Package ‘rPraat’

February 27, 2021

Type Package
Title Interface to Praat
Version 1.3.2-1
Encoding UTF-8
Maintainer Tomas Boril <borilt@gmail.com>
Description Read, write and manipulate ‘Praat’ TextGrid, PitchTier, Pitch, IntensityTier, For-
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 (>= 0.8.5), stringr (>= 1.4.0),
       readr(>= 1.3.1), dygraphs (>= 1.1.1.6), tuneR (>= 1.3.3)
RoxygenNote 7.1.1
Suggests testthat
NeedsCompilation no
Author Tomas Boril [aut, cre]
Repository CRAN
Date/Publication 2021-02-27 22:40:02 UTC

R topics documented:

  as.formant ................................................................. 4
  as.it ................................................................. 5
  as.pitch ............................................................... 5
  as.pt ................................................................. 6
  as.snd ............................................................... 6
  as.tg ................................................................. 7
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>col.read</td>
<td>8</td>
</tr>
<tr>
<td>col.write</td>
<td>9</td>
</tr>
<tr>
<td>detectEncoding</td>
<td>10</td>
</tr>
<tr>
<td>formant.cut</td>
<td>10</td>
</tr>
<tr>
<td>formant.cut0</td>
<td>11</td>
</tr>
<tr>
<td>formant.getPointIndexHigherThanTime</td>
<td>12</td>
</tr>
<tr>
<td>formant.getPointIndexLowerThanTime</td>
<td>13</td>
</tr>
<tr>
<td>formant.getPointIndexNearestTime</td>
<td>14</td>
</tr>
<tr>
<td>formant.plot</td>
<td>14</td>
</tr>
<tr>
<td>formant.read</td>
<td>15</td>
</tr>
<tr>
<td>formant.sample</td>
<td>16</td>
</tr>
<tr>
<td>formant.toArray</td>
<td>17</td>
</tr>
<tr>
<td>formant.toFrame</td>
<td>17</td>
</tr>
<tr>
<td>formant.write</td>
<td>18</td>
</tr>
<tr>
<td>ifft</td>
<td>19</td>
</tr>
<tr>
<td>isInt</td>
<td>19</td>
</tr>
<tr>
<td>isLogical</td>
<td>20</td>
</tr>
<tr>
<td>isNum</td>
<td>21</td>
</tr>
<tr>
<td>isString</td>
<td>22</td>
</tr>
<tr>
<td>it.cut</td>
<td>22</td>
</tr>
<tr>
<td>it.cut0</td>
<td>23</td>
</tr>
<tr>
<td>it.getPointIndexHigherThanTime</td>
<td>24</td>
</tr>
<tr>
<td>it.getPointIndexLowerThanTime</td>
<td>25</td>
</tr>
<tr>
<td>it.getPointIndexNearestTime</td>
<td>26</td>
</tr>
<tr>
<td>it.interpolate</td>
<td>26</td>
</tr>
<tr>
<td>it.legendre</td>
<td>27</td>
</tr>
<tr>
<td>it.legendreDemo</td>
<td>28</td>
</tr>
<tr>
<td>it.legendreSynth</td>
<td>28</td>
</tr>
<tr>
<td>it.plot</td>
<td>29</td>
</tr>
<tr>
<td>it.read</td>
<td>30</td>
</tr>
<tr>
<td>it.sample</td>
<td>31</td>
</tr>
<tr>
<td>it.write</td>
<td>31</td>
</tr>
<tr>
<td>pitch.cut</td>
<td>32</td>
</tr>
<tr>
<td>pitch.cut0</td>
<td>33</td>
</tr>
<tr>
<td>pitch.getPointIndexHigherThanTime</td>
<td>34</td>
</tr>
<tr>
<td>pitch.getPointIndexLowerThanTime</td>
<td>35</td>
</tr>
<tr>
<td>pitch.getPointIndexNearestTime</td>
<td>35</td>
</tr>
<tr>
<td>pitch.plot</td>
<td>36</td>
</tr>
<tr>
<td>pitch.read</td>
<td>37</td>
</tr>
<tr>
<td>pitch.sample</td>
<td>38</td>
</tr>
<tr>
<td>pitch.toArray</td>
<td>39</td>
</tr>
<tr>
<td>pitch.toFrame</td>
<td>39</td>
</tr>
<tr>
<td>pitch.write</td>
<td>40</td>
</tr>
<tr>
<td>pt.cut</td>
<td>41</td>
</tr>
<tr>
<td>pt.cut0</td>
<td>42</td>
</tr>
<tr>
<td>pt.getPointIndexHigherThanTime</td>
<td>43</td>
</tr>
<tr>
<td>pt.getPointIndexLowerThanTime</td>
<td>43</td>
</tr>
<tr>
<td>pt.getPointIndexNearestTime</td>
<td>44</td>
</tr>
</tbody>
</table>
R topics documented:

- pt.Hz2ST ......................................................... 45
- pt.interpolate .............................................. 45
- pt.legendre ................................................... 46
- pt.legendreDemo ............................................. 47
- pt.legendreSynth ........................................... 48
- pt.plot ......................................................... 49
- pt.read ......................................................... 49
- pt.sample ..................................................... 50
- pt.write ....................................................... 51
- round2 ........................................................ 51
- seqM .......................................................... 52
- snd.cut ......................................................... 54
- snd.cut0 ....................................................... 55
- snd.getPointIndexHigherThanTime ....................... 56
- snd.getPointIndexLowerThanTime ......................... 57
- snd.getPointIndexNearestTime ........................... 57
- snd.plot ....................................................... 58
- snd.read ....................................................... 59
- snd.sample ................................................... 60
- snd.write ..................................................... 60
- strTrim ......................................................... 61
- str_contains ................................................ 62
- str_find ........................................................ 62
- str_find1 ...................................................... 63
- tg.boundaryMagnet .......................................... 64
- tg.checkTierInd ............................................. 65
- tg.countLabels ............................................. 66
- tg.createNewTextGrid ..................................... 66
- tg.cut ........................................................ 67
- tg.cut0 ......................................................... 68
- tg.duplicateTier ............................................ 69
- tg.duplicateTierMergeSegments ......................... 70
- tg.findLabels ............................................... 71
- tg.getEndTime ............................................... 73
- tg.getIntervalDuration ................................... 74
- tg.getIntervalEndTime .................................... 74
- tg.getIntervalIndexAtTime ............................... 75
- tg.getIntervalStartTime ................................. 76
- tg.getLabel .................................................... 76
- tg.getNumberOfIntervals ................................. 77
- tg.getNumberOfPoints ..................................... 78
- tg.getNumberOfTiers ....................................... 78
- tg.getPointIndexHigherThanTime ........................ 79
- tg.getPointIndexLowerThanTime ......................... 80
- tg.getPointIndexNearestTime ............................ 80
- tg.getLabel .................................................... 81
- tg.getTierName .............................................. 82
Renames the `class(formant)["name"]` attribute and sets `class(formant)["type"]` <-"Formant 2" (if it is not already set)

**Usage**

```r
as.formant(formant, name = "")
```

**Arguments**

- `formant` Formant 2 object
- `name` New name

**Value**

Formant 2 object

**Examples**

```r
class(formant.sample())
class(as.formant(formant.sample(), name = "New Name"))
```
Description

Renames the class(it)["name"] attribute and sets class(it)["type"] <- "IntensityTier" (if it is not already set)

Usage

as.it(it, name = "")

Arguments

it 
IntensityTier object
name 
New name

Value

IntensityTier object

Examples

class(it.sample())
class(as.it(it.sample(), name = "New Name"))

Description

Renames the class(pitch)["name"] attribute and sets class(pitch)["type"] <- "Pitch 1" (if it is not already set)

Usage

as.pitch(pitch, name = "")

Arguments

pitch 
Pitch 1 object
name 
New name

Value

Pitch 1 object
Examples

```
class(pitch.sample())
class(as.pitch(pitch.sample(), name = "New Name"))
```

---

**as.pt**

Description

Renames the `class(pt)["name"]` attribute and sets `class(pt)["type"] <- "PitchTier"` (if it is not already set)

Usage

```
as.pt(pt, name = "")
```

Arguments

- `pt` PitchTier object
- `name` New name

Value

PitchTier object

Examples

```
class(pt.sample())
class(as.pt(pt.sample(), name = "New Name"))
```

---

**as.snd**

Description

Renames the `class(snd)["name"]` attribute and sets `class(snd)["type"] <- "Sound"` (if it is not already set)

Usage

```
as.snd(snd, name = "")
```

Arguments

- `snd` snd object
- `name` New name
Details
At least, $\text{sig}$ and $\text{fs}$ members must be present in $\text{snd}$ list.
If not present, it calculates $\text{t}$, $\text{nChannels}$, $\text{nBits}$ (default: 16), $\text{nSamples}$, and $\text{duration}$ members of $\text{snd}$ list.

Value
$\text{snd}$ object

Examples

```r
class(snd.sample())
class(as.snd(snd.sample(), name = "New Name"))
```

---

**as.tg**  
**as.tg**

Description
Renames the `class(tg)["name"]` attribute and sets `class(tg)["type"] <- "TextGrid"` (if it is not already set)

Usage
```
as.tg(tg, name = "")
```

Arguments
- **tg**  
  TextGrid object
- **name**  
  New name

Value
TextGrid object

Examples

```r
class(tg.sample())
class(as.tg(tg.sample(), name = "New Name"))
```
Description

Loads Collection from Praat in Text or Short text format. Collection may contain combination of TextGrids, PitchTiers, Pitch objects, Formant 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, formant.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$n # number of frames
pitch$t[4] # time instance of the 4th frame
pitch$frame[[4]] # 4th frame: pitch candidates
Description

Saves Collection of objects to a file (in UTF-8 encoding). col is list of objects, each item col[[i]] must contain class(col[[i]])["type"] ("TextGrid", "PitchTier", "IntensityTier", "Pitch 1", or "Formant 2") and class(col[[i]])["name"] (name of the object) parameters set. These parameters can be created easily using "as.something()" functions: as.tg(), as.pt(), as.it(), as.pitch(), as.formant()

Usage

col.write(col, fileNameCollection, format = "short")

Arguments

col  Collection object = list of objects (col[[1]], col[[2]], etc.) with class(col[[i]])["type"] and class(col[[i]])["name"] parameters set

fileNameCollection  file name to be created

format  Output file format ("short" (short text format) or "text" (a.k.a. full text format))

Details

Sound objects in col.read() and col.write() are not supported at this moment because they would occupy too much disc space in text format.

See Also

col.read
Examples

```r
## Not run:
col <- list(as.tg(tg.sample(), "My textgrid"), as.pt(pt.sample(), "My PitchTier 1"),
as.pt(pt.Hz2ST(pt.sample()), "My PitchTier 2"), as.it(it.sample(), "My IntensityTier"),
as.pitch(pitch.sample(), "My Pitch"), as.formant(formant.sample(), "My Formant"))
col.write(col, "my_collection.Collection")
## 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

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

formant.cut

Description
Cut the specified interval from the Formant object and preserve time

Usage
formant.cut(formant, tStart = -Inf, tEnd = Inf)
Arguments

formant  Formant object (either in Frame or Array format)
tStart  beginning time of interval to be cut (default -Inf = cut from the xmin of the Formant)
tEnd  final time of interval to be cut (default Inf = cut to the xmax of the Formant)

Value

Formant object

See Also

formant.cut0, tg.cut, tg.cut0, formant.read, formant.plot

Examples

formant <- formant.sample()
formant2 <- formant.cut(formant, tStart = 3)
formant2_0 <- formant.cut0(formant, tStart = 3)
formant3 <- formant.cut(formant, tStart = 2, tEnd = 3)
formant3_0 <- formant.cut0(formant, tStart = 2, tEnd = 3)
formant4 <- formant.cut(formant, tEnd = 1)
formant4_0 <- formant.cut0(formant, tEnd = 1)
formant5 <- formant.cut(formant, tStart = -1, tEnd = 1)
formant5_0 <- formant.cut0(formant, tStart = -1, tEnd = 1)

## Not run:
formant.plot(formant)
formant.plot(formant2)
formant.plot(formant2_0)
formant.plot(formant3)
formant.plot(formant3_0)
formant.plot(formant4)
formant.plot(formant4_0)
formant.plot(formant5)
formant.plot(formant5_0)

## End(Not run)

Description

Cut the specified interval from the Formant object and shift time so that the new xmin = 0

Usage

formant.cut0(formant, tStart = -Inf, tEnd = Inf)
formant.getPointIndexHigherThanTime

Arguments

formant Formant object (either in Frame or Array format)
tStart beginning time of interval to be cut (default \(-\infty = \text{cut from the } \text{xmin of the Formant}\))
tEnd final time of interval to be cut (default \(\infty = \text{cut to the } \text{xmax of the Formant}\))

Value

Formant object

See Also

formant.cut, tg.cut, tg.cut0, formant.read, formant.plot

Examples

```r
formant <- formant.sample()
formant2 <- formant.cut(formant, tStart = 3)
formant2_0 <- formant.cut0(formant, tStart = 3)
formant3 <- formant.cut(formant, tStart = 2, tEnd = 3)
formant3_0 <- formant.cut0(formant, tStart = 2, tEnd = 3)
formant4 <- formant.cut(formant, tEnd = 1)
formant4_0 <- formant.cut0(formant, tEnd = 1)
formant5 <- formant.cut(formant, tStart = -1, tEnd = 1)
formant5_0 <- formant.cut0(formant, tStart = -1, tEnd = 1)
## Not run:
formant.plot(formant)
formant.plot(formant2)
formant.plot(formant2_0)
formant.plot(formant3)
formant.plot(formant3_0)
formant.plot(formant4)
formant.plot(formant4_0)
formant.plot(formant5)
formant.plot(formant5_0)
## End(Not run)
```

Description

Returns index of frame which is nearest the given time from right, i.e. \(\text{time} \leq \text{frameTime}\).

Usage

```r
formant.getPointIndexHigherThanTime(formant, time)
```
formant.getPointIndexLowerThanTime

Arguments

formant     Formant object
            time     time which is going to be found in frames

Value

integer

See Also

formant.getPointIndexNearestTime, formant.getPointIndexLowerThanTime

Examples

formant <- formant.sample()
formant.getPointIndexLowerThanTime(formant, 0.5)

formant.getPointIndexLowerThanTime

formant.getPointIndexLowerThanTime

Description

Returns index of frame which is nearest the given time from left, i.e. frameTime <= time.

Usage

formant.getPointIndexLowerThanTime(formant, time)

Arguments

formant     Formant object
            time     time which is going to be found in frames

Value

integer

See Also

formant.getPointIndexNearestTime, formant.getPointIndexHigherThanTime

Examples

formant <- formant.sample()
formant.getPointIndexLowerThanTime(formant, 0.5)
formant.getPointIndexNearestTime

Description

Returns index of frame which is nearest the given time (from both sides).

Usage

formant.getPointIndexNearestTime(formant, time)

Arguments

formant Formant object
time time which is going to be found in frames

Value

integer

See Also

formant.getPointIndexLowerThanTime, formant.getPointIndexHigherThanTime

Examples

formant <- formant.sample()
formant.getPointIndexNearestTime(formant, 0.5)

formant.plot

Description

Plots interactive Formant object using dygraphs package.

Usage

formant.plot(formant, scaleIntensity = TRUE, drawBandwidth = TRUE, group = "")

Arguments

formant Formant object
scaleIntensity Point size scaled according to relative intensity
drawBandwidth Draw formant bandwidth
group [optional] character string, name of group for dygraphs synchronization
See Also

formant.read, formant.sample, formant.toArray, tg.plot

Examples

```r
## Not run:
formant <- formant.sample()
formant.plot(formant, drawBandwidth = TRUE)

## End(Not run)
```

Description


Usage

```r
formant.read(fileNameFormant, encoding = "UTF-8")
```

Arguments

- `fileNameFormant`: file name of Formant object
- `encoding`: File encoding (default: "UTF-8"), "auto" for auto-detect of Unicode encoding

Value

A Formant object represents formants as a function of time.

- `f$xmin`: start time (seconds)
- `f$xmax`: end time (seconds)
- `f$nx`: number of frames
- `f$dx`: time step = frame duration (seconds)
- `f$x1`: time associated with the first frame (seconds)
- `f$t`: vector of time instances associated with all frames
- `f$maxnFormants`: maximum number of formants in frame
- `f$frame[[1]]` to `f$frame[[f$nx]]`: frames
- `f$frame[[1]]$intensity`: intensity of the frame
- `f$frame[[1]]$nFormants`: actual number of formants in this frame
- `f$frame[[1]]$frequency`: vector of formant frequencies (in Hz)
- `f$frame[[1]]$bandwidth`: vector of formant bandwidths (in Hz)
See Also

formant.write, formant.plot, formant.cut, formant.getPointIndexNearestTime, pitch.read, pt.read, tg.read, it.read, col.read

Examples

## Not run:
f <- formant.read('demo/maminka.Formant')
names(f)
f$nx
f$t[4] # time instance of the 4th frame
f$frame[[4]] # 4th frame: formants
f$frame[[4]]$frequency[2]
f$frame[[4]]$bandwidth[2]

## End(Not run)
### Description
formant.toArray

### Usage
formant.toArray(formant)

### Arguments
- **formant**: Formant object

### Value
Formant object with frames converted to frequency and bandwidth arrays and intensity vector

### See Also
- `formant.read`
- `formant.plot`

### Examples
```r
formantArray <- formant.toArray(formant.sample())
formantArray$t[, 1:10]
formantArray$freqencyArray[, 1:10]
formantArray$bandwidthArray[, 1:10]
formantArray$intensityVector[1:10]
## Not run:
plot(formantArray$t, formantArray$freqencyArray[1, ]) # draw 1st formant track
## End(Not run)
```

---

### Description
formant.toFrame

### Usage
formant.toFrame(formantArray)

### Description
formant.toFrame
formant.write

Arguments

formantArray  Formant object (array format)

Value

Formant object with frames

See Also

formant.toArray, formant.read, formant.plot

Examples

formantArray <- formant.toArray(formant.sample())
formant <- formant.toFrame(formantArray)

formant.write(formant, fileNameFormant, format = "short")

Arguments

formant  Formant object
fileNameFormant  Output file name
format  Output file format ("short" (default, short text format) or "text" (a.k.a. full text format))

See Also

formant.read, tg.read

Examples

## Not run:
formant <- formant.sample()
formant.write(formant, "demo_output.Formant")

## End(Not run)
### ifft

**Description**

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

**Usage**

```plaintext
ifft(sig)
```

**Arguments**

- **sig**
  - input vector

**Details**

This really is 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**

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

---

### 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**

```plaintext
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))

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
isNum

Examples

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

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

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

Description

Cut the specified interval from the IntensityTier and preserve time

Usage

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

Arguments

it IntensityTier object

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

```r
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)
rt4_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:
```r
it.plot(it)
```
```r
it.plot(it2)
```
```r
it.plot(it2_0)
```
```r
it.plot(it3)
```
```r
it.plot(it3_0)
```
```r
it.plot(it4)
```
```r
it.plot(it4_0)
```
```r
it.plot(it5)
```
```r
it.plot(it5_0)
```

## End(Not run)

Description

Cut the specified interval from the IntensityTier and shift time so that the new \( t_{\text{min}} = 0 \)

Usage

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

Arguments

- **it**: IntensityTier object
- **tStart**: beginning time of interval to be cut (default \(-Inf = \text{cut from the } t_{\text{min}} \text{ of the IntensityTier})
- **tEnd**: final time of interval to be cut (default \(Inf = \text{cut to the } t_{\text{max}} \text{ of the IntensityTier})
it.getPointIndexHigherThanTime

Value

IntensityTier object

See Also

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

Examples

```r
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.getPointIndexLowerThanTime

Value

integer

See Also

it.getPointIndexNearestTime, it.getPointIndexLowerThanTime

Examples

it <- it.sample()
it.getPointIndexHigherThanTime(it, 0.5)

Description

Returns index of point which is nearest the given time from left, i.e. pointTime <= time.

Usage

it.getPointIndexLowerThanTime(it, time)

Arguments

it        IntensityTier object

            time        time which is going to be found in points

Value

integer

See Also

it.getPointIndexNearestTime, it.getPointIndexHigherThanTime

Examples

it <- it.sample()
it.getPointIndexLowerThanTime(it, 0.5)
**it.getPointIndexNearestTime**

### Description

Returns index of point which is nearest the given time (from both sides).

### Usage

```
     it.getPointIndexNearestTime(it, time)
```

### Arguments

- **it**: IntensityTier object
- **time**: time which is going to be found in points

### Value

integer

### See Also

- `it.getPointIndexLowerThanTime`, `it.getPointIndexHigherThanTime`

### Examples

```
     it <- it.sample()
     it.getPointIndexNearestTime(it, 0.5)
```

---

**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

\texttt{it.getPointIndexNearestTime}, \texttt{it.read}, \texttt{it.write}, \texttt{it.plot}, \texttt{it.cut}, \texttt{it.cut0}, \texttt{it.legendre}

Examples

\begin{verbatim}
  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)
\end{verbatim}

\section*{Description}

Interpolate the IntensityTier in \texttt{npoints} equidistant points and approximate it by Legendre polynomials

Usage

\begin{verbatim}
  it.legendre(it, npoints = 1000, npolynomials = 4)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{it} \hspace{1cm} IntensityTier object
  \item \texttt{npoints} \hspace{1cm} Number of points of IntensityTier interpolation
  \item \texttt{npolynomials} \hspace{1cm} Number of polynomials to be used for Legendre modelling
\end{itemize}

Value

Vector of Legendre polynomials coefficients

See Also

\texttt{it.legendreSynth}, \texttt{it.legendreDemo}, \texttt{it.cut}, \texttt{it.cut0}, \texttt{it.read}, \texttt{it.plot}, \texttt{it.interpolate}
Examples

```r
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)
```

Description

Plots first four Legendre polynomials

Usage

```r
it.legendreDemo()
```

See Also

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

Examples

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

Description

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

Usage

```r
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

```r
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)
```

Description

Plots interactive IntensityTier using dygraphs package.

Usage

```r
it.plot(it, group = "", snd = NULL)
```

Arguments

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

See Also

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

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

## End(Not run)
```

Description

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

Usage

```r
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`, `formant.read`, `col.read`

Examples

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

## End(Not run)
```
Description

Returns sample IntensityTier.

Usage

it.sample()

Value

IntensityTier

See Also

it.plot

Examples

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

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 $t\text{min}$ and $t\text{max}$ 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

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

## End(Not run)
```

### Description
Cut the specified interval from the Pitch object and preserve time

### Usage

```r
pitch.cut(pitch, tStart = -Inf, tEnd = Inf)
```

### Arguments

- **pitch**
  - Pitch object (either in Frame or Array format)
- **tStart**
  - beginning time of interval to be cut (default \(-\infty\) = cut from the \(\text{xmin}\) of the Pitch)
- **tEnd**
  - final time of interval to be cut (default \(\infty\) = cut to the \(\text{xmax}\) of the Pitch)

### Value
- Pitch object

### See Also
- `pitch.cut0`, `tg.cut`, `tg.cut0`, `pitch.read`, `pitch.plot`

### Examples

```r
pitch <- pitch.sample()
pitch2 <- pitch.cut(pitch, tStart = 3)
pitch2_0 <- pitch.cut0(pitch, tStart = 3)
pitch3 <- pitch.cut(pitch, tStart = 2, tEnd = 3)
pitch3_0 <- pitch.cut0(pitch, tStart = 2, tEnd = 3)
pitch4 <- pitch.cut(pitch, tEnd = 1)
pitch4_0 <- pitch.cut0(pitch, tEnd = 1)
pitch5 <- pitch.cut(pitch, tStart = -1, tEnd = 1)
pitch5_0 <- pitch.cut0(pitch, tStart = -1, tEnd = 1)

## Not run:
pitch.plot(pitch)
pitch.plot(pitch2)
```
pitch.cut0


pitch.cut0(pitch2_0)
pitch.plot(pitch3)
pitch.plot(pitch3_0)
pitch.plot(pitch4)
pitch.plot(pitch4_0)
pitch.plot(pitch5)
pitch.plot(pitch5_0)

## End(Not run)

---

pitch.cut0          pitch.cut0

### Description
Cut the specified interval from the Pitch object and shift time so that the new \( \text{xmin} = 0 \)

### Usage

\[
\text{pitch.cut0}(\text{pitch}, \ tStart = -\text{Inf}, \ tEnd = \text{Inf})
\]

### Arguments

- **pitch**: Pitch object (either in Frame or Array format)
- **tStart**: beginning time of interval to be cut (default \(-\text{Inf} = \text{cut from the xmin of the Pitch}\)
- **tEnd**: final time of interval to be cut (default \(\text{Inf} = \text{cut to the xmax of the Pitch}\)

### Value
Pitch object

### See Also

- `pitch.cut`
- `tg.cut`
- `tg.cut0`
- `pitch.read`
- `pitch.plot`

### Examples

```r
pitch <- pitch.sample()
pitch2 <- pitch.cut(pitch, tStart = 3)
pitch2_0 <- pitch.cut0(pitch, tStart = 3)
pitch3 <- pitch.cut(pitch, tStart = 2, tEnd = 3)
pitch3_0 <- pitch.cut0(pitch, tStart = 2, tEnd = 3)
pitch4 <- pitch.cut(pitch, tEnd = 1)
pitch4_0 <- pitch.cut0(pitch, tEnd = 1)
pitch5 <- pitch.cut(pitch, tStart = -1, tEnd = 1)
pitch5_0 <- pitch.cut0(pitch, tStart = -1, tEnd = 1)
## Not run:
pitch.plot(pitch)
```
pitch.getPointIndexHigherThanTime

Description

Returns index of frame which is nearest the given time from right, i.e. \( \text{time} \leq \text{frameTime} \).

Usage

\[
pitch.getPointIndexHigherThanTime(pitch, \text{time})
\]

Arguments

- \text{pitch} \quad \text{Pitch object}
- \text{time} \quad \text{time which is going to be found in frames}

Value

integer

See Also

\[ \text{pitch.getPointIndexNearestTime, pitch.getPointIndexLowerThanTime} \]

Examples

\[
pitch <- pitch.sample()
pitch.getPointIndexHigherThanTime(pitch, 0.5)
\]
### pitch.getPointIndexLowerThanTime

**Description**

Returns index of frame which is nearest the given time from left, i.e. `frameTime <= time`.

**Usage**

`pitch.getPointIndexLowerThanTime(pitch, time)`

**Arguments**

- **pitch**: Pitch object
- **time**: Time which is going to be found in frames

**Value**

Integer

**See Also**

`pitch.getPointIndexNearestTime`, `pitch.getPointIndexHigherThanTime`

**Examples**

```r
pitch <- pitch.sample()
pitch.getPointIndexLowerThanTime(pitch, 0.5)
```

---

### pitch.getPointIndexNearestTime

**Description**

Returns index of frame which is nearest the given time (from both sides).

**Usage**

`pitch.getPointIndexNearestTime(pitch, time)`

**Arguments**

- **pitch**: Pitch object
- **time**: Time which is going to be found in frames

---
Value

integer

See Also

pitch.getPointIndexLowerThanTime, pitch.getPointIndexHigherThanTime

Examples

```r
pitch <- pitch.sample()
pitch.getPointIndexNearestTime(pitch, 0.5)
```

Description

Plots interactive Pitch object using dygraphs package.

Usage

```r
pitch.plot(
pitch,
scaleIntensity = TRUE,
showStrength = FALSE,
group = "",
pt = NULL
)
```

Arguments

- **pitch**: Pitch object
- **scaleIntensity**: Point size scaled according to relative intensity
- **showStrength**: Show strength annotation
- **group**: [optional] character string, name of group for dygraphs synchronization
- **pt**: [optional] PitchTier object

See Also

pitch.read, pitch.sample, pitch.toArray, tg.plot, pt.plot, formant.plot
**Examples**

```r
## Not run:
pitch <- pitch.sample()
pitch.plot(pitch, scaleIntensity = TRUE, showStrength = TRUE)
pitch.plot(pitch, scaleIntensity = TRUE, showStrength = TRUE, pt = pt.sample())
## End(Not run)
```

---

**Description**


**Usage**

```r
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.


- `$xmin`    start time (seconds)
- `$xmax`    end time (seconds)
- `$nx`      number of frames
- `$dx`      time step = frame duration (seconds)
- `$x1`      time associated with the first frame (seconds)
- `$t`       vector of time instances associated with all frames
- `$ceiling` a frequency above which a candidate is considered voiceless (Hz)
- `$maxnCandidates` maximum number of candidates in frame
- `$frame[[1]]` to `$frame[[p$nx]]` ... frames
- `$frame[[1]]$intensity` intensity of the frame
- `$frame[[1]]$nCandidates` actual number of candidates in this frame
- `$frame[[1]]$frequency` vector of candidates’ frequency (in Hz)
  (for a voiced candidate), or 0 (for an unvoiced candidate)
- `$frame[[1]]$strength` vector of degrees of periodicity of candidates (between 0 and 1)
pitch.sample

See Also

pitch.write, pitch.plot, pitch.cut, pitch.getPointIndexNearestTime, pt.read, tg.read, it.read, col.read

Examples

## Not run:
p <- pitch.read('demo/sound.Pitch')
names(p)
p$n
p$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)

description

Returns sample Pitch object.

Usage

pitch.sample()

Value

Pitch

See Also

tg.sample, pt.sample, it.sample, formant.sample

Examples

pitch <- pitch.sample()
Description

pitch.toArray

Usage

pitch.toArray(pitch)

Arguments

pitch Pitch object (frame format)

Value

Pitch object with frames converted to frequency and strength arrays and intensity vector

See Also

pitch.toFrame, pitch.read, pitch.plot

Examples

pitchArray <- pitch.toFrame(pitch.sample())
pitchArray$t[1:10]
pitchArray$frequencyArray[, 1:10]
pitchArray$bandwidthArray[, 1:10]
pitchArray$intensityVector[1:10]
Value

Pitch object with frames

See Also

`pitch.toArray, pitch.read, pitch.plot`

Examples

```r
pitchArray <- pitch.toArray(pitch.sample())
pitch <- pitch.toFrame(pitchArray)
```

Description

Saves Pitch to the file.

Usage

```
pitch.write(pitch, fileNamePitch, format = "short")
```

Arguments

- `pitch`: Pitch object
- `fileNamePitch`: Output file name
- `format`: Output file format ("short" (default, short text format) or "text" (a.k.a. full text format))

See Also

`pitch.read, pt.read`

Examples

```r
## Not run:
pitch <- pitch.sample()
pitch.write(pitch, "demo_output.Pitch")
## End(Not run)
```
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

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)
Description

Cut the specified interval from the PitchTier and shift time so that the new \( t_{\text{min}} = 0 \)

Usage

\[
\text{pt.cut0}(\text{pt}, \ t_{\text{Start}} = -\infty, \ t_{\text{End}} = \infty)
\]

Arguments

- **pt**: PitchTier object
- **tStart**: beginning time of interval to be cut (default \(-\infty = \) cut from the \( t_{\text{min}} \) of the PitchTier)
- **tEnd**: final time of interval to be cut (default \( \infty = \) cut to the \( t_{\text{max}} \) of the PitchTier)

Value

PitchTier object

See Also


Examples

```r
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.getPointIndexHigherThanTime**

Description

Returns index of point which is nearest the given time from right, i.e. time <= pointTime.

Usage

`pt.getPointIndexHigherThanTime(pt, time)`

Arguments

- **pt**: PitchTier object
- **time**: time which is going to be found in points

Value

integer

See Also

`pt.getPointIndexNearestTime`, `pt.getPointIndexLowerThanTime`

Examples

```r
pt <- pt.sample()
pt.getPointIndexHigherThanTime(pt, 0.5)
```

**pt.getPointIndexLowerThanTime**

Description

Returns index of point which is nearest the given time from left, i.e. pointTime <= time.

Usage

`pt.getPointIndexLowerThanTime(pt, time)`

Arguments

- **pt**: PitchTier object
- **time**: time which is going to be found in points

Value

integer

See Also

`pt.getPointIndexNearestTime`, `pt.getPointIndexHigherThanTime`
Value
integer

See Also
pt.getPointIndexNearestTime, pt.getPointIndexHigherThanTime

Examples
pt <- pt.sample()
pt.getPointIndexLowerThanTime(pt, 0.5)

pt.getPointIndexNearestTime

pt.getPointIndexNearestTime

Description
Returns index of point which is nearest the given time (from both sides).

Usage
pt.getPointIndexNearestTime(pt, time)

Arguments
pt PitchTier object
time time which is going to be found in points

Value
integer

See Also
pt.getPointIndexLowerThanTime, pt.getPointIndexHigherThanTime

Examples
pt <- pt.sample()
pt.getPointIndexNearestTime(pt, 0.5)
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


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

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


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)

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


Examples

```r
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)
```

Description

Plots first four Legendre polynomials

Usage

pt.legendreDemo()

See Also


Examples

```r
## Not run:
pt.legendreDemo()
## End(Not run)
```
Description

Synthetize the contour from vector of Legendre polynomials \( c \) in \( npoints \) equidistant points

Usage

\[
\text{pt.legendreSynth}(c, npoints = 1000)
\]

Arguments

- \( c \) Vector of Legendre polynomials coefficients
- \( npoints \) Number of points of PitchTier interpolation

Value

Vector of values of synthetized contour

See Also


Examples

```r
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)
\( \text{leg} \) <- pt.legendreSynth(c)
\( \text{ptLeg} \) <- pt
\( \text{ptLeg}\$t \) <- seq(ptLeg$tmin, ptLeg$tmax, length.out = length(leg))
\( \text{ptLeg}\$f \) <- 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)
```
Description

Plots interactive PitchTier using dygraphs package.

Usage

\texttt{pt.plot(pt, group = "}")

Arguments

\begin{itemize}
\item \texttt{pt} \hspace{1cm} \text{PitchTier object}
\item \texttt{group} \hspace{1cm} \text{[optional] character string, name of group for dygraphs synchronization}
\end{itemize}

See Also

\texttt{pt.read, pt.Hz2ST, pt.cut, pt.cut0, pt.interpolate, pt.write, tg.plot, pitch.plot, formant.plot}

Examples

\begin{verbatim}
## Not run:
pt <- pt.sample()
pt.plot(pt)

## End(Not run)
\end{verbatim}

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

\texttt{pt.read(fileNamePitchTier, encoding = \"UTF-8\")}

Arguments

\begin{itemize}
\item \texttt{fileNamePitchTier} \hspace{1cm} \text{file name of PitchTier}
\item \texttt{encoding} \hspace{1cm} \text{File encoding (default: \"UTF-8\"), \"auto\" for auto-detect of Unicode encoding}
\end{itemize}
Value

PitchTier object

See Also


Examples

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

## End(Not run)
```

Description

Returns sample PitchTier.

Usage

pt.sample()

Value

PitchTier

See Also

pt.plot

Examples

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

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

Usage

```r
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 $t_{\text{min}}$ and $t_{\text{max}}$ info))

See Also

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

Examples

```r
## 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)
```

Description

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

Usage

```r
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, 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
```

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)
seqM

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

<table>
<thead>
<tr>
<th>R: seqM</th>
<th>Matlab</th>
<th>R: seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>seqM(1, 3)</td>
<td>[1] 1 2 3</td>
<td>1:3</td>
</tr>
<tr>
<td>seqM(1, 3, by=.8)</td>
<td>[1] 1.0 1.8 2.6</td>
<td>1:.8:3</td>
</tr>
<tr>
<td>seqM(1, 3, by=5)</td>
<td>[1] 1</td>
<td>1:5:3</td>
</tr>
<tr>
<td>seqM(3, 1)</td>
<td>integer(0)</td>
<td>3:1</td>
</tr>
</tbody>
</table>
| 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)
Description

Cut the specified interval from the Sound object and preserve time

Usage

snd.cut(snd, Start = -Inf, End = Inf, units = "seconds")

Arguments

- **snd**: Sound object (list with $sig and $fs members at least)
- **Start**: beginning sample/time of interval to be cut (default -Inf = cut from the beginning of the Sound)
- **End**: final sample/time of interval to be cut (default Inf = cut to the end of the Sound)
- **units**: Units of Start and End arguments: "samples" (starting from 1, i.e., 1 == index of the 1st sample) or "seconds" (starting from 0)

Value

Sound object

See Also

snd.cut0, tg.cut, tg.cut0, snd.read, snd.plot

Examples

```r
snd <- snd.sample()
snd2 <- snd.cut(snd, Start = 0.3)
snd2_0 <- snd.cut0(snd, Start = 0.3)
snd3 <- snd.cut(snd, Start = 0.2, End = 0.3)
snd3_0 <- snd.cut0(snd, Start = 0.2, End = 0.3)
snd4 <- snd.cut(snd, End = 0.1)
snd4_0 <- snd.cut0(snd, End = 0.1)
snd5 <- snd.cut(snd, Start = -0.1, End = 0.1)
snd5_0 <- snd.cut0(snd, Start = -0.1, End = 0.1)
snd6 <- snd.cut(snd, End = 1000, units = "samples")
snd6_0 <- snd.cut0(snd, End = 1000, units = "samples")
```

## Not run:
snd.plot(snd)
snd.plot(snd2)
snd.plot(snd2_0)
Description

Cut the specified interval from the Sound object and shift time so that the new snd$t[1] = 0

Usage

```r
snd.cut0(snd, Start = -Inf, End = Inf, units = "seconds")
```

Arguments

- `snd`: Sound object (list with $sig and $fs members at least)
- `Start`: beginning sample/time of interval to be cut (default -Inf = cut from the beginning of the Sound)
- `End`: final sample/time of interval to be cut (default Inf = cut to the end of the Sound)
- `units`: Units of `Start` and `End` arguments: “samples” (starting from 1, i.e., 1 == index of the 1st sample) or "seconds" (starting from 0)

Value

Sound object

See Also

`snd.cut`, `tg.cut`, `tg.cut0`, `snd.read`, `snd.plot`

Examples

```r
snd <- snd.sample()
snd2 <- snd.cut(snd, Start = 0.3)
snd2_0 <- snd.cut0(snd, Start = 0.3)
snd3 <- snd.cut(snd, Start = 0.2, End = 0.3)
snd3_0 <- snd.cut0(snd, Start = 0.2, End = 0.3)
snd4 <- snd.cut(snd, End = 0.1)
snd4_0 <- snd.cut0(snd, End = 0.1)
```
Description

Returns index of sample which is nearest the given time from right, i.e. \( time \leq sampleTime \).

Usage

\[
\text{snd.getPointIndexHigherThanTime}(\text{snd}, \text{time})
\]

Arguments

\[
\begin{align*}
\text{snd} & \quad \text{Sound object} \\
\text{time} & \quad \text{time which is going to be found in samples}
\end{align*}
\]

Value

integer

See Also

\[
\text{snd.getPointIndexNearestTime, snd.getPointIndexLowerThanTime}
\]

Examples

\[
\begin{align*}
\text{snd} & \leftarrow \text{snd.sample()} \\
\text{snd.getPointIndexHigherThanTime}(\text{snd}, 0.5)
\end{align*}
\]
Description

Returns index of sample which is nearest the given time from left, i.e. sampleTime <= time.

Usage

\[
\text{snd.getPointIndexLowerThanTime(snd, time)}
\]

Arguments

- \text{snd} \quad \text{Sound object}
- \text{time} \quad \text{time which is going to be found in samples}

Value

integer

See Also

\text{snd.getPointIndexNearestTime}, \text{snd.getPointIndexHigherThanTime}

Examples

\[
\begin{align*}
& \text{snd <- snd.sample()} \\
& \text{snd.getPointIndexLowerThanTime(snd, 0.5)}
\end{align*}
\]

Description

Returns index of sample which is nearest the given \textit{time} (from both sides).

Usage

\[
\text{snd.getPointIndexNearestTime(snd, time)}
\]

Arguments

- \text{snd} \quad \text{Sound object}
- \text{time} \quad \text{time which is going to be found in samples}
Value

integer

See Also

snd.getPointIndexLowerThanTime, snd.getPointIndexHigherThanTime

Examples

```r
snd <- snd.sample()
snd.getPointIndexNearestTime(snd, 0.5)
```
Description

Loads sound file (.wav or .mp3) using tuneR package.

Usage

```r
snd.read(
  fileNameSound,
  fileType = "auto",
  from = 1,
  to = Inf,
  units = "samples"
)
```

Arguments

- `fileNameSound`: Sound file name (.wav or .mp3)
- `fileType`: "wav", "mp3" or "auto"
- `from`: Where to start reading in units (beginning "samples": 1, "seconds": 0)
- `to`: Where to stop reading in units (Inf = end of the file)
- `units`: Units of from and to arguments: "samples" (starting from 1) or "seconds" (starting from 0)

Value

Sound object with normalized amplitude (PCM / 2^(nbits-1) - 1) resulting to the range of [-1; +1]. In fact, the minimum value can be one quantization step lower (e.g. PCM 16bit: -32768).

- `t` ... vector of discrete time instances (seconds)
- `sig` ... signal matrix (nrow(snd$sig) = number of samples, ncol(snd$sig) = number of channels, i.e., $sig[,1] ... 1st channel)
- `fs` ... sample rate (Hz)
- `nChannels` ... number of signal channels (ncol(snd$sig)), 1 == mono, 2 == stereo
- `nBits` ... number of bits per one sample
- `nSamples` ... number of samples (nrow(snd$sig))
- `duration` ... duration of signal (seconds), snd$duration == snd$nSamples/snd$fs

See Also

- `snd.write`, `snd.plot`, `snd.cut`, `snd.getPointIndexNearestTime`

Examples

```r
## Not run:
snd <- snd.read("demo/H.wav")
snd.plot(snd)

## End(Not run)
```
Description

Returns sample Sound object.

Usage

snd.sample()

Value

snd

See Also

snd.plot

Examples

snd <- snd.sample()
snd.plot(snd)

snd.write

Description

Saves Sound object to a file. snd is a list with $\text{sig}$ and $\text{fs}$ members at least. If $\text{nBits}$ is not present, default value of 16 bits is used. Vector $t$ is ignored. If the sound signal is 2-channel (stereo), $\text{sig}$ must be a two-column matrix (1st column corresponds to the left channel, 2nd column to the right channel). If the sound is 1-channel (mono), $\text{sig}$ can be either a numeric vector or a one-column matrix. optional $t$, $\text{nChannels}$, $\text{nSamples}$, $\text{duration}$ vectors are ignored.

Usage

snd.write(snd, fileNameSound)

Arguments

snd Sound object (with $\text{sig}$, $\text{nBits}$ and $\text{fs}$ members)
fileNameSound file name to be created

See Also

snd.read
strTrim

Examples

```r
## Not run:
snd <- snd.sample()
snd.plot(snd)
snd.write(snd, "temp1.wav")

signal <- 0.8*sin(seq(0, 2*pi*440, length.out = 8000))
snd.write(list(sig = signal, fs = 8000, nBits = 16), "temp2.wav")

left <- 0.3*sin(seq(0, 2*pi*440, length.out = 4000))
right <- 0.5*sin(seq(0, 2*pi*220, length.out = 4000))
snd.write(list(sig = matrix(c(left, right), ncol = 2), fs = 8000, nBits = 16), "temp3.wav")

## End(Not run)
```

---

strTrim

**Description**

Trim leading and trailing whitespace in character string.

**Usage**

```r
strTrim(string)
```

**Arguments**

- `string`: character string

**Details**

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


**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 without regexp.

**Examples**

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

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

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

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 expressions
\textit{str\_find1}

\textbf{Value}

indices of all occurrences (1 = 1st character)

\textbf{See Also}

\textit{str\_find1, str\_contains, isString}

\textbf{Examples}

\begin{verbatim}
str_find("Hello, hello, hello world", "ell")  # 2 9 16
str_find("Hello, hello, hello world", "q")  # integer(0)
\end{verbatim}

---

\textit{str\_find1} \quad \textit{str\_find1}

\textbf{Description}

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

\textbf{Usage}

\texttt{str\_find1(string, patternNoRegex)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{string} \quad string in which we try to find something
  \item \texttt{patternNoRegex} \quad string we want to find, "as it is" - no regular expressions
\end{itemize}

\textbf{Value}

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

\textbf{See Also}

\textit{str\_find, str\_contains, isString}

\textbf{Examples}

\begin{verbatim}
str_find1("Hello, hello, hello world", "ell")  # 2
str_find1("Hello, hello, hello world", "q")    # integer(0)
\end{verbatim}
Description

Aligns boundaries of intervals in the target tier (typically: "word") to the closest boundaries in the pattern tier (typically: "phone"). If there is no boundary within the tolerance limit in the pattern tier, the boundary position in the target tier is kept at its original position.

Usage

tg.boundaryMagnet(
  tg,
  targetTier,
  patternTier,
  boundaryTolerance = Inf,
  verbose = TRUE
)

Arguments

tg
  TextGrid object

targetTier
  index or "name" of the tier to be aligned

patternTier
  index or "name" of the pattern tier

boundaryTolerance
  if there is not any boundary in the pattern tier within this tolerance, the target boundary is kept at its position [default: Inf]

verbose
  if TRUE, every boundary shift is printed [default: TRUE]

Value

TextGrid object

See Also

tg.insertBoundary, tg.insertInterval, tg.duplicateTier

Examples

## Not run:
tg <- tg.sample()
tg <- tg.removeTier(tg, "phoneme")
tg <- tg.removeTier(tg, "syllable")
tg <- tg.removeTier(tg, "phrase")

# garble times in "word" tier a little
n <- length(tg\$word\$label)
deltaT <- runif(n - 1, min = -0.01, max = 0.015)
tg$word$t2[1: (n-1)] <- tg$word$t2[1: (n-1)] + deltaT
tg$word$t1[2: n] <- tg$word$t2[1: (n-1)]
tg.plot(tg)

# align "word" tier according to "phone tier"
tg2 <- tg.boundaryMagnet(tg, targetTier = "word", patternTier = "phone")
tg.plot(tg2)

## 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**

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**

Description

Creates new and empty TextGrid. tMin and tMax 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 <- tg.cut()

describe

Cut the specified time frame from the TextGrid and preserve time

Usage

tg.cut(tg, tStart = -Inf, tEnd = Inf)

Arguments

tg         TextGrid object
	tStart    beginning time of time frame to be cut (default -Inf = cut from the tmin of the
            TextGrid)
	tEnd      final time of time frame to be cut (default Inf = cut to the tmax of the TextGrid)

Value

TextGrid object
See Also
tg.cut0, pt.cut, pt.cut0, tg.read, tg.plot, tg.write, tg.insertInterval

Examples

tg <- tg.sample()
tg2 <- tg.cut(tg, tStart = 3)
tg2_0 <- tg.cut0(tg, tStart = 3)
tg3 <- tg.cut(tg, tStart = 2, tEnd = 3)
tg3_0 <- tg.cut0(tg, tStart = 2, tEnd = 3)
tg4 <- tg.cut(tg, tEnd = 1)
tg4_0 <- tg.cut0(tg, tEnd = 1)
tg5 <- tg.cut(tg, tStart = -1, tEnd = 5)
tg5_0 <- tg.cut0(tg, tStart = -1, tEnd = 5)
## Not run:
tg.plot(tg)
tg.plot(tg2)
tg.plot(tg2_0)
tg.plot(tg3)
tg.plot(tg3_0)
tg.plot(tg4)
tg.plot(tg4_0)
tg.plot(tg5)
tg.plot(tg5_0)
## End(Not run)

tg.cut0

Description

Cut the specified time frame from the TextGrid and shift time so that the new tmin = 0

Usage

tg.cut0(tg, tStart = -Inf, tEnd = Inf)

Arguments

tg TextGrid object
tStart beginning time of time frame to be cut (default -Inf = cut from the tmin of the TextGrid)
tEnd final time of time frame to be cut (default Inf = cut to the tmax of the TextGrid)

Value

TextGrid object
See Also
tg.cut, pt.cut, pt.cut0, tg.read, tg.plot, tg.write, tg.insertInterval

Examples

tg <- tg.sample()
tg2 <- tg.cut(tg, tStart = 3)
tg2_0 <- tg.cut0(tg, tStart = 3)
tg3 <- tg.cut(tg, tStart = 2, tEnd = 3)
tg3_0 <- tg.cut0(tg, tStart = 2, tEnd = 3)
tg4 <- tg.cut(tg, tEnd = 1)
tg4_0 <- tg.cut0(tg, tEnd = 1)
tg5 <- tg.cut(tg, tStart = -1, tEnd = 5)
tg5_0 <- tg.cut0(tg, tStart = -1, tEnd = 5)

## Not run:
tg.plot(tg)
tg.plot(tg2)
tg.plot(tg2_0)
tg.plot(tg3)
tg.plot(tg3_0)
tg.plot(tg4)
tg.plot(tg4_0)
tg.plot(tg5)
tg.plot(tg5_0)

## End(Not run)

tg.duplicateTier
tg.duplicateTier

duplicateTier

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
tg.duplicateTierMergeSegments

Value

TextGrid object

See Also

tg.duplicateTierMergeSegments, tg.setTierName, tg.removeTier, tg.boundaryMagnet

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-11o-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**

```r
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 = "-")
```

```r
## Not run:
tg.plot(tg)
tg.plot(tg2)
## End(Not run)
```

---

**Description**

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

**Usage**

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tg</code></td>
<td>TextGrid object</td>
</tr>
<tr>
<td><code>tierInd</code></td>
<td>tier index or &quot;name&quot;</td>
</tr>
<tr>
<td><code>labelVector</code></td>
<td>character string (one label) or vector of character strings (consecutive sequence of labels) to be found</td>
</tr>
<tr>
<td><code>returnTime</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

Value

If `returnTime == FALSE`, returns list of all occurrences, each member of the list is one occurrence and contains vector of label indices, if `returnTime == TRUE`, returns list with 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()
tg.getEndTime

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

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

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
tg.getIntervalIndexAtTime

Value
numeric

See Also
tg.getIntervalStartTime, tg.getIntervalsDuratin, tg.getIntervalsIndexAtTime, tg.findLabels

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

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.getIntervalsEndTime, tg.getLabel, tg.findLabels

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

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
tg.getNumberofIntervals

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")
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)

tg.getNumberOfTiers

Description

Returns number of tiers.

Usage

tg.getNumberOfTiers(tg)

Arguments

tg TextGrid object

Value

integer
tg.getPointIndexHigherThanTime

See Also
tg.getTierName, tg.isIntervalTier, tg.isPointTier

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

tg.getPointIndexHigherThanTime(tg, "phoneme", 0.5)
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, tierInd, time)

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)
tg.getPointTime

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)

---

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

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

Description

Returns name of the tier.

Usage

tg.getTierName(tg, tierInd)

Arguments

tg            TextGrid object

tierInd      tier index or "name"
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")
**Description**

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

**Usage**

```
 tg.insert Boundary(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 split 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.boundaryMagnet`, `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.boundaryMagnet, 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

```r
## 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)
```

---

### Description

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

### Usage

```r
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

```r
## 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

```r
## 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

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

Returns TRUE if the tier is PointTier, FALSE otherwise.

**Usage**

```r
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**

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

---

**Description**

Plots interactive TextGrid using dygraphs package.

**Usage**

```r
tg.plot(
  tg,
  group = "",
  pt = NULL,
  it = NULL,
  formant = NULL,
  formantScaleIntensity = TRUE,
  
```
tg.plot

formantDrawBandwidth = TRUE,
pitch = NULL,
pitchScaleIntensity = TRUE,
pitchShowStrength = FALSE,
snd = NULL
)

Arguments

tg TextGrid object

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

pt [optional] PitchTier object

it [optional] IntensityTier object

formant [optional] Formant object

formantScaleIntensity [optional] Point size scaled according to relative intensity

formantDrawBandwidth [optional] Draw formant bandwidth

pitch [optional] Pitch object

pitchScaleIntensity [optional] Point size scaled according to relative intensity

pitchShowStrength [optional] Show strength annotation

snd [optional] Sound object

See Also
tg.read, pt.plot, it.plot, pitch.plot

Examples

## Not run:
tg <- tg.sample()
tg.plot(tg)
tg.plot(tg.sample(), pt = pt.sample())

## End(Not run)
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.duplicateTierMergeSegments, pt.read, pitch.read, formant.read, it.read, col.read

examples

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

## end(not run)
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 1-2-3. 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

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

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

## End(Not run)
```
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 NumberOfPoints, tg.removeIntervalBothBoundaries

Examples

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

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

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

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

```r
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

```r
## 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
tg.sampleProblem
tg.sampleProblem

Description
Returns sample TextGrid with continuity problem.

Usage
tg.sampleProblem()

Value
TextGrid
tg.setLabel

See Also

tg.repairContinuity

Examples

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

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)
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)

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

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

## End(Not run)
```
Index

as.formant, 4
as.it, 5
as.pitch, 5
as.pt, 6
as.snd, 6
as.tg, 7
ceiling, 52
col.read, 8, 9, 16, 30, 38, 50, 92
col.write, 9
Conj, 19
detectEncoding, 10
fft, 19
floor, 52
formant.cut, 10, 12, 16
formant.cut0, 11, 11
formant.getPointIndexHigherThanTime, 12, 13, 14
formant.getPointIndexLowerThanTime, 13, 13, 14
formant.getPointIndexNearestTime, 13, 14, 16
formant.plot, 11, 12, 14, 16–18, 36, 49
formant.read, 8, 11, 12, 15, 17, 18, 30, 50, 92
formant.sample, 15, 16, 38
formant.toArray, 15, 17, 18
formant.toFrame, 17
formant.write, 16, 18
ifft, 19, 53
Im, 19
isInt, 19, 20–22, 53
isLogical, 20, 20, 21, 22
isNum, 20, 21, 22, 53
isString, 20, 21, 22, 61–63
it.cut, 22, 24, 27, 29, 30
it.cut0, 23, 23, 27, 29, 30
it.getPointIndexHigherThanTime, 24, 25, 26
it.getPointIndexLowerThanTime, 25, 25, 26
it.getPointIndexNearestTime, 25, 26, 27
it.interpolate, 23, 24, 26, 27–31
it.legendre, 23, 24, 27, 28, 29
it.legendreDemo, 23, 24, 27, 28, 29
it.legendreSynth, 23, 24, 27, 28, 28
it.plot, 23, 24, 27–29, 29, 30, 31, 91
it.read, 8, 16, 23, 24, 27–29, 30, 31, 38, 50, 92
it.sample, 16, 31, 38
it.write, 27, 29, 30, 31
Mod, 19
pitch.cut, 32, 33, 38
pitch.cut0, 32, 33
pitch.getPointIndexHigherThanTime, 34, 35, 36
pitch.getPointIndexLowerThanTime, 34, 35, 36
pitch.getPointIndexNearestTime, 34, 35, 36
pitch.plot, 32, 33, 36, 38–40, 49, 91
pitch.read, 8, 16, 30, 32, 33, 36, 37, 39, 40, 50, 92
pitch.sample, 16, 36, 38
pitch.toArray, 36, 39, 40
pitch.toFrame, 39, 39
pitch.write, 38, 40
pt.cut, 41, 42, 45–47, 49, 50, 68, 69
pt.cut0, 41, 42, 45–47, 49, 50, 68, 69
pt.getPointIndexHigherThanTime, 43, 44
pt.getPointIndexLowerThanTime, 43, 43, 44
pt.getPointIndexNearestTime, 43, 44, 44, 46
pt.Hz2ST, 41, 42, 45, 46–51
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt.interpolate</td>
<td>41, 42, 45, 45, 47–51</td>
</tr>
<tr>
<td>pt.legendre</td>
<td>41, 42, 46, 46, 47, 48, 50</td>
</tr>
<tr>
<td>pt.legendreDemo</td>
<td>41, 42, 47, 47, 48</td>
</tr>
<tr>
<td>pt.legendreSynth</td>
<td>41, 42, 47, 48</td>
</tr>
<tr>
<td>pt.plot</td>
<td>36, 41, 42, 45–48, 49, 50, 91</td>
</tr>
<tr>
<td>pt.read</td>
<td>8, 16, 30, 38, 40–42, 45–49, 49, 51, 92</td>
</tr>
<tr>
<td>pt.sample</td>
<td>16, 38, 50</td>
</tr>
<tr>
<td>pt.write</td>
<td>45, 46, 49, 50, 51, 101</td>
</tr>
<tr>
<td>Re</td>
<td>19</td>
</tr>
<tr>
<td>round</td>
<td>52</td>
</tr>
<tr>
<td>round2</td>
<td>51, 53</td>
</tr>
<tr>
<td>seqM</td>
<td>52</td>
</tr>
<tr>
<td>snd.cut</td>
<td>54, 55, 59</td>
</tr>
<tr>
<td>snd.cut0</td>
<td>54, 55</td>
</tr>
<tr>
<td>snd.getPointIndexHigherThanTime</td>
<td>56, 57, 58</td>
</tr>
<tr>
<td>snd.getPointIndexLowerThanTime</td>
<td>56, 57, 58</td>
</tr>
<tr>
<td>snd.getPointIndexNearestTime</td>
<td>56, 57, 57, 59</td>
</tr>
<tr>
<td>snd.plot</td>
<td>54, 55, 58, 59, 60</td>
</tr>
<tr>
<td>snd.read</td>
<td>54, 55, 58, 59, 60</td>
</tr>
<tr>
<td>snd.sample</td>
<td>60</td>
</tr>
<tr>
<td>snd.write</td>
<td>59, 60</td>
</tr>
<tr>
<td>str_contains</td>
<td>61, 62, 63</td>
</tr>
<tr>
<td>str_find</td>
<td>61, 62, 63</td>
</tr>
<tr>
<td>str_findI</td>
<td>61–63, 63</td>
</tr>
<tr>
<td>strTrim</td>
<td>61</td>
</tr>
<tr>
<td>tg.getBoundaryMagnet</td>
<td>64, 70, 84, 86</td>
</tr>
<tr>
<td>tg.checkTierInd</td>
<td>65</td>
</tr>
<tr>
<td>tg.countLabels</td>
<td>66, 72, 77</td>
</tr>
<tr>
<td>tg.createNewTextGrid</td>
<td>66, 92</td>
</tr>
<tr>
<td>tg.cut</td>
<td>11, 12, 32, 33, 41, 54, 55, 67, 69</td>
</tr>
<tr>
<td>tg.cut0</td>
<td>11, 12, 32, 33, 41, 54, 55, 68, 68</td>
</tr>
<tr>
<td>tg.duplicateTier</td>
<td>64, 69, 71, 87, 88, 97</td>
</tr>
<tr>
<td>tg.duplicateTierMergeSegments</td>
<td>70, 70, 72, 84, 86, 87, 92</td>
</tr>
<tr>
<td>tg.findLabels</td>
<td>66, 71, 74–77, 79–81, 89, 90, 92</td>
</tr>
<tr>
<td>tg.getEndTime</td>
<td>73, 82, 83</td>
</tr>
<tr>
<td>tg.getIntervalDuration</td>
<td>74, 75, 76</td>
</tr>
<tr>
<td>tg.getIntervalEndTime</td>
<td>74, 74, 75, 76</td>
</tr>
<tr>
<td>tg.getIntervalIndexAtTime</td>
<td>74, 75, 75, 76</td>
</tr>
<tr>
<td>tg.getIntervalStarttime</td>
<td>74, 75, 76</td>
</tr>
<tr>
<td>tg.getLabel</td>
<td>66, 72, 73, 76, 79–81, 99</td>
</tr>
<tr>
<td>tg.getNumberOfIntervals</td>
<td>77, 78</td>
</tr>
<tr>
<td>tg.getNumberOfPoints</td>
<td>77, 78, 96</td>
</tr>
<tr>
<td>tg.getNumberOfTiers</td>
<td>65, 78</td>
</tr>
<tr>
<td>tg.getPointIndexHigherThanTime</td>
<td>79, 80, 81</td>
</tr>
<tr>
<td>tg.getPointIndexLowerThanTime</td>
<td>79, 80, 81</td>
</tr>
<tr>
<td>tg.getPointIndexNearestTime</td>
<td>79, 80, 80, 81</td>
</tr>
<tr>
<td>tg.getPointTime</td>
<td>81</td>
</tr>
<tr>
<td>tg.getStartTime</td>
<td>73, 82, 83</td>
</tr>
<tr>
<td>tg.getTierName</td>
<td>65, 79, 82, 89, 90, 100</td>
</tr>
<tr>
<td>tg.getTotalDuration</td>
<td>73, 82, 83</td>
</tr>
<tr>
<td>tg.insertBoundary</td>
<td>64, 84, 86, 89, 93–95</td>
</tr>
<tr>
<td>tg.insertInterval</td>
<td>64, 68, 69, 84, 85, 87, 89, 93–95</td>
</tr>
<tr>
<td>tg.insertNewIntervalTier</td>
<td>67, 86, 88, 97</td>
</tr>
<tr>
<td>tg.insertNewPointTier</td>
<td>67, 87, 87, 97</td>
</tr>
<tr>
<td>tg.insertPoint</td>
<td>88, 88, 96</td>
</tr>
<tr>
<td>tg.isIntervalTier</td>
<td>65, 79, 83, 89, 90</td>
</tr>
<tr>
<td>tg.isPointTier</td>
<td>65, 79, 83, 89, 90</td>
</tr>
<tr>
<td>tg.plot</td>
<td>13, 29, 36, 49, 65, 68, 69, 90, 92, 98</td>
</tr>
<tr>
<td>tg.read</td>
<td>8, 16, 18, 30, 38, 50, 68, 69, 91, 92, 101</td>
</tr>
<tr>
<td>tg.removeIntervalBothBoundaries</td>
<td>84, 86, 92, 94–96</td>
</tr>
<tr>
<td>tg.removeIntervalLeftBoundary</td>
<td>84, 86, 93, 94, 95</td>
</tr>
<tr>
<td>tg.removeIntervalRightBoundary</td>
<td>84, 86, 93, 94, 95</td>
</tr>
<tr>
<td>tg.removePoint</td>
<td>89, 96</td>
</tr>
<tr>
<td>tg.removeTier</td>
<td>70, 71, 87, 88, 96</td>
</tr>
<tr>
<td>tg.repairContinuity</td>
<td>92, 97, 99</td>
</tr>
<tr>
<td>tg.sample</td>
<td>16, 38, 98</td>
</tr>
<tr>
<td>tg.sampleProblem</td>
<td>97, 98</td>
</tr>
<tr>
<td>tg.setLabel</td>
<td>77, 99</td>
</tr>
<tr>
<td>tg.setTierName</td>
<td>70, 71, 83, 100</td>
</tr>
<tr>
<td>tg.write</td>
<td>31, 51, 68, 69, 92, 100</td>
</tr>
<tr>
<td>trunc</td>
<td>52</td>
</tr>
</tbody>
</table>