Package 'tabr'

May 18, 2018

Version 0.2.0

Title Create Guitar Tablature

Description Creates guitar tablature from R code by providing functions for describing and organizing musical structures and wrapping around the 'LilyPond' backend (<http://lilypond.org>). 'LilyPond' is open source music engraving software for generating high quality sheet music based on markup language. 'tabr' provides a wrapper around this software and generates files following the 'LilyPond' markup syntax to be subsequently processed by 'LilyPond' into sheet music pdf files.
A standalone 'LilyPond' file can be created or the package can make a system call to 'LilyPond' directly to render the guitar tablature output.
While 'LilyPond' caters to sheet music in general, 'tabr' is focused on leveraging it specifically for creating quality guitar tablature.
'tabr' offers nominal MIDI file support in addition to its focus on tablature transcription. See the 'tuneR' package for more general use of MIDI files in R.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

ByteCompile true

URL https://github.com/leonawicz/tabr

BugReports https://github.com/leonawicz/tabr/issues

Depends R (>= 3.4.0)

SystemRequirements LilyPond

Suggests testthat, knitr, rmarkdown, covr, kableExtra, lintr

Imports magrittr, dplyr, purrr

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Matthew Leonawicz [aut, cre]

Maintainer Matthew Leonawicz <mfleonawicz@alaska.edu>

Repository CRAN

Date/Publication 2018-05-17 22:13:22 UTC

R topics documented:

`append_phrases` *Append and duplicate*

Description

Helper functions for appending or pasting musical phrases and other strings together as well as repetition. The functions `glue` and `dup` are based on base functions `paste` and `rep`, respectively, but are tailored for efficiency in creating musical phrases. These functions respect and retain the phrase class when applied to phrases. Combining a phrase with a non-phrase string will assume compatibility and result in a new phrase object.

Usage

```
glue(...)

dup(x, n = 1)
```

Arguments

...	character, phrase or non-phrase string.
x	character, phrase or non-phrase string.
n	integer, number of repetitions.

Details

This is especially useful for repeated instances. This function applies to general slur notation as well. Multiple input formats are allowed. Total number of note durations must be even because all slurs require start and stop points.

Value

phrase on non-phrase character string.

Examples

```
glue(8, "16-", "8^")
dup(1, 2)
x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
y <- phrase("a", 1, 5)
glue(x, y)
glue(x, dup(y, 2))
glue(x, "r1") # add a simple rest instance
class(glue(x, y))
class(dup(y, 2))
class(glue(x, "r1"))
class(dup("r1", 2))
class(glue("r1", "r4"))
```

---

chord_set *Generate a chord set*

Description

Generate a chord set for a music score.

Usage

```
chord_set(x, id = NULL)
```

Arguments

x	character, n-string chord description from lowest to highest pitch, strings n through 1. E.g., "xo221o". See details.
id	character, the name of the chord in LilyPond readable format, e.g., "a:m". Ignored if x is already a named vector.

Details

The chord set list returned by `chord_set` is only used for top center placement of a full set of chord fretboard diagrams for a music score. `chord_set` returns a named list. The names are the chords and the list elements are strings defining string and fret fingering readable by LilyPond. Multiple chord positions can be defined for the same chord name. Instruments with a number of strings other than six are not currently supported.

When defining chords, you may also wish to define rests or silence for chords added to a score for placement above the staff in time, where no chord is to be played. Therefore, there are occasions where you may pass chord names and positions that happen to include entries `r` and/or `s` as `NA` as shown in the example. These two special cases are passed through by `chord_set` but are ignored when the chord chart is generated.

Value

a named list.

Examples

```
chord_names <- c("e:m", "c", "d", "e:m", "d", "r", "s")
chord_positions <- c("xx997x", "x5553x", "x7775x", "ooo22o", "232oxx", NA, NA)
chord_set(chord_positions, chord_names)
```

---

hp *Hammer ons and pull offs*

Description

Helper function for generating hammer on and pull off syntax.

Usage

```
hp(...)
```

Arguments

...	character, note durations. Numeric is allowed for lists of single inputs. See examples.

Details

This is especially useful for repeated instances. This function applies to general slur notation as well. Multiple input formats are allowed. Total number of note durations must be even because all slurs require start and stop points.

Value

character.

Examples

```
hp(16, 16)
hp("16 16")
hp("16 8 16", "8 16 8")
```

---

keys *Key signatures*

Description

Helper functions for key signature information.

Usage

```
keys(type = c("all", "sharp", "flat"))

is_natural(key)

is_sharp(key)

is_flat(key)

n_sharps(key)

n_flats(key)

is_major(key)

is_minor(key)
```

Arguments

type	character, defaults to "all".
key	character, key signature.

Details

The keys function returns a vector of valid key signature IDs. These IDs are how key signatures are specified throughout tabr, including in the other helper functions here via key. Like the other functions here, is_sharp and is_flat are for *key signatures*, not single pitches whose sharp or flat status is always self-evident from their notation. Major and minor keys are also self-evident from their notation, but is_major and is_minor can still be useful when programming.

Value

character vector.

Examples

```
keys()
is_natural(c("c", "am", "c#"))
x <- c("a", "e_")
is_sharp(x)
is_flat(x)
n_sharps(x)
n_flats(x)
```

`lilypond` *Save score to LilyPond file*

Description

Write a score to a LilyPond format (`.ly`) text file for later use by LilyPond or subsequent editing outside of R.

Usage

```
lilypond(score, file, key = "c", time = "4/4", tempo = "2 = 60",
  header = NULL, string_names = NULL, paper = NULL, endbar = TRUE,
  midi = TRUE, path = NULL)
```

Arguments

`score`	a score object.
`file`	character, LilyPond output file ending in `.ly`. May include an absolute or relative path.
`key`	character, key signature, e.g., `c`, `b_`, `f#m`, etc.
`time`	character, defaults to `"4/4"`.
`tempo`	character, defaults to `"2 = 60"`.
`header`	a named list of arguments passed to the header of the LilyPond file. See details.
`string_names`	label strings at beginning of tab staff. `NULL` (default) for non-standard tunings only, `TRUE` or `FALSE` for force on or off completely.
`paper`	a named list of arguments for the LilyPond file page layout. See details.
`endbar`	character, the end bar.
`midi`	logical, add midi inclusion specification to LilyPond file.
`path`	character, optional output directory prefixed to `file`, may be an absolute or relative path. If `NULL` (default), only `file` is used.

Details

All header list elements are character strings. The options for header include:

- title
- subtitle
- composer
- album
- arranger
- instrument
- meter
- opus
- piece
- poet
- copyright
- tagline

All paper list elements are numeric except page_numbers, which is logical. The options for paper include:

- textheight
- linewidth
- indent
- first_page_number
- page_numbers
- fontsize

Value

nothing returned; a file is written.

See Also

tab, midily,

Examples

```
x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
x <- track(x)
x <- score(x)
outfile <- file.path(tempdir(), "out.ly")
lilypond(x, outfile)
```

`midily` *Convert MIDI to LilyPond file*

Description

Convert a MIDI file (`.mid`) to a LilyPond format (`.ly`) text file.

Usage

```
midily(midi_file, file, key = "c", absolute = FALSE, quantize = NULL,
  explicit = FALSE, start_quant = NULL, allow_tuplet = c("4*2/3", "8*2/3",
  "16*2/3"), details = FALSE, lyric = FALSE, path = NULL)
```

Arguments

`midi_file`	character, MIDI file (`.mid`). May include an absolute or relative path.
`file`	LilyPond output file ending in `.ly`.
`key`	key signature, defaults to `"c"`.
`absolute`	logical, print absolute pitches.
`quantize`	integer, duration, quantize notes on duration.
`explicit`	logical, print explicit durations.
`start_quant`	integer, duration, quantize note starts on the duration.
`allow_tuplet`	character vector, allow tuplet durations. See details.
`details`	logical, verbose detail.
`lyric`	logical, treat all text as lyrics.
`path`	character, optional output directory prefixed to `file`, may be an absolute or relative path. If `NULL` (default), only `file` is used.

Details

Under development/testing. See warning and details below.

This function is a wrapper around the `midi2ly` command line utility provided by LilyPond. It inherits all the limitations thereof. LilyPond is not intended to be used to produce meaningful sheet music from arbitrary MIDI files. *A future version will offer additional arguments that use* `tabr` *to subsequently edit the generated LilyPond file as a second step, allowing the user to make some nominal substitutions or additions to the default output.* While `lilypond` converts R code `score` objects to LilyPond markup directly, MIDI conversion to LilyPond markup by `midily` requires LilyPond.

WARNING: Even though the purpose of the command line utility is to convert an existing MIDI file to a LilyPond file, it nevertheless generates a LilyPond file that *specifies inclusion of MIDI output*. This means when you subsequently process the LilyPond file with LilyPond or if you use `miditab` to go straight from your MIDI file to pdf output, the command line tool will also produce a MIDI file output. It will overwrite your original MIDI file if it has the same file name and location! The

next version of this function will add an default argument `midi_out = FALSE` to remove this from the generated LilyPond file. If `TRUE` and the basename of `midi_file` matches the basename of `file`, then `file` will be renamed, the basename appended with a `-1`.

`allow_tuplets = NULL` to disallow all tuplets. Fourth, eighth and sixteenth note triplets are allowed. The format is a character vector where each element is `duration*numerator/denominator`, no spaces. See default argument.

On Windows systems, it may be necessary to specify a path in tabr_options to both `midi2ly` and `python` if they are not already successfully set as follows. On package load, `tabr` will attempt to check for `midi2ly.exe` at `C:/Program Files (x86)/LilyPond/usr/bin/midi2ly.py` and similarly for the `python.exe` that ships with LilyPond at `C:/Program Files (x86)/LilyPond/usr/bin/python.exe`. If this is not where LilyPond is installed, then LilyPond and Python need to be provided to `tabr_options` or added to the system PATH variable.

Value

nothing returned; a file is written.

See Also

`miditab`, `tab`, `lilypond`

Examples

```
if(tabr_options()$midi2ly != ""){
  midi <- system.file("example.mid", package = "tabr")
  outfile <- file.path(tempdir(), "out.ly")
  midily(midi, outfile) # requires LilyPond installation
}
```

---

miditab *Convert MIDI to tablature*

Description

Convert a MIDI file to sheet music/guitar tablature.

Usage

```
miditab(midi_file, file, keep_ly = FALSE, path = NULL, details = TRUE,
  ...)
```

Arguments

`midi_file`	character, MIDI file (`.mid`). May include an absolute or relative path.
`file`	character, output file ending in .pdf or .png.
`keep_ly`	logical, keep LilyPond file.

path	character, optional output directory prefixed to file, may be an absolute or relative path. If NULL (default), only file is used.
details	logical, set to FALSE to disable printing of log output to console.
...	additional arguments passed to midily.

Details

Under development/testing. See warning and details below.

Convert a MIDI file to a pdf or png music score using the LilyPond music engraving program. Output format is inferred from file extension. This function is a wrapper around midily, the function that converts the MIDI file to a LilyPond (.ly) file using a LilyPond command line utility.

WARNING: Even though the purpose of the command line utility is to convert an existing MIDI file to a LilyPond file, it nevertheless generates a LilyPond file that *specifies inclusion of MIDI output*. This means when you subsequently process the LilyPond file with LilyPond or if you use miditab to go straight from your MIDI file to pdf output, the command line tool will also produce a MIDI file output. It will overwrite your original MIDI file if it has the same file name and location! The next version of this function will add an default argument midi_out = FALSE to remove this from the generated LilyPond file. If TRUE and the basename of midi_file matches the basename of file, then file will be renamed, the basename appended with a -1.

On Windows systems, it may be necessary to specify a path in tabr_options to both midi2ly and python if they are not already successfully set as follows. On package load, tabr will attempt to check for midi2ly.exe at C:/Program Files (x86)/LilyPond/usr/bin/midi2ly.py and similarly for the python.exe that ships with LilyPond at C:/Program Files (x86)/LilyPond/usr/bin/python.exe. If this is not where LilyPond is installed, then LilyPond and Python need to be provided to tabr_options or added to the system PATH variable.

Value

nothing returned; a file is written.

See Also

midily, tab, lilypond

Examples

```
if(tabr_options()$midi2ly != ""){
  midi <- system.file("example.mid", package = "tabr")
  outfile <- file.path(tempdir(), "out.pdf")
  miditab(midi, outfile, details = FALSE) # requires LilyPond installation
}
```

notate *Add text to music staff*

Description

Annotate a music staff, vertically aligned above or below the music staff at a specific note/time.

Usage

```
notate(x, text, position = "top")
```

Arguments

x	character.
text	character.
position	character, top or bottom.

Details

This function binds text annotation in LilyPond syntax to a note's associated `info` entry. Technically, the syntax is a hybrid form, but is later updated safely and unambiguously to LilyPond syntax with respect to the rest of the note info substring when it is fed to `phrase` for musical phrase assembly.

Value

a character string.

Examples

```
notate("8", "Solo")
phrase("c'~ c' d' e'", glue(notate(8, "First solo"), "8 8 4."), "5 5 5 5")
```

phrase *Create a musical phrase*

Description

Create a musical phrase from character strings that define notes, note metadata, and optionally explicit strings fretted. The latter can be used to ensure proper tablature layout. Notes separated in time are separated in the `notes` string by spaces. Sharps and flats are indicated by appending # and _, respectively, e.g. `f#` or `g_`.

Usage

```
phrase(notes, info, string = NULL, bar = FALSE)

p(...)
```

Arguments

notes	character, notes a through g. See details.
info	character, metadata pertaining to the notes . See details.
string	character, optional string that specifies which guitar strings to play for each specific note.
bar	logical, insert a bar check at the end of the phrase.
...	arguments passed to phrase.

Details

Specifying notes that are one or multiple octaves below or above the middle can be done by appending one or multiple commas or single quote tick marks, respectively, e.g. c, or c''. But this is not necessary. Instead, you can use octave numbering. This may easier to read, generally more familiar, potentially requires less typing, can still be omitted completely for the middle octave (no need to type c3, d3, ...), and is automatically converted for you by phrase to the tick mark format interpreted by LilyPond. That said, using the raised and lowered tick mark approach can be surprisingly easier to read for chords, which have no spaces between notes, especially six-string chords, given that the tick marks help break up the notes in the chord visually much more so than integers do. See examples.

The function p is a convenient shorthand wrapper for phrase.

Tied notes indicated by ~ are part of the note notation and not part of the info notation, e.g. c''~.

Notes can comprise chords. These are bound tightly rather than space-delimited, as they are not separated in time. For example, a C chord could be given as ceg and in the case of tied notes would be c~e~g~.

Other information about a note is indicated with the info string. The most pertinent information, minimally required, is the note duration. A string of space-delimited notes will always be accompanied by a space-delimited string of an equal number of integer durations. Durations are powers of 2: 1, 2, 4, 8, 16, 32, 64. They represent the fraction of a measure, e.g., 2 means 1/2 of a measure and 8 refers to an eighth note. Dotted notes are indicated by adding . immediately after the integer, e.g., 2. or 2.. Any other note metadata is appended to these durations. See examples.

Opening and closing slurs (or hammer ons and pull offs) are indicated with opening and closing parentheses, slides with -, and simple bends with ^. Text annotations aligned vertically with a note in time on the staff is done by appending the text to the note info entry itself. See notate. For more details and example, see the package vignettes.

Value

a phrase.

Examples

```
phrase("c ec'g' ec'g'", "4 4 2") # no explicit string specification (not recommended)
phrase("c ec4g4 ec4g4", "4 4 2") # same as above
phrase("c b, c", "4. 8( 8)", "5 5 5") # direction implies hammer on
phrase("b2 c d", "4( 4)- 2", "5 5 5") # hammer and slide

phrase("c ec'g' ec'g'", "1 1 1", "5 432 432")
p("c ec'g' ec'g'", "1 1 1", "5 432 432") # same as above
```

`repeats` *Repeat phrases*

Description

Create a repeat section in LilyPond readable format.

Usage

```
rp(phrase, n = 1)

pct(phrase, n = 1, counter = FALSE, step = 1, reset = TRUE)

volta(phrase, n = 1, endings = NULL, silent = FALSE)
```

Arguments

`phrase`	a phrase or basic string to be repeated.
`n`	integer, number of repeats of `phrase` (one less than the total number of plays).
`counter`	logical, if `TRUE`, print the percent repeat counter above the staff, applies only to *measure* repeats of more than two repeats (`n` > 2).
`step`	integer, print the *measure* percent repeat counter above the staff only at every `step` measures when `counter = TRUE`.
`reset`	logical, percent repeat `counter` and `step` settings are only applied to the single `pct` call and are reset afterward. If `reset = FALSE`, the settings are left open to apply to any subsequent percent repeat sections in a track.
`endings`	list of phrases or basic strings, alternate endings.
`silent`	if `TRUE`, no text will be printed above the staff at the beginning of a volta section.See details.

Details

These functions wraps a phrase object or a character string in LilyPond repeat syntax. The most basic is `rp` for basic wrapping a LilyPond `unfold` repeat tag around a phrase. This repeats the phrase `n` times, but it is displayed in the engraved sheet music fully written out as a literal propagation of the phrase with no repeat notation used to reduce redundant presentation. The next is `pct`, which wraps a `percent` repeat tag around a phrase. This is displayed in sheet music as percent repeat

notation whose specific notation changes based on the length of the repeated section of music, used for beats or whole measures. `volta` wraps a phrase in a `volta` repeat tag, used for long repeats of one or more full measures or bars of music, optionally with alternate endings.

Note that basic strings should still be interpretable as a valid musical phrase by LilyPond and such strings will be coerced to the phrase class by these functions. For example, a one-measure rest, `"r1"`, does not need to be a phrase object to work with these functions, nor does any other character string explicitly written out in valid LilyPond syntax. As always, see the LilyPond documentation if you are not familiar with LilyPond syntax.

VOLTA REPEAT: When `silent = TRUE` there is no indication of the number of plays above the staff at the start of the volta section. This otherwise happens automatically when the number of repeats is greater than one and no alternate endings are included (which are already numbered). This override creates ambiguity on its own, but is important to use multiple staves are present and another staff already displays the text regarding the number or plays. This prevents printing the same text above every staff.

PERCENT REPEAT: As indicated in the parameter descriptions, the arguments `counter` and `step` only apply to full measures or bars of music. It does not apply to shorter beats that are repeated using `pct`.

Value

a phrase.

See Also

`phrase`

Examples

```
x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
e1 <- phrase("a", 1, 5) # ending 1
e2 <- phrase("b", 1, 5) # ending 2

rp(x) # simple unfolded repeat, one repeat or two plays
rp(x, 3) # three repeats or four plays

pct(x) # one repeat or two plays
pct(x, 9, TRUE, 5) # 10 plays, add counter every 5 steps
pct(x, 9, TRUE, 5, FALSE) # as above, but do not reset counter settings

volta(x) # one repeat or two plays
volta(x, 1, list(e1, e2)) # one repeat with alternate ending
volta(x, 4, list(e1, e2)) # multiple repeats but with only one alternate ending
volta(x, 4) # no alternates, more than one repeat
```

rest *Create rests*

Description

Create multiple rests efficiently with a simple wrapper around `rep` using the `times` argument.

Usage

```
rest(x, n = 1)
```

Arguments

`x`	integer, duration.
`n`	integer, number of repetitions.

Value

a character string.

Examples

```
rest(c(1, 8), c(1, 4))
```

score *Create a music score*

Description

Create a music score from a collection of tracks.

Usage

```
score(track, chords = NULL, chord_seq = NULL)
```

Arguments

`track`	a track table consisting of one or more tracks.
`chords`	an optional named list of chords and respective fingerings generated by `chord_set`, for inclusion of a top center chord diagram chart.
`chord_seq`	an optional named vector of chords and their durations, for placing chord diagrams above staves in time.

Details

Score takes track tables generated by track and fortifies them as a music score. It optionally binds tracks with a set of chord diagrams. There may be only one track in track as well as no chord information passed, but for consistency score is still required to fortify the single track as a score object that can be rendered by tab.

Value

a score table.

Examples

```
x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
x <- track(x)
score(x)
```

---

sf_phrase *Create a musical phrase from string/fret combinations*

Description

Create a musical phrase from character strings that define string numbers, fret numbers and note metadata. This function is a wrapper around phrase. It allows for specifying string/fret combinations instead of unambiguous pitch as is used by phrase. In order to remove ambiguity, it is critical to specify the instrument string tuning and key signature. It essentially uses string and fret in combination with a known tuning and key signature in order to generate notes for phrase. info is passed straight through to phrase, as is string once it is done being used to help inform notes.

Usage

```
sf_phrase(string, fret, info, key = "c", tuning = "standard",
  to_notes = FALSE, bar = FALSE)

sfp(...)

sf_note(...)

sfn(...)
```

Arguments

string	character, string numbers associated with notes.
fret	character, fret numbers associated with notes.
info	character, metadata associated with notes.
key	character, key signature or just specify "sharp" or "flat".
tuning	character, instrument tuning.

to_notes	logical, return only the mapped notes character string rather than the entire phrase object.
bar	logical, insert a bar check at the end of the phrase.
...	arguments passed to sf_phrase.

Details

See the main function phrase for more details. If you landed here first and are not familiar with phrase, be aware that sf_phrase is a tangential extra feature wrapper function in tabr and for a variety of reasons (see below) the approach it uses is discouraged in general. If this is your only option, take note of the details and limitations below.

This function is a crutch for users not working with musical notes (what to play), but rather just position on the guitar neck (where to play). This method has its conveniences, but it is inherently limiting. In order to remove ambiguity, it is necessary to specify the instrument tuning and the key signature (or at least whether non-natural notes in the output should be sharps or flats).

In the standard approach where you specify what to play, specifying exactly where to play is optional, but highly recommended (by providing string). Here string is of course required along with fret. But any time the tuning changes, this "where to play" method breaks down and must be redone. It is much more robust to provide the string and pitch rather than the string and fret. The key is always important because it is the only way to indicate if non-natural notes are sharps or flats.

This crutch method also increases redundancy and typing. In order to specify rests r, silent rests s, and tied notes ~, these must now be providing in parallel in both the string and fret arguments, whereas in the standard method using phrase, they need only be provided once to notes. A mismatch will throw an error. Despite the redundancy, this is helpful for ensuring proper match up between string and fret, which is essentially a dual entry method that aims to reduce itself inside sf_phrase to a single notes string that is passed internally to phrase.

The important thing to keep in mind is that by its nature, this method of writing out music does not lend itself well to high detail. Tabs that are informed by nothing but string and fret number remove a lot of important information, and those that attempt to compensate with additional symbols in say, an ascii tab, are difficult to read. This wrapper function providing this alternative input method does its job of allowing users to create phrase objects that are equivalent to standard phrase-generated objects, including rests and ties, but practice and comfort with working with phrase and not this wrapper is highly recommended, not just for eventual ease of use but for not preventing yourself from learning your way around the guitar neck and where all the different pitches are located.

The function sfp is a convenient shorthand wrapper for sf_phrase. sf_note and the alias sfn are wrappers around sf_phrase that force to_notes = TRUE.

Value

a phrase.

See Also

phrase

Examples

```
sf_phrase("5 4 3 2 1", "1 3 3 3 1", "8*4 1", key = "b_")
sf_phrase("654321 6s 12 1 21", "133211 355333 11 (13) (13)(13)", "4 4 8 8 4", key = "f")
sfp("6s*2 1*4", "000232*2 2*4", "4 4 8*4", tuning = "dropD", key = "d")
```

---

tab *Create tablature*

Description

Create sheet music/guitar tablature from a music score.

Usage

```
tab(score, file, key = "c", time = "4/4", tempo = "2 = 60",
  header = NULL, string_names = NULL, paper = NULL, endbar = TRUE,
  midi = TRUE, keep_ly = FALSE, path = NULL, details = TRUE)
```

Arguments

score	a score object.
file	character, output file ending in .pdf or .png. May include an absolute or relative path.
key	character, key signature, e.g., c, b_, f#m, etc.
time	character, defaults to "4/4".
tempo	character, defaults to "2 = 60".
header	a named list of arguments passed to the header of the LilyPond file. See details.
string_names	label strings at beginning of tab staff. NULL (default) for non-standard tunings only, TRUE or FALSE for force on or off completely.
paper	a named list of arguments for the LilyPond file page layout. See details.
endbar	character, the end bar.
midi	logical, output midi file in addition to tablature.
keep_ly	logical, keep LilyPond file.
path	character, optional output directory prefixed to file, may be an absolute or relative path. If NULL (default), only file is used.
details	logical, set to FALSE to disable printing of log output to console.

Details

Generate a pdf or png of a music score using the LilyPond music engraving program. Output format is inferred from file extension. This function is a wrapper around lilypond, the function that creates the LilyPond (.ly) file.

Value

nothing returned; a file is written.

See Also

lilypond, miditab

Examples

```
if(tabr_options()$lilypond != ""){
  x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
  x <- track(x)
  x <- score(x)
  outfile <- file.path(tempdir(), "out.pdf")
  tab(x, outfile, details = FALSE) # requires LilyPond installation
}
```

---

tabr *tabr: Guitar tablature and sheet music engraving.*

Description

The tabr package provides programmatic music notation and wraps around the open source music engraving program, LilyPond, for creating quality guitar tablature.

---

tabrSyntax *tabr syntax.*

Description

A data frame containing descriptions of syntax used in phrase construction in tabr.

Usage

```
tabrSyntax
```

Format

A data frame with 3 columns for syntax description, operators and examples.

tabr_options *Options*

Description

Options for tabr package.

Usage

```
tabr_options(...)
```

Arguments

...	a list of options.

Details

Currently only `lilypond`, `midi2ly` and `python` are used. On Windows systems, if the system path for lilypond.exe, midi2ly.py and python.exe are not stored in the system PATH environmental variable, they must be provided by the user after loading the package.

Value

The function prints all set options if called with no arguments. When setting options, nothing is returned.

Examples

```
tabr_options()
tabr_options(lilypond = "C:/Program Files (x86)/LilyPond/usr/bin/lilypond.exe")
```

tie *Tied notes*

Description

Tie notes efficiently.

Usage

```
tie(x)
```

Arguments

x	character, a single chord.

Details

This function is useful for bar chords.

Value

a character string.

Examples

```
tie("e,b,egbe'")
```

---

track *Create a music track*

Description

Create a music track from a collection of musical phrases.

Usage

```
track(phrase, tuning = "standard", voice = 1L, music_staff = "treble_8",
  ms_transpose = 0, ms_key = NA)
```

Arguments

phrase	a phrase object.
tuning	character, space-delimited pitches describing the instrument string tuning or a predefined tuning ID (see tunings). Defaults to standard guitar tuning. Tick or integer octave numbering accepted for custom tuning entries.
voice	integer, ID indicating the unique voice phrase belongs to within a single track (another track may share the same tab/music staff but have a different voice ID).
music_staff	add a standard sheet music staff above the tablature staff. See details.
ms_transpose	integer, positive or negative number of semitones to transpose an included music staff relative to the tablature staff. See details.
ms_key	character, specify the new key signature for a transposed music staff. See details.

Details

Musical phrases generated by phrase are fortified in a track table. All tracks are stored as track tables, one per row, even if that table consists of a single track. track creates a single-entry track table. See trackbind for merging single tracks into a multi-track table. This is simply row binding that properly preserves phrase and track classes.

The default for an additional music staff is "treble_8" for 8va treble clef, which is commonly displayed in quality guitar tablature above the tablature staff to include precise rhythm and timing information. Note that guitar is a transposing instrument. For this reason, the default ID is "treble_8", not "treble". Set music_staff = NA to suppress the additional music staff above

the tablature staff. This is appropriate for simple patterns where there are already multiple tracks and the additional space required for two staves per instrument is unnecessary and wasteful.

The arguments `ms_transpose` and `ms_key` pertain to the transposition of the music staff relative to the tab staff if `music_staff` is not `NA`. These arguments default to 0 and `NA`, respectively. The transposition and new key are simply stored in the `ms_transpose` and `ms_key` columns in the resulting track table. This information is used by `lilypond` or `tab` to transpose the music staff relative to the tab staff at the time of LilyPond file generation. Non-zero semitone transposition works without providing an explicit new key signature, but it is recommended to specify because it helps ensure the correct selection of accidentals in the output. As with the `transpose` function, you can simply specify `key = "flat"` or `key = "sharp"`. The exact key signature is not required; it is merely more clear and informative for the user.

Value

a track table.

Examples

```
x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
track(x)
```

`trackbind` *Bind track tables*

Description

Bind together track tables by row.

Usage

```
trackbind(..., tabstaff)
```

Arguments

`...`	track tables.
`tabstaff`	integer, ID vector indicating the tablature staff for each track. See details.

Details

This function appends multiple track tables into a single track table for preparation of generating a multi-track score. `tabstaff` is used to separate music staves in the sheet music/tablature output. A track's `voice` is used to separate distinct voices within a common music staff.

If not provided, the `tabstaff` ID automatically propagates `1:n` for `n` tracks passed to `...` when binding these tracks together. This expresses the default assumption of one tab staff per track. This is the typical use case where each single `track` object being bound into a multi-`track` object is a fully separated track on its own staff.

However, some tracks represent different voices that share the same staff. These should be assigned the same staff ID value, in which case you must provide the `tabstaff` argument. An error will be thrown if any two tracks have both the same `voice` and the same `tabstaff`. The pair must be unique. E.g., provide `tabstaff = c(1, 1)` when you have two tracks with `voice` equal to 1 and 2. See examples.

Note that the actual ID values assigned to each track do not matter; only the order in which tracks are bound, first to last.

Value

a track table.

Examples

```
x <- phrase("c ec'g' ec'g'", "4 4 2", "5 432 432")
x1 <- track(x)
x2 <- track(x, voice = 2)
trackbind(x1, x1)
trackbind(x1, x2, tabstaff = c(1, 1))
```

---

transpose *Transpose pitch*

Description

Transpose pitch by a number of semitones.

Usage

```
transpose(notes, n = 0, key = NA, style = c("default", "tick", "integer"))

tp(...)
```

Arguments

`notes`	character, a valid string of notes, the type passed to `phrase`.
`n`	integer, positive or negative number of semitones to transpose.
`key`	character, the new key signature after transposing `notes`. See details.
`style`	character, specify tick or integer style octave numbering in result. See details.
`...`	arguments passed to transpose.

Details

This function transposes the pitch of notes in a valid character string. The string must be of the form passed to the `info` argument to `phrase`.

Transposing is not done on a phrase object. The notes in a phrase object have already been transformed to LilyPond syntax and mixed with other potentially complex and variable information. Transposing is intended to be done on a string of notes prior to passing it to `phrase`. It will work on strings that use either integer or tick mark octave numbering formats. The transposed result will be a string with integer octave numbering.

If `key` is provided, this helps ensure proper use of sharps vs. flats. Alternatively, you can simply provide `key = "sharp"` or `key = "flat"`. The exact key signature is not required, just more clear and informative for the user. If not provided (`key = NA`), transposition lacks full information and simply defaults to sharping any resulting accidentals for positive `n` and flattening for negative `n`. `n = 0` returns the input without any modification.

The only element other pitch that occurs in a valid notes string is a rest, `"r"` or `"s"` (silent rest). Rests are ignored by transpose.

The default `style` is to use tick style if no integers occur in `notes`. The other two options force the respective styles. When integer style is returned, all `3`s are dropped since the third octave is the implicit center in LilyPond.

Value

a character string.

Examples

```
transpose("a_3 b_4 c5", 0)
tp("a_3 b_4 c5", -1)
tp("a_3 b_4 c5", 1)
tp("a#3 b4 c#5", 11)
tp("a#3 b4 c#5", 12)
tp("a#3 b4 c#5", 13)
tp("a3 b4 c5", 2, key = "f")
tp("a3 b4 c5", 2, key = "g")
tp("a b' c''", 2, key = "flat")
tp("a, b ceg", 2, key = "sharp")
```

---

tunings *Predefined instrument tunings.*

Description

A data frame containing some predefined instrument tunings commonly used for guitar, bass, mandolin, banjo, ukulele and orchestral instruments.

Usage

```
tunings
```

Format

A data frame with 2 columns for the tuning ID and corresponding pitches and 32 rows for all predefined tunings.

`tuplet` *Tuplets*

Description

Helper function for generating tuplet syntax.

Usage

```
tuplet(x, n, string = NULL, a = 3, b = 2)

triplet(x, n, string = NULL)
```

Arguments

`x`	phrase object or character string of notes.
`n`	integer, duration of each tuplet note, e.g., 8 for 8th note tuplet.
`string,`	character, optional string that specifies which guitar strings to play for each specific note.
`a`	integer, notes per tuplet.
`b`	integer, beats per tuplet.

Details

This function gives control over tuplet construction. The default arguments `a = 3` and `b = 2` generates a triplet where three triplet notes, each lasting for two thirds of a beat, take up two beats. `n` is used to describe the beat duration with the same fraction-of-measure denominator notation used for notes in `tabr` phrases, e.g., 16th note triplet, 8th note triplet, etc.

If you provide a note sequence for multiple tuplets in a row of the same type, they will be connected automatically. It is not necessary to call `tuplet` each time when the pattern is constant. If you provide a complete phrase object, it will simply be wrapped in the tuplet tag, so take care to ensure the phrase contents make sense as part of a tuplet.

Value

character.

Examples

```
tuplet("c c# d", 8)
triplet("c c# d", 8)
tuplet("c c# d c c# d", 4, a = 6, b = 4)

p1 <- phrase("c c# d", "8] 8( 8)", "5*3")
tuplet(p1, 8)
```

`%>%` *Pipe*

Description

The tabr package exports the the pipe operator, `%>%`, from magrittr just like dplyr.

Index