Package 'rPraat'

August 2, 2018

Type Package

Title Interface to Praat

Version 1.1.1

Encoding UTF-8

Maintainer Tomas Boril <borilt@gmail.com>

Description Read, write and manipulate 'Praat' TextGrid, PitchTier, Pitch and Intensity files <http://www.fon.hum.uva.nl/praat/>.

URL https://github.com/bbTomas/rPraat/

BugReports https://github.com/bbTomas/rPraat/issues

License MIT + file LICENSE

LazyData TRUE

Depends R (>= 3.4.0)

Imports graphics (>= 3.1.0),
dplyr (>= 0.7.1),
stringr (>= 1.2.0),
readr(>= 1.1.1),
dygraphs (>= 1.1.1.4),

RoxygenNote 6.1.0

Suggests testthat

R topics documented:

`col.read` *col.read*

Description

Loads Collection from Praat in Text or Short text format. Collection may contain combination of TextGrids, PitchTiers, Pitch objects, and IntensityTiers.

Usage

```
col.read(fileName, encoding = "UTF-8")
```

Arguments

`fileName`	Input file name
`encoding`	File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

Collection object

See Also

`tg.read`, `pt.read`, `pitch.read`, `it.read`

Examples

```
## Not run:
coll <- col.read("coll_text.Collection")
length(coll)  # number of objects in collection
class(coll[[1]])["type"]  # 1st object type
class(coll[[1]])["name"]  # 1st object name
it <- coll[[1]]  # 1st object
it.plot(it)

class(coll[[2]])["type"]  # 2nd object type
class(coll[[2]])["name"]  # 2nd object name
tg <- coll[[2]]  # 2nd object
tg.plot(tg)
```

```
length(tg)  # number of tiers in TextGrid
tg$word$label

class(coll[[3]])["type"]  # 3rd object type
class(coll[[3]])["name"]  # 3rd object type
pitch <- coll[[3]]  # 3rd object
names(pitch)
pitch$nx  # number of frames
pitch$t[4]        # time instance of the 4th frame
pitch$frame[[4]]  # 4th frame: pitch candidates
pitch$frame[[4]]$frequency[2]
pitch$frame[[4]]$strength[2]

class(coll[[4]])["type"]  # 4th object type
class(coll[[4]])["name"]  # 4th object name
pt <- coll[[4]]  # 2nd object
pt.plot(pt)

## End(Not run)
```

`detectEncoding` *detectEncoding*

Description

Detects unicode encoding of Praat text files

Usage

```
detectEncoding(fileName)
```

Arguments

`fileName`	Input file name

Value

detected encoding of the text input file

Examples

```
## Not run:
detectEncoding("demo/H.TextGrid")
detectEncoding("demo/H_UTF16.TextGrid")

## End(Not run)
```

`ifft` *ifft*

Description

Inverse Fast Fourier Transform (discrete FT), Matlab-like behavior.

Usage

```
ifft(sig)
```

Arguments

`sig`	input vector

Details

This is really the inverse of the fft function, so ifft(fft(x)) == x.

Value

output vector of the same length as the input vector

See Also

`fft`, `Re`, `Im`, `Mod`, `Conj`

Examples

```
ifft(fft(1:5))
```

`isInt` *isInt*

Description

Returns TRUE / FALSE whether it is exactly 1 integer number (in fact, the class can be numeric but the number must be integer), non-missing

Usage

```
isInt(num)
```

Arguments

`num`	variable to be tested

Value

TRUE / FALSE

See Also

isNum, isLogical, isString

Examples

```
isInt(2)
isInt(2L)
isInt(-2)
isInt(-2L)
isInt(2.1)
isInt(-2.1)
isInt(1:5)
isInt(NA_integer_)
isInt(integer(0))
```

`isLogical` *isLogical*

Description

Returns TRUE / FALSE whether it is exactly 1 logical value, non-missing

Usage

```
isLogical(logical)
```

Arguments

`logical`	variable to be tested

Value

TRUE / FALSE

See Also

isNum, isInt, isString

Examples

```
isLogical(TRUE)
isLogical(FALSE)
isLogical(1)
isLogical(0)
isLogical(2)
isLogical(NA)
isLogical(NaN)
isLogical(logical(0))
```

isNum *isNum*

Description

Returns TRUE / FALSE whether it is exactly 1 number (numeric or integer vector of length 1, non-missing)

Usage

```
isNum(num)
```

Arguments

num	variable to be tested

Value

TRUE / FALSE

See Also

isInt, isLogical, isString

Examples

```
isNum(2)
isNum(2L)
isNum(-2)
isNum(-2L)
isNum(2.1)
isNum(-2.1)
isNum(1:5)
isNum(NA_real_)
isNum(numeric(0))
```

isString *isString*

Description

Returns TRUE / FALSE whether it is exactly 1 character string (character vector of length 1, non-missing)

Usage

```
isString(string)
```

Arguments

string	variable to be tested

Value

TRUE / FALSE

See Also

isInt, isNum, isLogical

Examples

```
isString("hello")
isString(2)
isString(c("hello", "world"))
isString(NA_character_)
```

`it.cut` *it.cut*

Description

Cut the specified interval from the IntensityTier and preserve time

Usage

```
it.cut(it, tStart = -Inf, tEnd = Inf)
```

Arguments

it	IntensityTier object
tStart	beginning time of interval to be cut (default -Inf = cut from the tMin of the IntensityTier)
tEnd	final time of interval to be cut (default Inf = cut to the tMax of the IntensityTier)

Value

IntensityTier object

See Also

it.cut0, it.read, it.plot, it.interpolate, it.legendre, it.legendreSynth, it.legendreDemo

Examples

```
it <- it.sample()
it2 <-   it.cut(it,  tStart = 0.3)
it2_0 <- it.cut0(it, tStart = 0.3)
it3 <-   it.cut(it,  tStart = 0.2, tEnd = 0.3)
it3_0 <- it.cut0(it, tStart = 0.2, tEnd = 0.3)
it4 <-   it.cut(it,  tEnd = 0.3)
it4_0 <- it.cut0(it, tEnd = 0.3)
it5 <-   it.cut(it,  tStart = -1, tEnd = 1)
it5_0 <- it.cut0(it, tStart = -1, tEnd = 1)
## Not run:
```

```
it.plot(it)
it.plot(it2)
it.plot(it2_0)
it.plot(it3)
it.plot(it3_0)
it.plot(it4)
it.plot(it4_0)
it.plot(it5)
it.plot(it5_0)

## End(Not run)
```

`it.cut0` *it.cut0*

Description

Cut the specified interval from the IntensityTier and shift time so that the new tmin = 0

Usage

```
it.cut0(it, tStart = -Inf, tEnd = Inf)
```

Arguments

`it`	IntensityTier object
`tStart`	beginning time of interval to be cut (default -Inf = cut from the tMin of the IntensityTier)
`tEnd`	final time of interval to be cut (default Inf = cut to the tMax of the IntensityTier)

Value

IntensityTier object

See Also

`it.cut`, `it.read`, `it.plot`, `it.interpolate`, `it.legendre`, `it.legendreSynth`, `it.legendreDemo`

Examples

```
it <- it.sample()
it2 <-   it.cut(it,  tStart = 0.3)
it2_0 <- it.cut0(it, tStart = 0.3)
it3 <-   it.cut(it,  tStart = 0.2, tEnd = 0.3)
it3_0 <- it.cut0(it, tStart = 0.2, tEnd = 0.3)
it4 <-   it.cut(it,  tEnd = 0.3)
it4_0 <- it.cut0(it, tEnd = 0.3)
it5 <-   it.cut(it,  tStart = -1, tEnd = 1)
it5_0 <- it.cut0(it, tStart = -1, tEnd = 1)
## Not run:
it.plot(it)
it.plot(it2)
it.plot(it2_0)
```

```
it.plot(it3)
it.plot(it3_0)
it.plot(it4)
it.plot(it4_0)
it.plot(it5)
it.plot(it5_0)

## End(Not run)
```

`it.interpolate` *it.interpolate*

Description

Interpolates IntensityTier contour in given time instances.

Usage

```
it.interpolate(it, t)
```

Arguments

`it`	IntensityTier object
`t`	vector of time instances of interest

Details

a) If t < min(it$t) (or t > max(it$t)), returns the first (or the last) value of it$i. b) If t is existing point in it$t, returns the respective it$f. c) If t is Between two existing points, returns linear interpolation of these two points.

Value

IntensityTier object

See Also

`it.read`, `it.write`, `it.plot`, `it.cut`, `it.cut0`, `it.legendre`

Examples

```
it <- it.sample()
it2 <- it.interpolate(it, seq(it$t[1], it$t[length(it$t)], by = 0.001))
## Not run:
it.plot(it)
it.plot(it2)

## End(Not run)
```

`it.legendre` *it.legendre*

Description

Interpolate the IntensityTier in 'npoints' equidistant points and approximate it by Legendre polynomials

Usage

```
it.legendre(it, npoints = 1000, npolynomials = 4)
```

Arguments

`it`	IntensityTier object
`npoints`	Number of points of IntensityTier interpolation
`npolynomials`	Number of polynomials to be used for Legendre modelling

Value

Vector of Legendre polynomials coefficients

See Also

`it.legendreSynth`, `it.legendreDemo`, `it.cut`, `it.cut0`, `it.read`, `it.plot`, `it.interpolate`

Examples

```
it <- it.sample()
it <- it.cut(it, tStart = 0.2, tEnd = 0.4)  # cut IntensityTier and preserve time
c <- it.legendre(it)
print(c)
leg <- it.legendreSynth(c)
itLeg <- it
itLeg$t <- seq(itLeg$tmin, itLeg$tmax, length.out = length(leg))
itLeg$i <- leg
## Not run:
plot(it$t, it$i, xlab = "Time (sec)", ylab = "Intensity (dB)")
lines(itLeg$t, itLeg$i, col = "blue")

## End(Not run)
```

it.legendreDemo *it.legendreDemo*

Description

Plots first four Legendre polynomials

Usage

```
it.legendreDemo()
```

See Also

it.legendre, it.legendreSynth, it.read, it.plot, it.interpolate

Examples

```
## Not run:
it.legendreDemo()

## End(Not run)
```

it.legendreSynth *it.legendreSynth*

Description

Synthetize the contour from vector of Legendre polynomials 'c' in 'npoints' equidistant points

Usage

```
it.legendreSynth(c, npoints = 1000)
```

Arguments

c	Vector of Legendre polynomials coefficients
npoints	Number of points of IntensityTier interpolation

Value

Vector of values of synthetized contour

See Also

it.legendre, it.legendreDemo, it.read, it.plot, it.interpolate

Examples

```
it <- it.sample()
it <- it.cut(it, tStart = 0.2, tEnd = 0.4)  # cut IntensityTier and preserve time
c <- it.legendre(it)
print(c)
leg <- it.legendreSynth(c)
itLeg <- it
itLeg$t <- seq(itLeg$tmin, itLeg$tmax, length.out = length(leg))
itLeg$i <- leg
## Not run:
plot(it$t, it$i, xlab = "Time (sec)", ylab = "Intensity (dB)")
lines(itLeg$t, itLeg$i, col = "blue")

## End(Not run)
```

`it.plot` *it.plot*

Description

Plots interactive IntensityTier using dygraphs package.

Usage

```
it.plot(it, group = "")
```

Arguments

`it`	IntensityTier object
`group`	[optional] character string, name of group for dygraphs synchronization

See Also

`it.read`, `tg.plot`, `it.cut`, `it.cut0`, `it.interpolate`, `it.write`

Examples

```
## Not run:
it <- it.sample()
it.plot(it)

## End(Not run)
```

it.read *it.read*

Description

Reads IntensityTier from Praat. Supported formats: text file, short text file.

Usage

```
it.read(fileNameIntensityTier, encoding = "UTF-8")
```

Arguments

fileNameIntensityTier	file name of IntensityTier
encoding	File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

IntensityTier object

See Also

it.write, it.plot, it.cut, it.cut0, it.interpolate, tg.read, pt.read, pitch.read, col.read

Examples

```
## Not run:
it <- it.read("demo/maminka.IntensityTier")
it.plot(it)

## End(Not run)
```

it.sample *it.sample*

Description

Returns sample IntensityTier.

Usage

```
it.sample()
```

Value

IntensityTier

See Also

it.plot

Examples

```
it <- it.sample()
it.plot(it)
```

`it.write` *it.write*

Description

Saves IntensityTier to file (in UTF-8 encoding). it is list with at least $t and $i vectors (of the same length). If there are no $tmin and $tmax values, there are set as min and max of $t vector.

Usage

```
it.write(it, fileNameIntensityTier, format = "short")
```

Arguments

`it`	IntensityTier object
`fileNameIntensityTier`	file name to be created
`format`	Output file format ("short" (short text format - default), "text" (a.k.a. full text format))

See Also

`it.read`, `tg.write`, `it.interpolate`

Examples

```
## Not run:
it <- it.sample()
it.plot(pt)
it.write(it, "demo/intensity.IntensityTier")

## End(Not run)
```

`pitch.read` *pitch.read*

Description

Reads Pitch object from Praat. Supported formats: text file, short text file.

Usage

```
pitch.read(fileNamePitch, encoding = "UTF-8")
```

Arguments

fileNamePitch	file name of Pitch object
encoding	File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

A Pitch object represents periodicity candidates as a function of time.

[ref: Praat help, http://www.fon.hum.uva.nl/praat/manual/Pitch.html]

p$xmin ... start time (seconds)

p$xmax ... end time (seconds)

p$nx ... number of frames

p$dx ... time step = frame duration (seconds)

p$x1 ... time associated with the first frame (seconds)

p$t ... vector of time instances associated with all frames

p$ceiling ... a frequency above which a candidate is considered voiceless (Hz)

p$maxnCandidates ... maximum number of candidates in frame

p$frame[[1]] to p$frame[[p$nx]] ... frames

p$frame[[1]]$intensity ... intensity of the frame

p$frame[[1]]$nCandidates ... actual number of candidates in this frame

p$frame[[1]]$frequency ... vector of candidates' frequency (in Hz)

(for a voiced candidate), or 0 (for an unvoiced candidate)

p$frame[[1]]$strength ... vector of degrees of periodicity of candidates (between 0 and 1)

See Also

pt.read, tg.read, it.read, col.read

Examples

```
## Not run:
p <- pitch.read('demo/sound.Pitch')
names(p)
p$nx
p$t[4]        # time instance of the 4th frame
p$frame[[4]]  # 4th frame: pitch candidates
p$frame[[4]]$frequency[2]
p$frame[[4]]$strength[2]

## End(Not run)
```

pt.cut *pt.cut*

Description

Cut the specified interval from the PitchTier and preserve time

Usage

```
pt.cut(pt, tStart = -Inf, tEnd = Inf)
```

Arguments

pt	PitchTier object
tStart	beginning time of interval to be cut (default -Inf = cut from the tMin of the PitchTier)
tEnd	final time of interval to be cut (default Inf = cut to the tMax of the PitchTier)

Value

PitchTier object

See Also

pt.cut0, pt.read, pt.plot, pt.Hz2ST, pt.interpolate, pt.legendre, pt.legendreSynth, pt.legendreDemo

Examples

```
pt <- pt.sample()
pt2 <-   pt.cut(pt,  tStart = 3)
pt2_0 <- pt.cut0(pt, tStart = 3)
pt3 <-   pt.cut(pt,  tStart = 2, tEnd = 3)
pt3_0 <- pt.cut0(pt, tStart = 2, tEnd = 3)
pt4 <-   pt.cut(pt,  tEnd = 1)
pt4_0 <- pt.cut0(pt, tEnd = 1)
pt5 <-   pt.cut(pt,  tStart = -1, tEnd = 1)
pt5_0 <- pt.cut0(pt, tStart = -1, tEnd = 1)
## Not run:
pt.plot(pt)
pt.plot(pt2)
pt.plot(pt2_0)
pt.plot(pt3)
pt.plot(pt3_0)
pt.plot(pt4)
pt.plot(pt4_0)
pt.plot(pt5)
pt.plot(pt5_0)

## End(Not run)
```

pt.cut0 *pt.cut0*

Description

Cut the specified interval from the PitchTier and shift time so that the new tmin = 0

Usage

```
pt.cut0(pt, tStart = -Inf, tEnd = Inf)
```

Arguments

pt	PitchTier object
tStart	beginning time of interval to be cut (default -Inf = cut from the tMin of the PitchTier)
tEnd	final time of interval to be cut (default Inf = cut to the tMax of the PitchTier)

Value

PitchTier object

See Also

pt.cut, pt.read, pt.plot, pt.Hz2ST, pt.interpolate, pt.legendre, pt.legendreSynth, pt.legendreDemo

Examples

```
pt <- pt.sample()
pt2 <-   pt.cut(pt,  tStart = 3)
pt2_0 <- pt.cut0(pt, tStart = 3)
pt3 <-   pt.cut(pt,  tStart = 2, tEnd = 3)
pt3_0 <- pt.cut0(pt, tStart = 2, tEnd = 3)
pt4 <-   pt.cut(pt,  tEnd = 1)
pt4_0 <- pt.cut0(pt, tEnd = 1)
pt5 <-   pt.cut(pt,  tStart = -1, tEnd = 1)
pt5_0 <- pt.cut0(pt, tStart = -1, tEnd = 1)
## Not run:
pt.plot(pt)
pt.plot(pt2)
pt.plot(pt2_0)
pt.plot(pt3)
pt.plot(pt3_0)
pt.plot(pt4)
pt.plot(pt4_0)
pt.plot(pt5)
pt.plot(pt5_0)

## End(Not run)
```

pt.Hz2ST *pt.Hz2ST*

Description

Converts Hz to Semitones with given reference (default 0 ST = 100 Hz).

Usage

```
pt.Hz2ST(pt, ref = 100)
```

Arguments

pt	PitchTier object
ref	reference value (in Hz) for 0 ST. Default: 100 Hz.

Value

PitchTier object

See Also

pt.read, pt.write, pt.plot, pt.interpolate, pt.cut, pt.cut0

Examples

```
pt <- pt.sample()
pt2 <- pt.Hz2ST(pt, ref = 200)
## Not run:
pt.plot(pt)  %>% dygraphs::dyAxis("y", label = "Frequency (Hz)")
pt.plot(pt2) %>% dygraphs::dyAxis("y", label = "Frequency (ST re 200 Hz)")

## End(Not run)
```

pt.interpolate *pt.interpolate*

Description

Interpolates PitchTier contour in given time instances.

Usage

```
pt.interpolate(pt, t)
```

Arguments

pt	PitchTier object
t	vector of time instances of interest

Details

a) If t < min(pt$t) (or t > max(pt$t)), returns the first (or the last) value of pt$f. b) If t is existing point in pt$t, returns the respective pt$f. c) If t is Between two existing points, returns linear interpolation of these two points.

Value

PitchTier object

See Also

pt.read, pt.write, pt.plot, pt.Hz2ST, pt.cut, pt.cut0, pt.legendre

Examples

```
pt <- pt.sample()
pt <- pt.Hz2ST(pt, ref = 100)  # conversion of Hz to Semitones, reference 0 ST = 100 Hz.
pt2 <- pt.interpolate(pt, seq(pt$t[1], pt$t[length(pt$t)], by = 0.001))
## Not run:
pt.plot(pt)
pt.plot(pt2)

## End(Not run)
```

---

pt.legendre *pt.legendre*

Description

Interpolate the PitchTier in 'npoints' equidistant points and approximate it by Legendre polynomials

Usage

```
pt.legendre(pt, npoints = 1000, npolynomials = 4)
```

Arguments

pt	PitchTier object
npoints	Number of points of PitchTier interpolation
npolynomials	Number of polynomials to be used for Legendre modelling

Value

Vector of Legendre polynomials coefficients

See Also

pt.legendreSynth, pt.legendreDemo, pt.cut, pt.cut0, pt.read, pt.plot, pt.Hz2ST, pt.interpolate

Examples

```
pt <- pt.sample()
pt <- pt.Hz2ST(pt)
pt <- pt.cut(pt, tStart = 3)  # cut PitchTier from t = 3 sec and preserve time
c <- pt.legendre(pt)
print(c)
leg <- pt.legendreSynth(c)
ptLeg <- pt
ptLeg$t <- seq(ptLeg$tmin, ptLeg$tmax, length.out = length(leg))
ptLeg$f <- leg
## Not run:
plot(pt$t, pt$f, xlab = "Time (sec)", ylab = "F0 (ST re 100 Hz)")
lines(ptLeg$t, ptLeg$f, col = "blue")

## End(Not run)
```

---

pt.legendreDemo *pt.legendreDemo*

Description

Plots first four Legendre polynomials

Usage

```
pt.legendreDemo()
```

See Also

pt.legendre, pt.legendreSynth, pt.read, pt.plot, pt.Hz2ST, pt.interpolate

Examples

```
## Not run:
pt.legendreDemo()

## End(Not run)
```

---

pt.legendreSynth *pt.legendreSynth*

Description

Synthetize the contour from vector of Legendre polynomials 'c' in 'npoints' equidistant points

Usage

```
pt.legendreSynth(c, npoints = 1000)
```

Arguments

c	Vector of Legendre polynomials coefficients
npoints	Number of points of PitchTier interpolation

Value

Vector of values of synthetized contour

See Also

pt.legendre, pt.legendreDemo, pt.read, pt.plot, pt.Hz2ST, pt.interpolate

Examples

```
pt <- pt.sample()
pt <- pt.Hz2ST(pt)
pt <- pt.cut(pt, tStart = 3)  # cut PitchTier from t = 3 sec and preserve time
c <- pt.legendre(pt)
print(c)
leg <- pt.legendreSynth(c)
ptLeg <- pt
ptLeg$t <- seq(ptLeg$tmin, ptLeg$tmax, length.out = length(leg))
ptLeg$f <- leg
## Not run:
plot(pt$t, pt$f, xlab = "Time (sec)", ylab = "F0 (ST re 100 Hz)")
lines(ptLeg$t, ptLeg$f, col = "blue")

## End(Not run)
```

---

pt.plot *pt.plot*

Description

Plots interactive PitchTier using dygraphs package.

Usage

```
pt.plot(pt, group = "")
```

Arguments

pt	PitchTier object
group	[optional] character string, name of group for dygraphs synchronization

See Also

pt.read, tg.plot, pt.Hz2ST, pt.cut, pt.cut0, pt.interpolate, pt.write

Examples

```
## Not run:
pt <- pt.sample()
pt.plot(pt)

## End(Not run)
```

`pt.read` *pt.read*

Description

Reads PitchTier from Praat. Supported formats: text file, short text file, spreadsheet, headerless spreadsheet (headerless not recommended, it does not contain tmin and tmax info).

Usage

```
pt.read(fileNamePitchTier, encoding = "UTF-8")
```

Arguments

`fileNamePitchTier`	file name of PitchTier
`encoding`	File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

PitchTier object

See Also

`pt.write`, `pt.plot`, `pt.Hz2ST`, `pt.cut`, `pt.cut0`, `pt.interpolate`, `pt.legendre`, `tg.read`, `pitch.read`, `it.read`, `col.read`

Examples

```
## Not run:
pt <- pt.read("demo/H.PitchTier")
pt.plot(pt)

## End(Not run)
```

pt.sample *pt.sample*

Description

Returns sample PitchTier.

Usage

```
pt.sample()
```

Value

PitchTier

See Also

pt.plot

Examples

```
pt <- pt.sample()
pt.plot(pt)
```

pt.write *pt.write*

Description

Saves PitchTier to file (in UTF-8 encoding). pt is list with at least $t and $f vectors (of the same length). If there are no $tmin and $tmax values, there are set as min and max of $t vector.

Usage

```
pt.write(pt, fileNamePitchTier, format = "spreadsheet")
```

Arguments

pt	PitchTier object
fileNamePitchTier	file name to be created
format	Output file format ("short" (short text format), "text" (a.k.a. full text format), "spreadsheet" (default), "headerless" (not recommended, it does not contain tmin and tmax info))

See Also

pt.read, tg.write, pt.Hz2ST, pt.interpolate

Examples

```
## Not run:
pt <- pt.sample()
pt <- pt.Hz2ST(pt)    #  conversion of Hz to Semitones, reference 0 ST = 100 Hz.
pt.plot(pt)
pt.write(pt, "demo/H_st.PitchTier")

## End(Not run)
```

---

round2 *round2*

Description

Rounds a number to the specified order. Round half away from zero (this is the difference from built-in round function.)

Usage

```
round2(x, order = 0)
```

Arguments

x	number to be rounded
order	0 (default) = units, -1 = 0.1, +1 = 10

Value

rounded number to the specified order

See Also

round, trunc, ceiling, floor

Examples

```
round2(23.5)   # = 24, compare: round(23.5) = 24
round2(23.4)   # = 23
round2(24.5)   # = 25, compare: round(24.5) = 24
round2(-23.5)   # = -24, compare: round(-23.5) = -24
round2(-23.4)   # = -23
round2(-24.5)   # = -25, compare: round(-24.5) = -24
round2(123.456, -1)   # 123.5
round2(123.456, -2)   # 123.46
round2(123.456, 1)  # 120
round2(123.456, 2)  # 100
round2(123.456, 3)  # 0
round2(-123.456, -1)   # -123.5
round2(-123.456, -2)   # -123.46
round2(-123.456, 1)  # -120
round2(-123.456, 2)  # -100
round2(-123.456, 3)  # 0
```

seqM *seqM*

Description

Matlab-like behaviour of colon operator or linspace for creating sequences, for-loop friendly.

Usage

```
seqM(from = NA, to = NA, by = NA, length.out = NA)
```

Arguments

from	starting value of the sequence (the first number)
to	end value of the sequence (the last number or the boundary number)
by	increment of the sequence (if specified, do not use the length.out parameter). If both by and length.out are not specified, then by = +1.
length.out	desired length of the sequence (if specified, do not use the by parameter)

Details

Like seq() but with Matlab-like behavior ([: operator] with by or [linspace] with length.out).

If I create a for-loop, I would like to get an empty vector for 3:1 (I want a default step +1) and also an empty vector for seq(3, 1, by = 1) (not an error). This is solved by this seqM function.

Value

returns a vector of type "integer" or "double"

Comparison

R: seqM		Matlab		R: seq
seqM(1, 3)	[1] 1 2 3	1:3	the same	the same
seqM(1, 3, by=.8)	[1] 1.0 1.8 2.6	1:.8:3	the same	the same
seqM(1, 3, by=5)	[1] 1	1:5:3	the same	the same
seqM(3, 1)	integer(0)	3:1	the same	[1] 3 2 1
seqM(3, 1, by=+1)	integer(0)	3:1:1	the same	Error: wrong 'by'
seqM(3, 1, by=-1)	[1] 3 2 1	3:-1:1	the same	the same
seqM(3, 1, by=-3)	[1] 3	3:-3:1	the same	the same
seqM(1, 3, len=5)	[1] 1.0 1.5 2.0 2.5 3.0	linspace(1,3,5)	the same	the same
seqM(1, 3, len=3)	[1] 1 2 3	linspace(1,3,3)	the same	the same
seqM(1, 3, len=2)	[1] 1 3	linspace(1,3,2)	the same	the same
seqM(1, 3, len=1)	[1] 3	linspace(1,3,1)	the same	[1] 1
seqM(1, 3, len=0)	integer(0) + warning	linspace(1,3,0)	the same without warning	the same without warning
seqM(3, 1, len=3)	[1] 3 2 1	linspace(3,1,3)	the same	the same

See Also

round2, isNum, isInt, ifft.

Examples

```
seqM(1, 3)
seqM(1, 3, by=.8)
seqM(1, 3, by=5)
seqM(3, 1)
seqM(3, 1, by=+1)
seqM(3, 1, by=-1)
seqM(3, 1, by=-3)
seqM(1, 3, len=5)
seqM(1, 3, len=3)
seqM(1, 3, len=2)
seqM(1, 3, len=1)
seqM(1, 3, len=0)
seqM(3, 1, len=3)
```

---

strTrim *strTrim*

Description

Trim leading and trailing whitespace in character string.

Usage

```
strTrim(string)
```

Arguments

string	character string

Details

Like str_trim() in stringr package or trimws() in R3.2.0 but way faster.

Source: Hadley Wickham comment at http://stackoverflow.com/questions/2261079/how-to-trim-leading-and-trailing-whitespace-in-r

Value

returns a character string with removed leading and trailing whitespace characters.

See Also

isString for testing whether it is 1 character vector, str_contains for finding string in string without regexp, str_find for all indices without regexp, str_find1 for the first index withoud regexp.

Examples

```
strTrim("      Hello World!    ")
```

`str_contains` *str_contains*

Description

Find string in another string (without regular expressions), returns TRUE / FALSE.

Usage

```
str_contains(string, patternNoRegex)
```

Arguments

`string`	string in which we try to find something
`patternNoRegex`	string we want to find, "as it is" - no regular exprressions

Value

TRUE / FALSE

See Also

`str_find`, `str_find1`, `isString`

Examples

```
str_contains("Hello world", "wor")  # TRUE
str_contains("Hello world", "WOR")  # FALSE
str_contains(tolower("Hello world"), tolower("wor"))  # TRUE
str_contains("Hello world", "")  # TRUE
```

`str_find` *str_find*

Description

Find string in another string (without regular expressions), returns indices of all occurences.

Usage

```
str_find(string, patternNoRegex)
```

Arguments

`string`	string in which we try to find something
`patternNoRegex`	string we want to find, "as it is" - no regular exprressions

Value

indices of all occurences (1 = 1st character)

See Also

str_find1, str_contains, isString

Examples

```
str_find("Hello, hello, hello world", "ell")   # 2 9 16
str_find("Hello, hello, hello world", "q")     # integer(0)
```

---

str_find1 *str_find1*

Description

Find string in another string (without regular expressions), returns indices of the first occurence only.

Usage

```
str_find1(string, patternNoRegex)
```

Arguments

string	string in which we try to find something
patternNoRegex	string we want to find, "as it is" - no regular exprressions

Value

index of the first occurence only (1 = 1st character)

See Also

str_find, str_contains, isString

Examples

```
str_find1("Hello, hello, hello world", "ell")   # 2
str_find1("Hello, hello, hello world", "q")     # integer(0)
```

tg.checkTierInd *tg.checkTierInd*

Description

Returns tier index. Input can be either index (number) or tier name (character string). It performs checks whether the tier exists.

Usage

```
tg.checkTierInd(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	Tier index or "name"

Value

Tier index

See Also

tg.getTierName, tg.isIntervalTier, tg.isPointTier, tg.plot, tg.getNumberOfTiers

Examples

```
tg <- tg.sample()
tg.checkTierInd(tg, 4)
tg.checkTierInd(tg, "word")
```

tg.countLabels *tg.countLabels*

Description

Returns number of labels with the specified label.

Usage

```
tg.countLabels(tg, tierInd, label)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
label	character string: label to be counted

Value

integer number

See Also

tg.findLabels, tg.getLabel

Examples

```
tg <- tg.sample()
tg.countLabels(tg, "phone", "a")
```

---

tg.createNewTextGrid *tg.createNewTextGrid*

Description

Creates new and empty TextGrid. tStart and tEnd specify the total start and end time for the TextGrid. If a new interval tier is added later without specified start and end, they are set to TextGrid start and end.

Usage

```
tg.createNewTextGrid(tMin, tMax)
```

Arguments

tMin	Start time of TextGrid
tMax	End time of TextGrid

Details

This empty TextGrid cannot be used for almost anything. At least one tier should be inserted using tg.insertNewIntervalTier() or tg.insertNewPointTier().

Value

TextGrid object

See Also

tg.insertNewIntervalTier, tg.insertNewPointTier

Examples

```
tg <- tg.createNewTextGrid(0, 5)
tg <- tg.insertNewIntervalTier(tg, 1, "word")
tg <- tg.insertInterval(tg, "word", 1, 2, "hello")
tg.plot(tg)
```

tg.duplicateTier *tg.duplicateTier*

Description

Duplicates tier originalInd to new tier with specified index newInd (existing tiers are shifted). It is highly recommended to set a name to the new tier (this can also be done later by tg.setTierName). Otherwise, both original and new tiers have the same name which is permitted but not recommended. In such a case, we cannot use the comfort of using tier name instead of its index in other functions.

Usage

```
tg.duplicateTier(tg, originalInd, newInd = Inf, newTierName = "")
```

Arguments

tg	TextGrid object
originalInd	tier index or "name"
newInd	new tier index (1 = the first, Inf = the last [default])
newTierName	[optional but recommended] name of the new tier

Value

TextGrid object

See Also

tg.duplicateTierMergeSegments, tg.setTierName, tg.removeTier

Examples

```
tg <- tg.sample()
tg2 <- tg.duplicateTier(tg, "word", 1, "NEW")
tg.plot(tg2)
```

tg.duplicateTierMergeSegments *tg.duplicateTierMergeSegments*

Description

Duplicate tier originalInd and merge segments (according to the pattern) to the new tier with specified index newInd (existing tiers are shifted). Typical use: create new syllable tier from phone tier. It merges phones into syllables according to separators in pattern.

Usage

```
tg.duplicateTierMergeSegments(tg, originalInd, newInd = Inf, newTierName,
  pattern, sep = "-")
```

Arguments

tg	TextGrid object
originalInd	tier index or "name"
newInd	new tier index (1 = the first, Inf = the last [default])
newTierName	name of the new tier
pattern	merge segments pattern for the new tier (e.g., "he-llo-world")
sep	separator in pattern (default: "-")

Details

Note 1: there can be segments with empty labels in the original tier (pause), do not specify them in the pattern

Note 2: if there is an segment with empty label in the original tier in the place of separator in the pattern, the empty segment is duplicated into the new tier, i.e. at the position of the separator, there may or may not be an empty segment, if there is, it is duplicated. And they are not specified in the pattern.

Note 3: if the segment with empty label is not at the position corresponding to separator, it leads to error - the part specified in the pattern between separators cannot be split by empty segments

Note 4: beware of labels that appear empty but they are not (space, new line character etc.) - these segments are handled as classical non-empty labels. See example - one label is " ", therefore it must be specified in the pattern.

Value

TextGrid object

See Also

tg.duplicateTier, tg.setTierName, tg.removeTier

Examples

```
tg <- tg.sample()
tg <- tg.removeTier(tg, "syllable")
collapsed <- paste0(tg$phone$label, collapse = "")  # get actual labels
print(collapsed) # all labels in collapsed form - copy the string, include separators -> pattern
pattern <- "ja:-ci-P\\ek-nu-t_so-?u-J\\e-la:S- -nej-dP\\i:f-naj-deZ-h\\ut_S-ku-?a-?a-ta-ma-na:"
tg2 <- tg.duplicateTierMergeSegments(tg, "phone", 1, "syll", pattern, sep = "-")
## Not run:
tg.plot(tg)
tg.plot(tg2)

## End(Not run)
```

`tg.findLabels` *tg.findLabels*

Description

Find label or consecutive sequence of labels and returns their indices.

Usage

```
tg.findLabels(tg, tierInd, labelVector, returnTime = FALSE)
```

Arguments

`tg`	TextGrid object
`tierInd`	tier index or "name"
`labelVector`	character string (one label) or vector of character strings (consecutive sequence of labels) to be found
`returnTime`	If TRUE, return vectors of begin (t1) and end time (t2) for each found group of sequence of labels instead of indices (when FALSE = default).

Value

If returnTime == FALSE, returns list of all occurrences, each member of the list is one occurence and contains vector of label indices, if returnTime == TRUE, returns list witch vectors t1 (begin) and t2 (end) for each found group of sequence of labels.

See Also

`tg.countLabels`, `tg.getLabel`, `tg.duplicateTierMergeSegments`

Examples

```
tg <- tg.sample()
i <- tg.findLabels(tg, "phoneme", "n")
i
length(i)
i[[1]]
i[[2]]
tg$phoneme$label[unlist(i)]

i <- tg.findLabels(tg, "phone", c("?", "a"))
i
length(i)
tg$phone$label[i[[1]]]
tg$phone$label[i[[2]]]
tg$phone$label[unlist(i)]

t <- tg.findLabels(tg, "phone", c("?", "a"), returnTime = TRUE)
t
t$t2[1] - t$t1[1]   # duration of the first result
t$t2[2] - t$t1[2]   # duration of the second result
```

```
i <- tg.findLabels(tg.sample(), "word", c("ti", "reknu", "co"))
i
length(i)
length(i[[1]])
i[[1]]
i[[1]][3]
tg$word$label[i[[1]]]

t <- tg.findLabels(tg.sample(), "word", c("ti", "reknu", "co"), returnTime = TRUE)
pt <- pt.sample()
tStart <- t$t1[1]
tEnd <- t$t2[1]
## Not run:
pt.plot(pt.cut(pt, tStart, tEnd))

## End(Not run)
```

`tg.getEndTime` *tg.getEndTime*

Description

Returns end time. If tier index is specified, it returns end time of the tier, if it is not specified, it returns end time of the whole TextGrid.

Usage

```
tg.getEndTime(tg, tierInd = 0)
```

Arguments

`tg`	TextGrid object
`tierInd`	[optional] tier index or "name"

Value

numeric

See Also

`tg.getStartTime`, `tg.getTotalDuration`

Examples

```
tg <- tg.sample()
tg.getEndTime(tg)
tg.getEndTime(tg, "phone")
```

tg.getIntervalDuration *tg.getIntervalDuration*

Description

Return duration (i.e., end - start time) of interval in interval tier.

Usage

```
tg.getIntervalDuration(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of interval

Value

numeric

See Also

tg.getIntervalStartTime, tg.getIntervalEndTime, tg.getIntervalIndexAtTime, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getIntervalDuration(tg, "phone", 5)
```

tg.getIntervalEndTime *tg.getIntervalEndTime*

Description

Return end time of interval in interval tier.

Usage

```
tg.getIntervalEndTime(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of interval

Value

numeric

See Also

tg.getIntervalStartTime, tg.getIntervalDuration, tg.getIntervalIndexAtTime, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getIntervalEndTime(tg, "phone", 5)
```

---

tg.getIntervalIndexAtTime

tg.getIntervalIndexAtTime

Description

Returns index of interval which includes the given time, i.e. tStart <= time < tEnd. Tier index must belong to interval tier.

Usage

```
tg.getIntervalIndexAtTime(tg, tierInd, time)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
time	time which is going to be found in intervals

Value

integer

See Also

tg.getIntervalStartTime, tg.getIntervalEndTime, tg.getLabel, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getIntervalIndexAtTime(tg, "word", 0.5)
```

tg.getIntervalStartTime *tg.getIntervalStartTime*

Description

Returns start time of interval in interval tier.

Usage

```
tg.getIntervalStartTime(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of interval

Value

numeric

See Also

tg.getIntervalEndTime, tg.getIntervalDuration, tg.getIntervalIndexAtTime, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getIntervalStartTime(tg, "phone", 5)
```

tg.getLabel *tg.getLabel*

Description

Return label of point or interval at the specified index.

Usage

```
tg.getLabel(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of point or interval

Value

character string

See Also

tg.setLabel, tg.countLabels, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getLabel(tg, "phoneme", 4)
tg.getLabel(tg, "phone", 4)
```

---

tg.getNumberOfIntervals *tg.getNumberOfIntervals*

Description

Returns number of intervals in the given interval tier.

Usage

```
tg.getNumberOfIntervals(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"

Value

integer

See Also

tg.getNumberOfPoints

Examples

```
tg <- tg.sample()
tg.getNumberOfIntervals(tg, "phone")
```

tg.getNumberOfPoints *tg.getNumberOfPoints*

Description

Returns number of points in the given point tier.

Usage

```
tg.getNumberOfPoints(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"

Value

integer

See Also

tg.getNumberOfIntervals

Examples

```
tg <- tg.sample()
tg.getNumberOfPoints(tg, "phoneme")
```

tg.getNumberOfTiers *tg.getNumberOfTiers*

Description

Returns number of tiers.

Usage

```
tg.getNumberOfTiers(tg)
```

Arguments

tg	TextGrid object

Value

integer

See Also

tg.getTierName, tg.isIntervalTier, tg.isPointTier

Examples

```
tg <- tg.sample()
tg.getNumberOfTiers(tg)
```

`tg.getPointIndexHigherThanTime`
tg.getPointIndexHigherThanTime

Description

Returns index of point which is nearest the given time from right, i.e. time <= pointTime. Tier index must belong to point tier.

Usage

```
tg.getPointIndexHigherThanTime(tg, tierInd, time)
```

Arguments

`tg`	TextGrid object
`tierInd`	tier index or "name"
`time`	time which is going to be found in points

Value

integer

See Also

`tg.getPointIndexNearestTime`, `tg.getPointIndexLowerThanTime`, `tg.getLabel`, `tg.findLabels`

Examples

```
tg <- tg.sample()
tg.getPointIndexHigherThanTime(tg, "phoneme", 0.5)
```

`tg.getPointIndexLowerThanTime`
tg.getPointIndexLowerThanTime

Description

Returns index of point which is nearest the given time from left, i.e. pointTime <= time. Tier index must belong to point tier.

Usage

```
tg.getPointIndexLowerThanTime(tg, tierInd, time)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
time	time which is going to be found in points

Value

integer

See Also

tg.getPointIndexNearestTime, tg.getPointIndexHigherThanTime, tg.getLabel, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getPointIndexLowerThanTime(tg, "phoneme", 0.5)
```

---

tg.getPointIndexNearestTime

tg.getPointIndexNearestTime

Description

Returns index of point which is nearest the given time (from both sides). Tier index must belong to point tier.

Usage

```
tg.getPointIndexNearestTime(tg, tierInd, time)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
time	time which is going to be found in points

Value

integer

See Also

tg.getPointIndexLowerThanTime, tg.getPointIndexHigherThanTime, tg.getLabel, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getPointIndexNearestTime(tg, "phoneme", 0.5)
```

tg.getPointTime *tg.getPointTime*

Description

Return time of point at the specified index in point tier.

Usage

```
tg.getPointTime(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of point

Value

numeric

See Also

tg.getLabel, tg.getPointIndexNearestTime, tg.getPointIndexLowerThanTime, tg.getPointIndexHigherThanTime, tg.findLabels

Examples

```
tg <- tg.sample()
tg.getPointTime(tg, "phoneme", 4)
```

tg.getStartTime *tg.getStartTime*

Description

Returns start time. If tier index is specified, it returns start time of the tier, if it is not specified, it returns start time of the whole TextGrid.

Usage

```
tg.getStartTime(tg, tierInd = 0)
```

Arguments

tg	TextGrid object
tierInd	[optional] tier index or "name"

Value

numeric

See Also

tg.getEndTime, tg.getTotalDuration

Examples

```
tg <- tg.sample()
tg.getStartTime(tg)
tg.getStartTime(tg, "phone")
```

---

tg.getTierName *tg.getTierName*

Description

Returns name of the tier.

Usage

```
tg.getTierName(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"

Value

character string

See Also

tg.setTierName, tg.isIntervalTier, tg.isPointTier

Examples

```
tg <- tg.sample()
tg.getTierName(tg, 2)
```

tg.getTotalDuration *tg.getTotalDuration*

Description

Returns total duration. If tier index is specified, it returns duration of the tier, if it is not specified, it returns total duration of the TextGrid.

Usage

```
tg.getTotalDuration(tg, tierInd = 0)
```

Arguments

tg	TextGrid object
tierInd	[optional] tier index or "name"

Value

numeric

See Also

tg.getStartTime, tg.getEndTime

Examples

```
tg <- tg.sample()
tg.getTotalDuration(tg)
tg.getTotalDuration(tg, "phone")
```

tg.insertBoundary *tg.insertBoundary*

Description

Inserts new boundary into interval tier. This creates a new interval, to which we can set the label (optional argument).

Usage

```
tg.insertBoundary(tg, tierInd, time, label = "")
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
time	time of the new boundary
label	[optional] label of the new interval

Details

There are more possible situations which influence where the new label will be set.

a) New boundary into the existing interval (the most common situation): The interval is splitted into two parts. The left preserves the label of the original interval, the right is set to the new (optional) label.

b) On the left of existing interval (i.e., enlarging the tier size): The new interval starts with the new boundary and ends at the start of originally first existing interval. The label is set to the new interval.

c) On the right of existing interval (i.e., enlarging the tier size): The new interval starts at the end of originally last existing interval and ends with the new boundary. The label is set to the new interval. This is somewhat different behaviour than in a) and b) where the new label is set to the interval which is on the right of the new boundary. In c), the new label is set on the left of the new boundary. But this is the only logical possibility.

It is a nonsense to insert a boundary between existing intervals to a position where there is no interval. This is against the basic logic of Praat interval tiers where, at the beginning, there is one large empty interval from beginning to the end. And then, it is divided to smaller intervals by adding new boundaries. Nevertheless, if the TextGrid is created by external programmes, you may rarely find such discontinuities. In such a case, at first, use the tgRepairContinuity() function.

Value

TextGrid object

See Also

tg.insertInterval, tg.removeIntervalLeftBoundary, tg.removeIntervalRightBoundary, tg.removeIntervalBothBoundaries, tg.duplicateTierMergeSegments

Examples

```
tg <- tg.sample()
tg2 <- tg.insertNewIntervalTier(tg, 1, "INTERVALS")
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.8)
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.1, "Interval A")
tg2 <- tg.insertInterval(tg2, "INTERVALS", 1.2, 2.5, "Interval B")
## Not run:
tg.plot(tg2)

## End(Not run)
```

---

tg.insertInterval *tg.insertInterval*

Description

Inserts new interval into an empty space in interval tier: a) Into an already existing interval with empty label (most common situation because, e.g., a new interval tier has one empty interval from beginning to the end. b) Outside of existing intervals (left or right), this may create another empty interval between.

Usage

```
tg.insertInterval(tg, tierInd, tStart, tEnd, label = "")
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
tStart	start time of the new interval
tEnd	end time of the new interval
label	[optional] label of the new interval

Details

In most cases, this function is the same as 1.) tgInsertBoundary(tEnd) and 2.) tgInsertBoundary(tStart, "new label"). But, additional checks are performed: a) tStart and tEnd belongs to the same empty interval, or b) both times are outside of existings intervals (both left or both right).

Intersection of the new interval with more already existing (even empty) does not make a sense and is forbidden.

In many situations, in fact, this function creates more than one interval. E.g., let's assume an empty interval tier with one empty interval from 0 to 5 sec. 1.) We insert a new interval from 1 to 2 with label "he". Result: three intervals, 0-1 "", 1-2 "he", 2-5 "". 2.) Then, we insert an interval from 7 to 8 with label "lot". Result: five intervals, 0-1 "", 1-2 "he", 2-5 "", 5-7 "", 7-8 "lot" Note: the empty 5-7 "" interval is inserted because we are going outside of the existing tier. 3.) Now, we insert a new interval exactly between 2 and 3 with label "said". Result: really only one interval is created (and only the right boundary is added because the left one already exists): 0-1 "", 1-2 "he", 2-3 "said", 3-5 "", 5-7 "", 7-8 "lot". 4.) After this, we want to insert another interval, 3 to 5: label "a". In fact, this does not create any new interval at all. Instead of that, it only sets the label to the already existing interval 3-5. Result: 0-1 "", 1-2 "he", 2-3 "said", 3-5 "a", 5-7 "", 7-8 "lot".

This function is not implemented in Praat (6.0.14). And it is very useful for adding separate intervals to an empty area in interval tier, e.g., result of voice activity detection algorithm. On the other hand, if we want continuously add new consequential intervals, tgInsertBoundary() may be more useful. Because, in the tgInsertInterval() function, if we calculate both boundaries separately for each interval, strange situations may happen due to numeric round-up errors, like 3.14*5 != 15.7. In such cases, it may be hard to obtain precisely consequential time instances. As 3.14*5 is slightly larger than 15.7 (let's try to calculate 15.7 - 3.14*5), if you calculate tEnd of the first interval as 3.14*5 and tStart of the second interval as 15.7, this function refuse to create the second interval because it would be an intersection. In the opposite case (tEnd of the 1st: 15.7, tStart of the 2nd: 3.14*5), it would create another "micro" interval between these two slightly different time instances. Instead of that, if you insert only one boundary using the tgInsertBoundary() function, you are safe that only one new interval is created. But, if you calculate the "15.7" (no matter how) and store in the variable and then, use this variable in the tgInsertInterval() function both for the tEnd of the 1st interval and tStart of the 2nd interval, you are safe, it works fine.

Value

TextGrid object

See Also

tg.insertBoundary, tg.removeIntervalLeftBoundary, tg.removeIntervalRightBoundary, tg.removeIntervalBothBoundaries, tg.duplicateTierMergeSegments

Examples

```
tg <- tg.sample()
tg2 <- tg.insertNewIntervalTier(tg, 1, "INTERVALS")
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.8)
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.1, "Interval A")
tg2 <- tg.insertInterval(tg2, "INTERVALS", 1.2, 2.5, "Interval B")
## Not run:
tg.plot(tg2)

## End(Not run)
```

`tg.insertNewIntervalTier` *tg.insertNewIntervalTier*

Description

Inserts new interval tier to the specified index (existing tiers are shifted). The new tier contains one empty interval from beginning to end. Then, if we add new boundaries, this interval is divided to smaller pieces.

Usage

```
tg.insertNewIntervalTier(tg, newInd = Inf, newTierName, tMin = NA,
  tMax = NA)
```

Arguments

`tg`	TextGrid object
`newInd`	new tier index (1 = the first, Inf = the last [default])
`newTierName`	new tier name
`tMin`	[optional] start time of the new tier
`tMax`	[optional] end time of the new tier

Value

TextGrid object

See Also

`tg.insertInterval`, `tg.insertNewPointTier`, `tg.duplicateTier`, `tg.duplicateTierMergeSegments`, `tg.removeTier`

Examples

```
## Not run:
tg <- tg.sample()
tg2 <- tg.insertNewIntervalTier(tg, 1, "INTERVALS")
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.8)
tg2 <- tg.insertBoundary(tg2, "INTERVALS", 0.1, "Interval A")
tg2 <- tg.insertInterval(tg2, "INTERVALS", 1.2, 2.5, "Interval B")
```

```
tg2 <- tg.insertNewIntervalTier(tg2, Inf, "LastTier")
tg2 <- tg.insertInterval(tg2, "LastTier", 1, 3, "This is the last tier")
tg.plot(tg2)

## End(Not run)
```

`tg.insertNewPointTier` *tg.insertNewPointTier*

Description

Inserts new point tier to the specified index (existing tiers are shifted).

Usage

```
tg.insertNewPointTier(tg, newInd = Inf, newTierName)
```

Arguments

`tg`	TextGrid object
`newInd`	new tier index (1 = the first, Inf = the last [default])
`newTierName`	new tier name

Value

TextGrid object

See Also

`tg.insertPoint`, `tg.insertNewIntervalTier`, `tg.duplicateTier`, `tg.removeTier`

Examples

```
## Not run:
tg <- tg.sample()
tg2 <- tg.insertNewPointTier(tg, 1, "POINTS")
tg2 <- tg.insertPoint(tg2, "POINTS", 3, "MY POINT")
tg2 <- tg.insertNewPointTier(tg2, Inf, "POINTS2")  # the last tier
tg2 <- tg.insertPoint(tg2, "POINTS2", 2, "point in the last tier")
tg.plot(tg2)

## End(Not run)
```

tg.insertPoint *tg.insertPoint*

Description

Inserts new point to point tier of the given index.

Usage

```
tg.insertPoint(tg, tierInd, time, label)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
time	time of the new point
label	time of the new point

Value

TextGrid object

See Also

tg.removePoint, tg.insertInterval, tg.insertBoundary

Examples

```
## Not run:
tg <- tg.sample()
tg2 <- tg.insertPoint(tg, "phoneme", 1.4, "NEW POINT")
tg.plot(tg2)

## End(Not run)
```

tg.isIntervalTier *tg.isIntervalTier*

Description

Returns TRUE if the tier is IntervalTier, FALSE otherwise.

Usage

```
tg.isIntervalTier(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"

Value

TRUE / FALSE

See Also

tg.isPointTier, tg.getTierName, tg.findLabels

Examples

```
tg <- tg.sample()
tg.isIntervalTier(tg, 1)
tg.isIntervalTier(tg, "word")
```

---

tg.isPointTier *tg.isPointTier*

Description

Returns TRUE if the tier is PointTier, FALSE otherwise.

Usage

```
tg.isPointTier(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"

Value

TRUE / FALSE

See Also

tg.isIntervalTier, tg.getTierName, tg.findLabels

Examples

```
tg <- tg.sample()
tg.isPointTier(tg, 1)
tg.isPointTier(tg, "word")
```

tg.plot *tg.plot*

Description

Plots interactive TextGrid using dygraphs package.

Usage

```
tg.plot(tg, group = "")
```

Arguments

tg	TextGrid object
group	[optional] character string, name of group for dygraphs synchronization

See Also

tg.read, pt.plot

Examples

```
## Not run:
tg <- tg.sample()
tg.plot(tg)

## End(Not run)
```

tg.read *tg.read*

Description

Loads TextGrid from Praat in Text or Short text format (UTF-8), it handles both Interval and Point tiers. Labels can may contain quotation marks and new lines.

Usage

```
tg.read(fileNameTextGrid, encoding = "UTF-8")
```

Arguments

fileNameTextGrid	Input file name
encoding	File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

TextGrid object

See Also

tg.write, tg.plot, tg.repairContinuity, tg.createNewTextGrid, tg.findLabels, tg.duplicateTierMergeSegm
pt.read, pitch.read, it.read, col.read

Examples

```
## Not run:
tg <- tg.read("demo/H.TextGrid")
tg.plot(tg)

## End(Not run)
```

---

tg.removeIntervalBothBoundaries
tg.removeIntervalBothBoundaries

Description

Remove both left and right boundary of interval of the given index in Interval tier. In fact, this operation concatenate three intervals into one (and their labels). It cannot be applied to the first and the last interval because they contain beginning or end boundary of the tier. E.g., let's assume interval 1-2-3. We remove both boundaries of the 2nd interval. The result is one interval 123. If we do not want to concatenate labels (we wanted to remove the label including its interval), we can set the label of the second interval to the empty string "" before this operation. If we only want to remove the label of interval "without concatenation", i.e., the desired result is 1-empty-3, it is not this operation of removing boundaries. Just set the label of the second interval to the empty string "".

Usage

```
tg.removeIntervalBothBoundaries(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of the interval

Value

TextGrid object

See Also

tg.removeIntervalLeftBoundary, tg.removeIntervalRightBoundary, tg.insertBoundary, tg.insertInterval

Examples

```
## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeIntervalBothBoundaries(tg, "word", 3)
tg.plot(tg2)

## End(Not run)
```

`tg.removeIntervalLeftBoundary`
tg.removeIntervalLeftBoundary

Description

Remove left boundary of the interval of the given index in Interval tier. In fact, it concatenates two intervals into one (and their labels). It cannot be applied to the first interval because it is the start boundary of the tier. E.g., we have interval 1-2-3, we remove the left boundary of the 2nd interval, the result is two intervals 12-3. If we do not want to concatenate labels, we have to set the label to the empty string "" before this operation.

Usage

```
tg.removeIntervalLeftBoundary(tg, tierInd, index)
```

Arguments

`tg`	TextGrid object
`tierInd`	tier index or "name"
`index`	index of the interval

Value

TextGrid object

See Also

`tg.removeIntervalRightBoundary`, `tg.removeIntervalBothBoundaries`, `tg.insertBoundary`, `tg.insertInterval`

Examples

```
## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeIntervalLeftBoundary(tg, "word", 3)
tg.plot(tg2)

## End(Not run)
```

tg.removeIntervalRightBoundary
tg.removeIntervalRightBoundary

Description

Remove right boundary of the interval of the given index in Interval tier. In fact, it concatenates two intervals into one (and their labels). It cannot be applied to the last interval because it is the end boundary of the tier. E.g., we have interval 1-2-3, we remove the right boundary of the 2nd interval, the result is two intervals 1-23. If we do not want to concatenate labels, we have to set the label to the empty string "" before this operation.

Usage

```
tg.removeIntervalRightBoundary(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of the interval

Value

TextGrid object

See Also

tg.removeIntervalLeftBoundary, tg.removeIntervalBothBoundaries, tg.insertBoundary, tg.insertInterval

Examples

```
## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeIntervalRightBoundary(tg, "word", 3)
tg.plot(tg2)

## End(Not run)
```

tg.removePoint *tg.removePoint*

Description

Remove point of the given index from the point tier.

Usage

```
tg.removePoint(tg, tierInd, index)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of point to be removed

Value

TextGrid object

See Also

tg.insertPoint, tg.getNumberOfPoints, tg.removeIntervalBothBoundaries

Examples

```
tg <- tg.sample()
tg$phoneme$label
tg2 <- tg.removePoint(tg, "phoneme", 1)
tg2$phoneme$label
```

tg.removeTier *tg.removeTier*

Description

Removes tier of the given index.

Usage

```
tg.removeTier(tg, tierInd)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"

Value

TextGrid object

See Also

tg.insertNewIntervalTier, tg.insertNewPointTier, tg.duplicateTier

Examples

```
## Not run:
tg <- tg.sample()
tg.plot(tg)
tg2 <- tg.removeTier(tg, "word")
tg.plot(tg2)

## End(Not run)
```

---

tg.repairContinuity *tg.repairContinuity*

Description

Repairs problem of continuity of T2 and T1 in interval tiers. This problem is very rare and it should not appear. However, e.g., automatic segmentation tool Prague Labeller produces random numeric round-up errors featuring, e.g., T2 of preceding interval is slightly higher than the T1 of the current interval. Because of that, the boundary cannot be manually moved in Praat edit window.

Usage

```
tg.repairContinuity(tg, verbose = TRUE)
```

Arguments

tg	TextGrid object
verbose	[optional, default=TRUE] If FALSE, the function performs everything quietly.

Value

TextGrid object

See Also

tg.sampleProblem

Examples

```
## Not run:
tgProblem <- tg.sampleProblem()
tgNew <- tg.repairContinuity(tgProblem)
tg.write(tgNew, "demo_problem_OK.TextGrid")

## End(Not run)
```

tg.sample *tg.sample*

Description

Returns sample TextGrid.

Usage

```
tg.sample()
```

Value

TextGrid

See Also

tg.plot

Examples

```
tg <- tg.sample()
tg.plot(tg)
```

tg.sampleProblem *tg.sampleProblem*

Description

Returns sample TextGrid with continuity problem.

Usage

```
tg.sampleProblem()
```

Value

TextGrid

See Also

tg.repairContinuity

Examples

```
tg <- tg.sampleProblem()
tg2 <- tg.repairContinuity(tg)
tg2 <- tg.repairContinuity(tg2)
tg.plot(tg2)
```

tg.setLabel *tg.setLabel*

Description

Sets (changes) label of interval or point of the given index in the interval or point tier.

Usage

```
tg.setLabel(tg, tierInd, index, newLabel)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
index	index of interval or point
newLabel	new "label"

See Also

tg.getLabel

Examples

```
tg <- tg.sample()
tg2 <- tg.setLabel(tg, "word", 3, "New Label")
tg.getLabel(tg2, "word", 3)
```

tg.setTierName *tg.setTierName*

Description

Sets (changes) name of tier of the given index.

Usage

```
tg.setTierName(tg, tierInd, name)
```

Arguments

tg	TextGrid object
tierInd	tier index or "name"
name	new "name" of the tier

See Also

tg.getTierName

Examples

```
tg <- tg.sample()
tg2 <- tg.setTierName(tg, "word", "WORDTIER")
tg.getTierName(tg2, 4)
```

`tg.write` *tg.write*

Description

Saves TextGrid to the file. TextGrid may contain both interval and point tiers (tg[[1]], tg[[2]], tg[[3]], etc.). If tier type is not specified in $type, is is assumed to be "interval". If specified, $type have to be "interval" or "point". If there is no class(tg)["tmin"] and class(tg)["tmax"], they are calculated as min and max of all tiers. The file is saved in UTF-8 encoding.

Usage

```
tg.write(tg, fileNameTextGrid, format = "short")
```

Arguments

`tg`	TextGrid object
`fileNameTextGrid`	Output file name
`format`	Output file format ("short" (default, short text format) or "text" (a.k.a. full text format))

See Also

`tg.read`, `pt.write`

Examples

```
## Not run:
tg <- tg.sample()
tg.write(tg, "demo_output.TextGrid")

## End(Not run)
```

Index