Package 'rPraat'

March 6, 2017

Type Package

Title Read, write and manipulate Praat TextGrid and PitchTier

Version 1.0.4

Encoding UTF-8

Maintainer Tomas Boril <borilt@gmail.com>

Description rPraat package for R constitutes an interface between the most popular software for phonetic analyses, Praat, and R.

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

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

License MIT + file LICENSE

LazyData TRUE

Depends R (>= 3.3.2)

Imports dplyr (>= 0.5.0),
stringr (>= 1.2.0),
readr(>= 1.0.0),
dygraphs (>= 1.1.1.4),
tbTools (>= 1.0.0)

RoxygenNote 6.0.1

Remotes bbTomas/tbTools

Suggests testthat

R topics documented:

`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.Hz2ST

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)")

## 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

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.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.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, spread sheet, headerless spread sheet (headerless not recommended, it does not contain tmin and tmax info).

Usage

```
pt.read(fileNamePitchTier)
```

Arguments

`fileNamePitchTier`
: file name of PitchTier

Value

PitchTier object

See Also

`pt.write`, `pt.plot`, `pt.Hz2ST`, `pt.interpolate`, `tg.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 (spread sheet file format). 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)
```

Arguments

pt	PitchTier object
fileNamePitchTier	file name to be created

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)
```

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.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, newTierName = "")
```

Arguments

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

Value

TextGrid object

See Also

tg.setTierName, tg.removeTier

Examples

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

---

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

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

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

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

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

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`

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

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

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`

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

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 og 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

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, newTierName, tMin = NA, tMax = NA)
```

Arguments

`tg`	TextGrid object
`newInd`	new tier index (1 = the first)
`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.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")
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, newTierName)
```

Arguments

tg	TextGrid object
newInd	new tier index (1 = the first)
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")
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

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

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)
```

Arguments

`fileNameTextGrid`
: Input file name

Value

TextGrid object

See Also

`tg.write`, `tg.plot`, `tg.repairContinuity`, `tg.createNewTextGrid`, `pt.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 = FALSE)
```

Arguments

tg	TextGrid object
verbose	[optional, default=FALSE] If TRUE, 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 Short text file, UTF-8 format.

Usage

```
tg.write(tg, fileNameTextGrid)
```

Arguments

`tg`	TextGrid object
`fileNameTextGrid`	Output file name

See Also

`tg.read`, `pt.write`

Examples

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

## End(Not run)
```

Index