largest (highest-pitched) octave

Value

a numeric vector of frequencies (in Hz)

Examples

```
# example code
getFrequencies(c('A3','A4','A5','C#6','Db6','A9','X0'))
getFrequencies(c('A3','A4','A5','C#6','Db6','A9','X0'),maxOctave=9)
```

---

getHarmonics *Harmonics sound sample*

Description

Creates a sound sample corresponding to the kth harmonics of a given frequency

Usage

```
getHarmonics(
  freq,
  k,
  peak = 0.03,
  decay = 0.8,
  duration = 1,
  sustain = 0.25,
  type = "sine"
)
```

Arguments

freq	Numeric, base frequency in Hz
k	Integer >=1, kth harmonics
peak	Numeric, peak time in seconds
decay	Numeric, end-of-decay time in seconds
duration	Numeric, total duration in seconds
sustain	Numeric, sustain volume
type	String, oscillator type, one of 'sine', 'saw', 'square' or 'triangle'. If an unknowm string is provided, a sine oscillator will be used.

Value

An object of class 'soundSample'.

Examples

```
sam1 <- getHarmonics(440,1)
plot(sam1)
sam2 <- getHarmonics(440,3)
plot(sam2)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam2)
## End(Not run)
```

`getNotes` *Frequencies-to-notes function*

Description

Get notes (in scientific pitch notation) from frequencies. The note with the closest frequency is returned.

Usage

```
getNotes(frequencies, minOctave = 0, maxOctave = 8, option = "b")
```

Arguments

`frequencies`	numeric vector, frequencies in Hz
`minOctave`	integer, smallest (lowest-pitched) octave
`maxOctave`	integer, largest (highest-pitched) octave
`option`	character, use 'b' or '#' in note names?

Value

a character vector of notes

Examples

```
# example code
getNotes(seq(440,10000,100))
getNotes(seq(440,10000,100),maxOctave=10,option='#')
```

`getSynth` *Synthesizer*

Description

Creates an additive, Hammond-inspired Synthesizer. Higher harmonics decay faster and have smaller sustain.

Usage

```
getSynth(
  notes,
  nHarmonics = 5,
  peak = 0.03,
  decay = 0.8,
  duration = 1,
  sustain = 0.25,
  decayPar = 1,
  sustainPar = 4,
  type = "sine"
)
```

Arguments

`notes`	Character vector, note names
`nHarmonics`	Integer >=1, number of harmonics
`peak`	Numeric, peak time in seconds
`decay`	Numeric, end-of-decay time in seconds
`duration`	Numeric, total duration in seconds
`sustain`	Numeric, sustain volume
`decayPar`	Numeric, the higher the value the smaller the decay time for higher harmonics
`sustainPar`	Numeric, the higher the value the smaller the sustain volume for higher harmonics
`type`	String, oscillator type, one of 'sine', 'saw', 'square' or 'triangle'. If an unknown string is provided, a sine oscillator will be used.

Value

An object of class 'instrument'.

Examples

```
synth <- getSynth(c('E2','B2','E3','G3','A3'))
w=play.instrument(synth,time=(0:(length(synth)-1))*0.5,fadeout=rep(Inf,length(synth)))
tuneR::plot(w)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to play the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
tuneR::play(w)
## End(Not run)
```

`getSynthNote` *Single note from a synthesizer*

Description

Creates one note with frequency freq from an additive, Hammond-inspired synth. Higher harmonics decay faster and have smaller sustain.

Usage

```
getSynthNote(
  freq,
  nHarmonics = 5,
  peak = 0.03,
  decay = 0.8,
  duration = 1,
  sustain = 0.25,
  decayPar = 1,
  sustainPar = 4,
  type = "sine"
)
```

Arguments

`freq`	Numeric, base frequency in Hz
`nHarmonics`	Integer >=1, number of harmonics
`peak`	Numeric, peak time in seconds
`decay`	Numeric, end-of-decay time in seconds
`duration`	Numeric, total duration in seconds
`sustain`	Numeric, sustain volume
`decayPar`	Numeric, the higher the value the smaller the decay time for higher harmonics
`sustainPar`	Numeric, the higher the value the smaller the sustain volume for higher harmonics
`type`	String, oscillator type, one of 'sine', 'saw', 'square' or 'triangle'. If an unknown string is provided, a sine oscillator will be used.

Value

An object of class 'soundSample'.

Examples

```
sam <- getSynthNote(440,nHarmonics=7)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

---

getTime *Get sampling time*

Description

Get the times steps associated with a sound sample.

Usage

```
getTime(x)
```

Arguments

x	sound sample object.

Value

a numeric vector containing the sampling times in second.

Examples

```
# Define sound sample
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)+0.1*rnorm(44100)) # 1-second noisy A
# Compute sampling times
timeSteps=getTime(sam)
```

globalT *Global Temperature Anomalies dataset*

Description

Times series of annual temperature anomalies at the global scale, 1850-2021. This time series is the one used to create the Warming Stripes (https://www.climate-lab-book.ac.uk/2018/warming-stripes/).

Usage

```
globalT
```

Format

An object of class `data.frame` with 172 rows and 2 columns.

Source

https://www.metoffice.gov.uk/hadobs/hadcrut4/data/current/download.html

hiHat *Hi-hat sample*

Description

A hi-hat sound sample object

Usage

```
hiHat
```

Format

An object of class `soundSample` of length 4.

hiHat2 *Hi-hat sample 2*

Description

A hi-hat sound sample object

Usage

```
hiHat2
```

Format

An object of class `soundSample` of length 4.

Source

http://www.archive.org/details/OpenPathMusic44V1

hiHat_o *Open Hi-hat sample*

Description

An open hi-hat sound sample object

Usage

```
hiHat_o
```

Format

An object of class `soundSample` of length 4.

instrument *Instrument constructor.*

Description

Creates a new instance of an 'instrument' object. An instrument is a named list of sound samples (all with the same sampling rate).

Usage

```
instrument(samples, notes = as.character(1:length(samples)))
```

Arguments

samples	list of sound samples
notes	string vector, name given to each sample

Value

An object of class 'Instrument'.

Examples

```
drumset <- instrument(samples=list(kick,snare,hiHat),notes=c('boom','tat','cheet'))
```

kick *Kick sample*

Description

A kick sound sample object

Usage

```
kick
```

Format

An object of class soundSample of length 4.

kick2 *Kick sample2*

Description

A kick sound sample object

Usage

```
kick2
```

Format

An object of class `soundSample` of length 4.

Source

http://www.archive.org/details/OpenPathMusic44V1

listen *Listen to a sound sample*

Description

Listen to a sound sample. Based on tuneR function 'play'

Usage

```
listen(x)
```

Arguments

x	sound sample object.

Value

nothing - listening function.

Examples

```
# Define sound sample
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)+0.1*rnorm(44100)) # 1-second noisy A
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

mini909 *TR-909 minimalistic drumkit*

Description

An instrument containing a few basic sounds from a TR-909-inspired drumkit

Usage

```
mini909
```

Format

An object of class `instrument` of length 6.

Source

https://freesound.org/people/altemark/packs/1643/

mix *Mix several waves*

Description

Take several wave objects (package tuneR) and mix them according to volume and pan.

Usage

```
mix(waves, volume = rep(1, length(waves)), pan = rep(0, length(waves)))
```

Arguments

`waves`	List of wave S4 objects (tuneR)
`volume`	Numeric vector, volume between 0 and 1.
`pan`	Numeric vector, pan between -1 (left) and 1 (right) (0 = centered).

Value

the result of th mix, an S4 Wave object (from package tuneR).

Examples

```
# A 2-second drum groove (4/4 measure)
# hi-hat on 16th notes
hh <- sequence(hiHat,time=2*(0:15)/16,volume=rep(c(1,rep(0.5,3)),4))
# bass kick on 1 and 3
k <- sequence(kick,time=2*c(0,8)/16)
# snare on 2 and 4
s <- sequence(snare,time=2*c(4,12)/16)
# Mix the 3 tracks
m1 <- mix(list(hh,k,s))
## Not run:
# All calls to function 'tuneR::play' ar wrapped in \dontrun{} since they rely
# on an external audio player to play the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
tuneR::play(m1)
## End(Not run)
# Try with less hihat, more kick
m2 <- mix(list(hh,k,s),volume=c(0.3,1,0.8))
## Not run: tuneR::play(m2)
```

---

noteFrequencyTable *Note-frequency table*

Description

Builds a dataframe containing notes (in scientific pitch notation) and corresponding frequencies.

Usage

```
noteFrequencyTable(minOctave = 0, maxOctave = 8)
```

Arguments

minOctave	integer, smallest (lowest-pitched) octave
maxOctave	integer, largest (highest-pitched) octave

Value

a data frame with 4 columns: note name 1 (written with 'b'), note name 2 (written with '#'),index (in semitones with respect to A4) and frequency (in Hz)

Examples

```
# example code
noteFrequencyTable()
```

oscillator *General oscillator*

Description

Creates a soundSample using a oscillator.

Usage

```
oscillator(type = "sine", freq = 440, duration = 1, phase = 0, rate = 44100)
```

Arguments

type	String, oscillator type, one of 'sine', 'saw', 'square' or 'triangle'. If an unknown string is provided, a sine oscillator will be used.
freq	Numeric, note frequency in Hz
duration	Numeric, note duration in second
phase	Numeric, phase in radians (typically between 0 and 2*pi)
rate	Numeric, sampling rate in Hz

Value

An object of class 'soundSample'.

Examples

```
sam <- oscillator(type='saw',freq=220,duration=0.1)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

oscillator_pattern *Pattern-based oscillator*

Description

Creates a soundSample by repeating a user-provided pattern.

Usage

```
oscillator_pattern(pattern, freq = 440, duration = 1, rate = 44100)
```

Arguments

pattern	Numeric vector, pattern.
freq	Numeric, note frequency in Hz
duration	Numeric, note duration in second
rate	Numeric, sampling rate in Hz

Value

An object of class 'soundSample'.

Examples

```
sam <- oscillator_pattern(pattern=airquality$Ozone,freq=110,duration=0.1)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

---

oscillator_saw *Saw oscillator*

Description

Creates a soundSample using a saw oscillator.

Usage

```
oscillator_saw(freq = 440, duration = 1, phase = 0, rate = 44100)
```

Arguments

freq	Numeric, note frequency in Hz
duration	Numeric, note duration in second
phase	Numeric, phase in radians (typically between 0 and 2*pi)
rate	Numeric, sampling rate in Hz

Value

An object of class 'soundSample'.

Examples

```
sam <- oscillator_saw(freq=220,duration=0.1)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

`oscillator_sine` *Sine oscillator*

Description

Creates a soundSample using a sine oscillator.

Usage

```
oscillator_sine(freq = 440, duration = 1, phase = 0, rate = 44100)
```

Arguments

`freq`	Numeric, note frequency in Hz
`duration`	Numeric, note duration in second
`phase`	Numeric, phase in radians (typically between 0 and 2*pi)
`rate`	Numeric, sampling rate in Hz

Value

An object of class 'soundSample'.

Examples

```
sam <- oscillator_sine(freq=220,duration=0.1)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

oscillator_square *Square oscillator*

Description

Creates a soundSample using a square oscillator.

Usage

```
oscillator_square(freq = 440, duration = 1, phase = 0, rate = 44100)
```

Arguments

freq	Numeric, note frequency in Hz
duration	Numeric, note duration in second
phase	Numeric, phase in radians (typically between 0 and 2*pi)
rate	Numeric, sampling rate in Hz

Value

An object of class 'soundSample'.

Examples

```
sam <- oscillator_square(freq=220,duration=0.1)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

oscillator_triangle *Triangle oscillator*

Description

Creates a soundSample using a triangle oscillator.

Usage

```
oscillator_triangle(freq = 440, duration = 1, phase = 0, rate = 44100)
```

Arguments

freq	Numeric, note frequency in Hz
duration	Numeric, note duration in second
phase	Numeric, phase in radians (typically between 0 and 2*pi)
rate	Numeric, sampling rate in Hz

Value

An object of class 'soundSample'.

Examples

```
sam <- oscillator_triangle(freq=220,duration=0.1)
plot(sam)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to listen to the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
listen(sam)
## End(Not run)
```

---

pitchMapping *Pitch mapping function*

Description

Maps a series of values into pitches of notes

Usage

```
pitchMapping(x, notes)
```

Arguments

x	Numeric vector
notes	character vector, notes onto which values are map (i.e. the musical scakle). Notes should be written in Scientific pitch notation, e.g. c('C4','E4','G4') (see https://en.wikipedia.org/wiki/Scientific_pitch_notation)

Value

a character vector representing the original values transformed into pitches

Examples

```
pitchMapping(x=1:10,notes=c('C4','E4','G4'))
pitchMapping(rnorm(20),notes=c('E3','Gb3','G3','A3','B3','C4','D4'))
```

`play.instrument` *Play an instrument*

Description

Take a sound sample and repeat it following given timeline, volume and pan.

Usage

```
play.instrument(
  inst,
  notes = 1:length(inst),
  time = seq(0, (length(notes) - 1) * 0.25, length.out = length(notes)),
  volume = rep(1, length(notes)),
  pan = rep(0, length(notes)),
  fadein = rep(0.01, length(notes)),
  fadeout = fadein,
  env = NULL,
  nmax = 10 * 10^6
)
```

Arguments

`inst`	Instrument object.
`notes`	String or integer vector, the notes of the instrument to be played, either by name or by index.
`time`	Numeric vector, time (in seconds) at which each note should be played. Should be non-negative, non-decreasing and have same size as notes.
`volume`	Numeric vector, volume between 0 and 1,
`pan`	Numeric vector, pan between -1 (left) and 1 (right) (0 = centered). Same size as notes.
`fadein`	Numeric vector, fade-in duration (in seconds), same size as notes.
`fadeout`	Numeric vector, fade-out duration (in seconds), same size as notes. Use Inf for 'let ring'.
`env`	list of envelope objects, envelope applied to each note.
`nmax`	Integer, max number of values for each channel of the resulting Wave. Default value (10*10^6) roughly corresponds to a 150 Mb stereo wave, ~3 min 45s.

Value

an S4 Wave object (from package tuneR).

Examples

```
# Create an instrument
samples=list(oscillator(freq=110),oscillator(freq=220),oscillator(freq=261.63),
             oscillator(freq=293.66),oscillator(freq=392))
notes=c('A2','A3','C4','D4','G4')
onTheMoon <- instrument(samples,notes)
# Play it
w=play.instrument(onTheMoon)
# View the result
tuneR::plot(w)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to play the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
tuneR::play(w)
## End(Not run)
# Use options
w=play.instrument(onTheMoon,time=c(0,0.2,0.4,0.6,0.8,0.9),
                  notes=c('A2','G4','D4','C4','A3','A2'),
                  volume=seq(0.2,1,length.out=6),pan=c(0,-1,1,-1,1,0),
                  fadeout=c(Inf,0.01,0.01,0.01,Inf,Inf))
# View the result
tuneR::plot(w)
## Not run:
# This line of code is wrapped in \dontrun{} since it relies
# on an external audio player to play the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
tuneR::play(w)
## End(Not run)
```

`plot.envelope` *Plot*

Description

Plot an envelope.

Usage

```
## S3 method for class 'envelope'
plot(x, ...)
```

Arguments

`x`	envelope object.
`...`	further arguments passed to the base plot function.

Value

nothing - plotting function.

Examples

```
# Define envelope
env <- envelope(t=c(0,0.1,0.3,0.8,1),v=c(0,1,0.4,0.4,0))
# plot it
plot(env)
```

`plot.soundSample` *Plot a sound sample*

Description

Plot a sound sample. Uses plotly to add zooming capability.

Usage

```
## S3 method for class 'soundSample'
plot(x, ...)
```

Arguments

`x`	sound sample object.
`...`	further arguments passed to tuneR plotting function.

Value

nothing - plotting function.

Examples

```
# Define sound sample
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)+0.1*rnorm(44100)) # 1-second noisy A
# plot it
plot(sam)
```

`read.soundSample` *Read a sound sample*

Description

Read a sound sample from a .mp3 or .wav file.

Usage

```
read.soundSample(file, ...)
```

Arguments

file	string, file with extension .wav or.mp3
...	additional arguments passed to function tuneR::readWave

Value

An object of class 'soundSample'.

Examples

```
sam=try(read.soundSample(file='vignettes/07027201.mp3'))
```

---

rescale *Rescale function*

Description

Rescale a series between two bounds

Usage

```
rescale(x, low = 0, high = 1)
```

Arguments

x	Numeric vector
low	Numeric, lower bound
high	Numeric, higher bound

Value

a rescaled numeric vector

Examples

```
# example code
rescale(1:10)
rescale(rnorm(10), 100, 101)
```

ride *Ride sample*

Description

A ride cymbal sound sample object

Usage

```
ride
```

Format

An object of class `soundSample` of length 4.

sequence *Sequence a sound sample*

Description

Take a sound sample and repeat it following given timeline, volume and pan.

Usage

```
sequence(
  sample,
  time,
  letRing = TRUE,
  volume = rep(1, NROW(time)),
  pan = rep(0, NROW(time)),
  nmax = 10 * 10^6
)
```

Arguments

`sample`	Sound sample object.
`time`	Numeric vector, time (in seconds) at which sample should be repeated
`letRing`	Logical. If TRUE overlapping samples are added; if FALSE, a new sample stops the previous one (=> beware of the click!))
`volume`	Numeric vector, volume between 0 and 1.
`pan`	Numeric vector, pan between -1 (left) and 1 (right) (0 = centered).
`nmax`	Integer, max number of values for each channel of the resulting Wave. Default value (10*10^6) roughly corresponds to a 150 Mb stereo wave, ~3 min 45s.

Value

an S4 Wave object (from package tuneR).

Examples

```
# EXAMPLE 1
# Define a sound sample
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)+0.1*rnorm(44100)) # 1-second noisy A
# Sequence it
s <- sequence(sam,time=c(0,0.5,0.75),letRing=FALSE,volume=c(0.4,1,1),pan=c(-1,0,1))
# View the result
tuneR::plot(s)
## Not run:
# All calls to function 'tuneR::play' are wrapped in \dontrun{} since
# they rely on an external audio player to play the audio sample.
# See ?tuneR::setWavPlayer for setting a default player.
tuneR::play(s)
## End(Not run)
#' EXAMPLE 2 - make it funkyer
# 2-second sequence based on hi-hat sample
s <- sequence(hiHat,time=seq(0,2,,16),volume=rep(c(1,rep(0.5,3)),4))
# View the result
tuneR::plot(s)
## Not run: tuneR::play(s)
```

`shiftPitch` *Pitch shifter*

Description

Shift the pitch of a sound sample by n semitones. Note that the duration of the resulting sample is not the same as that of the original.

Usage

```
shiftPitch(sample, n)
```

Arguments

`sample`	Sound sample object.
`n`	numeric, number of semitones.

Value

A sound sample object.

Examples

```
# Define a A sound sample and get a D by adding 5 semitones
A <- soundSample(sin(2*pi*seq(0,0.5,1/44100)*220)) # 0.5-second A (220 Hz)
D <- shiftPitch(A,5)
```

`snare` *Snare sample*

Description

A snare sound sample object

Usage

```
snare
```

Format

An object of class `soundSample` of length 4.

`snare2` *Snare sample 2*

Description

A snare sound sample object

Usage

```
snare2
```

Format

An object of class `soundSample` of length 4.

Source

http://www.archive.org/details/OpenPathMusic44V1

`sonifyStripes` *Climate stripes sonification*

Description

Sonification of climate stripes data, or more generally, of a time series of values. A smoothed version of the time series is computed by moving average, then sonification proceeds as follows:

- Backtrack is a standard house-like tune, including a four-on-the-floor kick+hi-hat pattern on the drum, a bass following the drum kick, and 3 chords played by a synthesizer
- The smoothed time series controls the master volume and the amount of 'distortion' in the synthesizer's sound
- Large anomalies below / above the smoothed series trigger percussion sounds (by default a snare and a hand clap) that are panned full left (negative anomalies) and full right (positive anomalies)

Usage

```
sonifyStripes(
  values = sequenceR::globalT$Anomaly,
  bpm = 135,
  minVol = 0.1,
  nma = 10,
  pClap = 0.15,
  synthVar = 0.5,
  kick = sequenceR::mini909$bass,
  hihat = sequenceR::mini909$hihat,
  openHihat = sequenceR::mini909$hihat_o,
  posPercussion = sequenceR::mini909$snare,
  negPercussion = sequenceR::mini909$clap,
  bassNote = "E1",
  chord1 = c("E2", "E3", "G3", "D4", "Gb4"),
  chord2 = c("E2", "D3", "Gb3", "A3", "E4"),
  chord3 = c("E2", "B2", "Gb3", "G3", "D4"),
  videoFile = NULL,
  videoResFactor = 1
)
```

Arguments

`values`	Numeric vector, values to sonify. Default is global temperature anomalies over the period 1850-2021
`bpm`	Numeric > 0, tempo in beat per minute
`minVol`	Numeric >= 0, minimum volume reached when smoothed series is minimum

nma	Numeric >=0 , number of moving average steps on each side of the current value (i.e. moving average window is 2*nma+1 when possible, nma+1 on the series' edges)
pClap	Numeric in (0,0.5). "Large" anomalies triggering claps/snare are defined as anomalies below (resp. above) the pClap (resp. (1-pClap))-quantile of anomalies.
synthVar	Numeric >= 0 , controls the variability of the synthesizer sound. When zero, the synthesizer sound does not change. Large values induce more variability in the synthesizer sound.
kick	soundSample, sound sample used to play the kick drum.
hihat	soundSample, sound sample used to play the closed hi-hat.
openHihat	soundSample, sound sample used to play the open hi-hat.
posPercussion	soundSample, sound sample used to play the positive-anomaly percussion.
negPercussion	soundSample, sound sample used to play the negative-anomaly percussion.
bassNote	string, bass note (in scientific pitch notation).
chord1	string vector, first chord played by synthesizer.
chord2	string vector, second chord played by synthesizer.
chord3	string vector, third chord played by synthesizer.
videoFile	file path, full path to video file. When NULL, video is not created.
videoResFactor	Numeric > 0 , video resolution, 2 recommended for good-quality video.

Value

A list with the following components:

- mix, tuneR::Wave object, the final mix of the sonification.
- dat, data frame with 4 columns: time step, raw value, smoothed value, anomaly
- quantiles, numeric vector of size 2, the quantiles defining large negative/positive anomalies
- waves, list of tuneR::Wave object, individual waves for each instrument in case you wish to mix them in your own way.

Examples

```
w <- sonifyStripes()
```

soundSample *Sound sample constructor.*

Description

Creates a new instance of a 'soundSample' object. A sound sample can be viewed as a minimalistic version of an "audio wave" object (see package tuneR for instance). It is necessarily mono and the wave time series is normalized between -1 and 1.

Usage

```
soundSample(wave, rate = 44100)
```

Arguments

wave	Numeric vector, wave time series
rate	Numeric, sampling rate (default 44100 Hz)

Value

An object of class 'soundSample'.

Examples

```
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)) # 1-second A (440 Hz)
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)+0.1*rnorm(44100)) # 1-second noisy A
```

timeVector *timeVector function*

Description

Compute the time vector starting from 0 associated with a duration and a sampling rate

Usage

```
timeVector(duration = 1, rate = 44100)
```

Arguments

duration	Numeric
rate	Numeric

Value

a numeric vector

WaggaWagga *Wagga-Wagga dataset*

Description

Times series of monthly temperatures and precipitations recorded at Wagga-Wagga, New South Wales, Australia, 1940-2018

Usage

```
WaggaWagga
```

Format

An object of class `data.frame` with 79 rows and 3 columns.

Source

http://www.bom.gov.au/cgi-bin/climate/hqsites/site_data.cgi?period=annual&variable=meanT&station=072150

http://www.bom.gov.au/cgi-bin/climate/hqsites/site_data.cgi?period=annual&variable=rain&station=072150

write.instrument *Write an instrument to file*

Description

Write each sound sample of the instrument as a separate .wav or .mp3 file.

Usage

```
write.instrument(inst, dir = tempdir(), fmt = "wav")
```

Arguments

`inst`	Instrument object.
`dir`	String, directory where files should be written.
`fmt`	String, 'wav' or 'mp3'.

Value

nothing - writing function.

Examples

```
# Create an instrument
drumset <- instrument(samples=list(kick,snare,hiHat),notes=c('boom','tat','cheet'))
# Write to files (one per element)
write.instrument(drumset)
```

`write.soundSample` *Write a sound sample*

Description

Write a sound sample in .wav or .mp3 format.

Usage

```
write.soundSample(x, file)
```

Arguments

`x`	sound sample object.
`file`	string, destination file. Default file format is .wav. If file extension is .mp3, conversion to mp3 is attempted using ffmpeg, which hence needs to be available (see https://ffmpeg.org/).

Value

nothing - writing function.

Examples

```
sam <- soundSample(sin(2*pi*seq(0,1,,44100)*440)) # 1-second A (440 Hz)
write.soundSample(sam,file=tempfile())
```

Index