Package 'gm'

July 22, 2025

Type Package

Title Create Music with Ease

Version 2.0.0

Author Renfei Mao

Maintainer Renfei Mao <renfeimao@gmail.com>

Description Provides a simple and intuitive high-level language for music representation. Generates and embeds music scores and audio files in 'RStudio', 'R Markdown' documents, and R 'Jupyter Notebooks'. Internally, uses 'MusicXML' <https://github.com/w3c/musicxml> to represent music, and 'MuseScore' <https://musescore.org/> to convert 'MusicXML'.

License MIT + file LICENSE

URL https://github.com/flujoo/gm, https://flujoo.github.io/gm/

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, rstudioapi, shiny, testthat, tibble

Imports base64enc, erify, htmltools, utils

VignetteBuilder knitr

SystemRequirements MuseScore - https://musescore.org/

NeedsCompilation no

Repository CRAN

Date/Publication 2024-07-10 09:00:01 UTC

Contents

`+.Music` *Add Component to* `Music` *Object*

Description

Add a component to a `Music` object.

Usage

```
## S3 method for class 'Music'
music + object
```

Arguments

`music`	A `Music` object.
`object`	An object of class `Line`, `Meter`, `Key`, `Tempo`, `Clef`, `Instrument`, `Pedal`, `Slur`, `Hairpin`, `Notehead`, `Accidental`, `Velocity`, `Dynamic`, `Grace`, `Stem`, `Lyric`, `Tie`, `Articulation`, `Fermata`, `Breath`, `Trill`, `Turn`, `Mordent`, `Schleifer` or `Tremolo`.

Value

A list of class `Music`.

See Also

`Music()` for initialization of a `Music` object.

Examples

```
# Initialize a `Music` object
music <- Music()

# Add a `Line`
music <- music + Line("C4", 1)
music

# Add a `Meter`
music <- music + Meter(4, 4)
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Accidental` *Create* `Accidental` *Object*

Description

Create an `Accidental` object to represent an accidental symbol.

Usage

```
Accidental(name, i, j = NULL, to = NULL, bracket = NULL)
```

Arguments

`name`	A single character, which represents the name of the accidental. `"flat"` and `"sharp"` are two common examples. For a complete list of accidentals, please refer to the MusicXML specification. Unfortunately, not all accidentals are supported in MuseScore.
`i`	A single positive integer, which represents the position of the accidental in a musical line.
`j`	Optional. A single positive integer, which represents the position of the accidental in a chord.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the accidental.

bracket	Optional. A single logical, which indicates if the accidental is enclosed in brackets.

Value

A list of class Accidental.

See Also

+.Music() for adding an Accidental to a Music object.

Examples

```
# Create an `Accidental`
accidental <- Accidental("natural", 2, bracket = TRUE)
accidental

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "C4")) + accidental
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Articulation *Create* Articulation *Object*

Description

Create an Articulation object to represent an articulation mark.

Usage

```
Articulation(name, i, to = NULL)
```

Arguments

name	A single character, which represents the name or symbol of the articulation. For example, to create a staccato dot, name can be "staccato" or ".", which looks like a staccato. See the *Details* section for supported articulations.
i	A single positive integer, which represents the position of the articulation in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the articulation.

Details

Supported articulation names and symbols:

- "accent" or ">"
- "staccato" or "."
- "staccatissimo" or "'"
- "tenuto" or "-"
- "tenuto-staccato", "detached-legato" or "-."
- "marcato", "strong-accent" or "^"
- "scoop"
- "plop"
- "doit"
- "fall" or "falloff"
- "stress" or ","
- "unstress" or "u"
- "soft accent", "soft-accent" or "<>"

The names are from the MusicXML specification and MuseScore.

Value

A list of class `Articulation`.

See Also

`+.Music()` for adding an `Articulation` to a `Music` object.

Examples

```
# Create a staccato
staccato <- Articulation(".", 1)
staccato

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + staccato
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Breath *Create* Breath *Object*

Description

Create a Breath object to represent a breath mark.

Usage

```
Breath(i, to = NULL, symbol = NULL)
```

Arguments

i	A single positive integer, which represents the position of the breath mark in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the breath mark.
symbol	Optional. A single character which can be "comma", "tick", "upbow", and "salzedo". It represents the symbol used for the breath mark. The default symbol is "comma". See the MusicXML specification.

Value

A list of class Breath.

See Also

+.Music() for adding a breath mark to a Music object.

Examples

```
# Create a breath mark
breath <- Breath(1)
breath

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + breath
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Clef` *Create* `Clef` *Object*

Description

Create a `Clef` object to represent a clef.

Usage

```
Clef(sign, line = NULL, octave = NULL, to = NULL, bar = NULL, offset = NULL)
```

Arguments

`sign`	A single character, which can be `"G"`, `"F"` or `"C"`. Case insensitive.
`line`	Optional. A single integer, which depends on `sign`: • `1` or `2`, if `sign` is `"G"`; • an integer between `3` and `5`, if `sign` is `"F"`; • an integer between `1` and `5`, if `sign` is `"C"`.
`octave`	Optional. A single integer, which can be `-1` or `1`. `octave` can be specified only when • `sign` is `"G"` and `line` is `2`, or • `sign` is `"F"` and `line` is `4`.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the clef.
`bar`	Optional. A positive integer, which indicates the number of the measure where to add the clef. By default, the clef will be added at the first measure.
`offset`	Optional. A non-negative number, which indicates the clef's position in a measure. The default value is `0`.

Details

See Wikipedia for more details.

Value

A list of class `Clef`.

See Also

`+.Music()` for adding a `Clef` to a `Music` object.

Examples

```
# Create a bass clef
clef <- Clef("F")
clef

# Add the clef to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C3", "D3")) + clef
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Dynamic` *Create* `Dynamic` *Object*

Description

Create a `Dynamic` object to represent a dynamic marking.

Usage

```
Dynamic(marking, i, to = NULL, velocity = NULL, above = NULL)
```

Arguments

`marking`	A single character, which represents the dynamic symbol on the score. If `marking` is on the list in the *Details* section, and `velocity` is not specified, the corresponding velocity on the list will be used. Otherwise, `velocity` must be specified, or the `Dynamic` will have no sound effect.
`i`	A single positive integer, which represents the position of the `Dynamic` object in a musical line.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the `Dynamic`.
`velocity`	Optional. A single integer between `0` and `127`, which indicates the loudness of the `Dynamic`.
`above`	Optional. A single logical, which indicates whether the dynamic symbol should appear above or below the staff.

Details

Common used dynamic markings and their velocities in MuseScore:

- pppppp: 1
- ppppp: 5

- pppp: 10
- ppp: 16
- pp: 33
- p: 49
- mp: 64
- mf: 80
- f: 96
- ff: 112
- fff: 126
- ffff: 127
- fffff: 127
- ffffff: 127
- fp: 96
- pf: 49
- sf: 112
- sfz: 112
- sff: 126
- sffz: 126
- sfp: 112
- sfpp: 112
- rfz: 112
- rf: 112
- fz: 112
- m: 96
- r: 112
- s: 112
- z: 80
- n: 49

Value

A list of class `Dynamic`.

See Also

`+.Music()` for adding an `Dynamic` to a `Music` object.

Examples

```
# Create a `Dynamic`
f <- Dynamic("f", 1)
f

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + f
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`export` *Export* `Music` *Object*

Description

Export a `Music` object to a file format such as PNG or MP3.

Usage

```
export(x, ...)

## S3 method for class 'Music'
export(x, path, musescore = NULL, ...)
```

Arguments

`x`	A `Music` object.
`...`	Optional arguments to `export()` methods. Should be ignored by the user.
`path`	A single character, which specifies the output file path. For example, `"my/music/x.mp3"`. See the *Details* section for supported file extensions.
`musescore`	Optional. A character vector, which represents the command line options passed to MuseScore. See MuseScore command line usage for details.

Details

Supported file extensions:

1. flac
2. metajson
3. mid
4. midi
5. mlog

6. mp3
7. mpos
8. mscx
9. mscz
10. musicxml
11. mxl
12. ogg
13. pdf
14. png
15. spos
16. svg
17. wav
18. xml

Value

An invisible NULL. A file is generated in the specified path.

Examples

```
if (interactive()) {
  music <- Music() + Meter(4, 4) + Line("C4")
  export(music, tempfile(fileext = ".mp3"), "-r 200 -b 520")
}
```

Fermata *Create* Fermata *Object*

Description

Create a Fermata object to represent a fermata symbol.

Usage

```
Fermata(i, to = NULL, shape = NULL, above = NULL)
```

Arguments

i	A single positive integer, which represents the position of the fermata in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the fermata.
shape	Optional. A single character, which indicates the shape of the fermata. The default value is "normal". See the *Details* section.
above	Optional. A single logical, which indicates whether the fermata symbol should appear above or below the staff.

Details

Supported fermata shapes:

- "normal"
- "short" or "angled"
- "long" or "square"
- "very short" or "double-angled"
- "very long" or "double-square"
- "long (Henze)" or "double-dot"
- "short (Henze)" or "half-curve"
- "curlew"

The shapes are from the MusicXML specification and MuseScore.

Value

A list of class `Fermata`.

See Also

`+.Music()` for adding a `Fermata` to a `Music` object.

Examples

```
# Create a fermata
fermata <- Fermata(1)
fermata

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + fermata
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Grace` *Create* `Grace` *Object*

Description

Create a `Grace` object. The `Grace` object can be added to an existing note or chord. It will turn the note or chord to a grace note or chord.

Usage

```
Grace(i, to = NULL, slash = NULL)
```

Arguments

i	A single positive integer, which represents the position of the Grace object in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the Grace object.
slash	Optional. A single logical, which indicates if there is a slash symbol on the grace note or chord. The default value is TRUE.

Details

A Grace object can not be added to a rest, tuplet, or note or chord that has a dotted duration. There must be a note or chord after the note or chord where the Grace object is added.

Value

A list of class Grace.

See Also

+.Music() for adding a Grace object to a Music object.

Examples

```
# Create a `Grace`
grace <- Grace(1)
grace

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4"), c(0.5, 1)) + grace
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Hairpin *Create* Hairpin *Object*

Description

Create a Hairpin object to represent a crescendo or diminuendo symbol.

Usage

```
Hairpin(symbol, i, j, to = NULL, above = NULL)
```

Arguments

symbol	A single character, which can be "<" or ">". They represent crescendo and diminuendo respectively.
i, j	A single positive integer. They indicate the start and end position of the Hairpin object in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the Hairpin object.
above	Optional. A single logical, which indicates whether the Hairpin object should appear above or below the staff.

Value

A list of class Hairpin.

See Also

+.Music() for adding a Hairpin to a Music object.

Examples

```
# Create a crescendo
crescendo <- Hairpin("<", 1, 3)
crescendo

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4", "E4")) + crescendo
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Instrument *Create* Instrument *Object*

Description

Create an Instrument object to represent an instrument.

Usage

```
Instrument(instrument, to = NULL, volume = NULL, pan = NULL)
```

Arguments

`instrument`	A single integer between `1` and `128`, which indicates the program number of the instrument. See the *Details* section for all instruments.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the instrument.
`volume`	Optional. A single integer between `0` and `100`, which represents the volume of the instrument. The default value is `80`. Please note that `volume` and `pan` only work in MuseScore 3.
`pan`	Optional. A single integer between `-90` and `90`, which represents the panning of the instrument. The default value is `0`.

Details

Supported instruments:

1. Acoustic Grand Piano
2. Bright Acoustic Piano
3. Electric Grand Piano
4. Honky-Tonk Piano
5. Electric Piano 1
6. Electric Piano 2
7. Harpsichord
8. Clavinet
9. Celesta
10. Glockenspiel
11. Music Box
12. Vibraphone
13. Marimba
14. Xylophone
15. Tubular Bells
16. Dulcimer
17. Drawbar Organ
18. Percussive Organ
19. Rock Organ
20. Church Organ
21. Reed Organ
22. Accordion
23. Harmonica
24. Tango Accordion
25. Acoustic Guitar (Nylon)

26. Acoustic Guitar (Steel)
27. Electric Guitar (Jazz)
28. Electric Guitar (Clean)
29. Electric Guitar (Muted)
30. Overdriven Guitar
31. Distortion Guitar
32. Guitar Harmonics
33. Acoustic Bass
34. Electric Bass (Finger)
35. Electric Bass (Pick)
36. Fretless Bass
37. Slap Bass 1
38. Slap Bass 2
39. Synth Bass 1
40. Synth Bass 2
41. Violin
42. Viola
43. Cello
44. Contrabass
45. Tremolo Strings
46. Pizzicato Strings
47. Orchestral Harp
48. Timpani
49. String Ensemble 1
50. String Ensemble 2
51. Synth Strings 1
52. Synth Strings 2
53. Choir Aahs
54. Voice Oohs
55. Synth Voice
56. Orchestra Hit
57. Trumpet
58. Trombone
59. Tuba
60. Muted Trumpet
61. French Horn
62. Brass Section

63. Synth Brass 1
64. Synth Brass 2
65. Soprano Sax
66. Alto Sax
67. Tenor Sax
68. Baritone Sax
69. Oboe
70. English Horn
71. Bassoon
72. Clarinet
73. Piccolo
74. Flute
75. Recorder
76. Pan Flute
77. Blown Bottle
78. Shakuhachi
79. Whistle
80. Ocarina
81. Lead 1 (Square)
82. Lead 2 (Sawtooth)
83. Lead 3 (Calliope)
84. Lead 4 (Chiff)
85. Lead 5 (Charang)
86. Lead 6 (Voice)
87. Lead 7 (Fifths)
88. Lead 8 (Bass + Lead)
89. Pad 1 (New Age)
90. Pad 2 (Warm)
91. Pad 3 (Polysynth)
92. Pad 4 (Choir)
93. Pad 5 (Bowed)
94. Pad 6 (Metallic)
95. Pad 7 (Halo)
96. Pad 8 (Sweep)
97. FX 1 (Rain)
98. FX 2 (Soundtrack)
99. FX 3 (Crystal)

100. FX 4 (Atmosphere)
101. FX 5 (Brightness)
102. FX 6 (Goblins)
103. FX 7 (Echoes)
104. FX 8 (Sci-Fi)
105. Sitar
106. Banjo
107. Shamisen
108. Koto
109. Kalimba
110. Bag Pipe
111. Fiddle
112. Shanai
113. Tinkle Bell
114. Agogo
115. Steel Drums
116. Woodblock
117. Taiko Drum
118. Melodic Tom
119. Synth Drum
120. Reverse Cymbal
121. Guitar Fret Noise
122. Breath Noise
123. Seashore
124. Bird Tweet
125. Telephone Ring
126. Helicopter
127. Applause
128. Gunshot

Value

A list of class `Instrument`.

See Also

`+.Music()` for adding an instrument to a `Music` object.

Examples

```
# Create a flute
flute <- Instrument(74, pan = -90)
flute

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C5", "D5", "E5", "F5")) + flute
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Key` *Create* `Key` *Object*

Description

Create a `Key` object to represent a key signature.

Usage

```
Key(key, bar = NULL, to = NULL, scope = NULL)
```

Arguments

`key`	A single integer between -7 and 7, which indicates the number of flat or sharp symbols in the key signature.
`bar`	Optional. A positive integer, which indicates the number of the measure where to add the key signature. By default, the key signature will be added at the first measure.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the key signature. By default, the key signature will be added to the whole music rather than some specific musical line.
`scope`	Optional. A single character of `"part"` or `"staff"`, which indicates whether to add the key signature to a whole part or only some staff of the part. Only when `to` is specified, can this argument be specified. The default value is `"part"`.

Value

A list of class `Key`.

See Also

`+.Music()` for adding a key signature to a `Music` object.

Examples

```
# Create a G major
g <- Key(1, to = 1)
g

# Add it only to some part of a `Music`
music <-
  Music() +
  Meter(4, 4) +
  Line(c("C4", "D4")) +
  Line("G3") +
  g

music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Line` *Create* `Line` *Object*

Description

Create a `Line` object to represent a musical line. In gm, the musical line is the basic unit of music. It appears in different forms, such as voices, staffs, and parts in music scores.

Usage

```
Line(
  pitches = NULL,
  durations = NULL,
  tie = NULL,
  name = NULL,
  as = NULL,
  to = NULL,
  after = NULL,
  bar = NULL,
  offset = NULL
)
```

Arguments

`pitches` A list or vector which represents the pitches of a musical line. The items of `pitches` can be

- single characters like `"C4"`, which represent pitch notations,
- single integers between 12 and 127, which represent MIDI note numbers,

- single NAs, which represent rests, and
- vectors of pitch notations and MIDI note numbers, which represent chords.

If not provided, the default value is NA. If pitches and durations are not of the same length, the shorter one will be recycled. pitches and durations can not both be empty.

durations — A list or vector which represents the durations of a musical line. The items of durations can be

- single numbers, which represent note lengths, and
- single characters like "quarter", which represent duration notations.

If not provided, the default value is 1.

tie — Deprecated. Was used to add ties to notes. Please use Tie() instead.

name — Optional. A single character which represents the name of the musical line. When adding components to a musical line, it can be referred to by its name.

as — Optional. A single character which can be "part", "staff", "voice", and "segment". It specifies how the musical line appears in the music score. The default value is "part".

to — Optional. A single character or integer, which represents the name or row number of a reference musical line to which to add the current musical line. By default, the musical line will be added at the end of the score.

after — Optional. A single logical which indicates whether to add the musical line after or before the reference musical line. The default value is TRUE.

bar — Optional. A positive integer, which indicates the number of the measure where to add the musical line. By default, the musical line will be added at the first measure.

offset — Optional. A non-negative number, which indicates the position in a measure where to add the musical line. The default value is 0.

Value

A list of class Line.

See Also

+.Music() for adding a musical line to a Music object.

Examples

```
# Create a musical line
line <- Line(c("C4", "D4", "E4"))
line

# Add it to a music
music <- Music() + Meter(4, 4) + line
music

# Generate the music score
```

```
if (interactive()) {
  show(music)
}
```

`Lyric` *Create* `Lyric` *Object*

Description

Create a `Lyric` object to represent a unit of lyrics.

Usage

```
Lyric(text, i, to = NULL, verse = NULL)
```

Arguments

`text`	A single character, which usually represents a word or syllable of the lyrics. See the *Details* section for more complex usage.
`i`	A single positive integer, which represents the position of the `Lyric` in a musical line.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the `Lyric`.
`verse`	Optional. A positive integer which indicates the verse where to add the `Lyric`. The default value is 1. See the MuseScore handbook .

Details

You can use `"-"` and `"_"` in argument `text` to create the following structures:

- Syllable: for example, with `Lyric("mo-", 1)` and `Lyric("-ther", 3)`, the two syllables of *mother* are added to the first and third notes, with a hyphen placed on the second note.
- Melisma: for example, with `Lyric("love_", 1)` and `Lyric("_", 3)`, the word *love* is added to the first note, followed by an underscore line which extends over the second and third notes.
- Elision: for example, with `Lyric("my_love", 1)`, words *my* and *love* are both added to the first note, connected by an elision slur.

Use `"\\-"` and `"\\_"` if you want to add hyphens and underscores literally.

Value

A list of class `Lyric`.

See Also

`+.Music()` for adding a `Lyric` to a `Music` object.

Examples

```
# Create two syllables
syllable_1 <- Lyric("He-", 1)
syllable_2 <- Lyric("-llo", 3)
syllable_1
syllable_2

# Add them to a `Music`
music <-
  Music() +
  Meter(4, 4) +
  Line(c("C4", "D4", "E4")) +
  syllable_1 +
  syllable_2

music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Meter *Create* Meter *Object*

Description

Create a Meter object to represent a time signature.

Usage

```
Meter(
  number,
  unit,
  bar = NULL,
  actual_number = NULL,
  actual_unit = NULL,
  invisible = NULL
)
```

Arguments

number	A positive integer to represent the upper numeral of the time signature, which indicates how many beats each measure has.
unit	A single integer which can be 1, 2, 4, 8, 16, 32 or 64. It represents the lower numeral of the time signature, which indicates the duration of one single beat.

bar — Optional. A positive integer, which indicates the number of the measure where to add the time signature. By default, the time signature will be added at the first measure.

actual_number, actual_unit — Optional. They define the actual time signature rather than the one that appears on the score. Usually used to create a pickup measure. By default, they are the same as number and unit.

invisible — Optional. A single logical, which indicates whether to show the time signature on the score. Usually used to create a pickup measure. The default value is FALSE.

Value

A list of class Meter.

See Also

+.Music() for adding a Meter to a Music object.

Examples

```
# Create a 3/4 time signature
meter <- Meter(3, 4)

# Add it to a `Music`
music <- Music() + Line(c("C4", "D4", "E4")) + meter
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Mordent *Create* Mordent *Object*

Description

Create a Mordent object to represent a mordent ornament.

Usage

```
Mordent(i, to = NULL, inverted = NULL, long = NULL, ornament = NULL)
```

Arguments

`i`	A single positive integer, which represents the position of the mordent in a musical line.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the mordent.
`inverted`	Optional. A single logical, which indicates whether the mordent is inverted or not. The default value is `FALSE`. See MusicXML specification of mordent and inverted mordent.
`long`	Optional. A single logical, which indicates whether the mordent is long or not. The default value is `FALSE`.
`ornament`	Optional. A single character, which can be `"left up"`, `"left down"`, `"right up"`, or `"right down"`. It indicates the direction of the mordent's left or right part.

Value

A list of class `Mordent`.

See Also

`+.Music()` for adding a `Mordent` to a `Music` object.

Examples

```
# Create a mordent
mordent <- Mordent(1)
mordent

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + mordent
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Music` *Initialize* `Music` *Object*

Description

Initialize a `Music` object. Other components can be added to it.

Usage

```
Music()
```

Value

A list of class `Music`.

See Also

`+.Music()` for adding components to a `Music` object.

Examples

```
# Initialize a `Music`
Music()
```

`Notehead` *Create* `Notehead` *Object*

Description

Create a `Notehead` object to customize the appearance of a note's head.

Usage

```
Notehead(
  i,
  j = NULL,
  to = NULL,
  shape = NULL,
  color = NULL,
  filled = NULL,
  bracket = NULL
)
```

Arguments

`i`	A single positive integer, which represents the position of the note in a musical line.
`j`	Optional. A single positive integer, which represents the position of the note in a chord.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to apply the `Notehead`.
`shape`	Optional. A single character which represents the shape of the note's head. See the MusicXML specification for all shapes. Unfortunately, not all shapes are supported in MuseScore.
`color`	Optional. A single character which represents the color of the note's head. It must be in the hexadecimal RGB or ARGB format.
`filled`	Optional. A single logical, which indicates whether the note's head is filled or hollow.
`bracket`	Optional. A single logical, which indicates whether the note's head is enclosed in brackets.

Value

A list of class Notehead.

See Also

+.Music() for adding a Notehead to a Music object.

Examples

```
# Create a `Notehead`
notehead <- Notehead(1, shape = "diamond", color = "#800080")
notehead

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + notehead
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Pedal *Create* Pedal *Object*

Description

Create a Pedal object to represent piano sustain pedal marks.

Usage

```
Pedal(i, j, to = NULL)
```

Arguments

i, j	A single positive integer. They indicate the start and end position of the Pedal object in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the Pedal object.

Value

A list of class Pedal.

See Also

+.Music() for adding a Pedal to a Music object.

Examples

```
# Create a `Pedal`
pedal <- Pedal(1, 3)
pedal

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4", "E4")) + pedal
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Schleifer *Create* Schleifer *Object*

Description

Create a Schleifer object to represent a slide ornament. See the MusicXML specification.

Usage

```
Schleifer(i, to = NULL)
```

Arguments

i	A single positive integer, which represents the position of the Schleifer object in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the Schleifer object.

Value

A list of class Schleifer.

See Also

+.Music() for adding a Schleifer to a Music object.

Examples

```
# Create a `Schleifer`
schleifer <- Schleifer(1)
schleifer

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + schleifer
```

```
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

---

show *Show* Music *Object*

Description

Display a Music object as a music score or audio file.

Usage

```
show(x, to, musescore)

## S3 method for class 'Music'
show(x, to = NULL, musescore = NULL)
```

Arguments

x	A Music object.
to	Optional. A character vector, which can be "score", "audio", or both. It specifies the output format. By default, both are displayed. You can change the default behavior by setting the gm.show_to option with options().
musescore	Optional. A character vector, which represents the command line options passed to MuseScore. See MuseScore command line usage for details.

Details

This function works in

- RStudio
- R Markdown files
- Jupyter Notebooks
- Shiny applications
- R.app GUI

Value

An invisible NULL. A music score or audio file will be displayed.

Examples

```
if (interactive()) {
  music <- Music() + Meter(4, 4) + Line("C4")
  show(music, musescore = "-r 800 -T 5")
}
```

Slur *Create* Slur *Object*

Description

Create a Slur object to represent a slur.

Usage

```
Slur(i, j, to = NULL, to_j = NULL, above = NULL)
```

Arguments

i, j	A single positive integer. They indicate the start and end positions of the slur.
to, to_j	Optional. A single character or a single positive integer, which indicates the musical line where to add the slur. Specify to_j if the start and end positions are in different musical lines.
above	Optional. A single logical, which indicates whether the slur should appear above or below the staff. By default, the position is decided by MuseScore.

Value

A list of class Slur.

See Also

+.Music() for adding a slur to a Music object.

Examples

```
# Create a slur
slur <- Slur(1, 3)
slur

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4", "E4")) + slur
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Stem *Create* Stem *Object*

Description

Create a Stem object to modify the stem of some note.

Usage

```
Stem(direction, i, to = NULL)
```

Arguments

direction	A single character, which can be "down", "up", "double", and "none". See the MusicXML specification.
i	A single positive integer, which represents the position of the stem in a musical line.
to	Optional. A single character or a single positive integer, which indicates the musical line where to modify the stem.

Value

A list of class Stem.

See Also

+.Music() for adding a Stem to a Music object.

Examples

```
# Create a `Stem`
stem <- Stem("none", 1)
stem

# Add a `Stem` to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + stem
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Tempo *Create* Tempo *Object*

Description

Create a Tempo object to represent a tempo marking.

Usage

```
Tempo(tempo, unit = NULL, bar = NULL, offset = NULL, marking = NULL)
```

Arguments

tempo	A positive number, which indicates the number of quarter notes per minute.
unit	Deprecated. Was used to specify the beat unit. Please use marking instead.
bar	Optional. A positive integer, which indicates the number of the measure where to add the tempo. By default, it will be added at the first measure.
offset	Optional. A non-negative number, which indicates the tempo's position in a measure. The default value is 0.
marking	Optional. A single character, which represents the marking that appears on the score. See the *Details* section.

Details

The parameter tempo is used to specify the actual playback speed, while marking to represent the marking that appears on the score.

Some examples:

- Tempo(50): the playback speed is 50 quarter notes per minute. A marking of "quarter = 50" will be added to the score.
- Tempo(50, marking = "Adagio"): the playback speed is 50 quarter notes per minute, while the marking on the score is "Adagio".
- Tempo(50, marking = "Adagio half. = 20"): the playback speed is 50 quarter notes per minute, while the marking on the score is "Adagio half. = 20".
- Tempo(50, marking = "Adagio (quarter = 45-80)"): you can add a speed range and parentheses to the marking.
- Tempo(50, marking = "quarter. = quarter"): you can also indicate metric modulations with marking.

Value

A list of class Tempo.

See Also

+.Music() for adding a tempo to a Music object.

Examples

```
# Create a tempo
tempo <- Tempo(50, marking = "Adagio (half = 25)")
tempo

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4", "E4", "F4")) + tempo
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Tie` *Create* `Tie` *Object*

Description

Create a `Tie` to tie some notes together.

Usage

```
Tie(i, j = NULL, to = NULL, above = NULL)
```

Arguments

`i`	A single positive integer, which represents the start position of the tie in a musical line.
`j`	Optional. A single positive integer, which represents the start position of the tie in a chord. If not provided, all notes in the chords that have equivalent pitches are tied.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the tie.
`above`	Optional. A single logical, which indicates if the tie is placed above the notes. By default, the position is decided by MuseScore.

Value

A list of class `Tie`.

See Also

`+.Music()` for adding a tie to a `Music` object.

Examples

```
# Create a tie
tie <- Tie(1)
tie

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "C4")) + tie
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Tremolo` *Create* `Tremolo` *Object*

Description

Create a `Tremolo` object to represent a tremolo.

Usage

```
Tremolo(number, i, to = NULL, between = NULL)
```

Arguments

`number`	A single integer which can be `1`, `2`, `3`, and `4`. It indicates the speed of the tremolo.
`i`	A single positive integer, which represents the position of the tremolo in a musical line.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the tremolo.
`between`	Optional. A single logical which indicates if the tremolo is between notes.

Value

A list of class `Tremolo`.

See Also

`+.Music()` for adding a tremolo to a `Music` object.

Examples

```
# Create a tremolo
tremolo <- Tremolo(3, 1, between = TRUE)
tremolo

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4", "E4", "F4")) + tremolo
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Trill *Create* Trill *Object*

Description

Create a Trill object to represent a trill ornament.

Usage

```
Trill(i, j = NULL, to = NULL)
```

Arguments

i	A single positive integer, which represents the position of the trill in a musical line.
j	Optional. A single positive integer, which indicates the end position of the trill line in a musical line. If not provided, the trill will appear as a *tr* symbol above only the trilled note. Otherwise, it will appear as a *tr*~~~ symbol above the notes between the start and end positions.
to	Optional. A single character or a single positive integer, which indicates the musical line where to add the trill.

Value

A list of class Trill.

See Also

+.Music() for adding a trill to a Music object.

Examples

```
# Create a trill
trill <- Trill(1, 3)
trill

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4", "E4", "F4")) + trill
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Turn` *Create* `Turn` *Object*

Description

Create a `Turn` object to represent a turn ornament.

Usage

```
Turn(i, to = NULL, inverted = NULL)
```

Arguments

`i`	A single positive integer, which represents the position of the turn in a musical line.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to add the turn.
`inverted`	Optional. A single logical, which indicates if it is an inverted turn. The default value is `FALSE`. See MusicXML specification of turn and inverted turn.

Value

A list of class `Turn`.

See Also

`+.Music()` for adding a turn to a `Music` object.

Examples

```
# Create a turn
turn <- Turn(1, inverted = TRUE)
turn

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + turn
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

`Velocity` *Create* `Velocity` *Object*

Description

Create a `Velocity` object to set some notes' velocities.

Usage

```
Velocity(velocity, to = NULL, i = NULL, j = NULL)
```

Arguments

`velocity`	A single integer between `0` and `127`, which indicates the velocity to apply.
`to`	Optional. A single character or a single positive integer, which indicates the musical line where to apply the velocity. If not provided, the velocity will be applied to all notes.
`i`	Optional. A single positive integer, which represents the position of the velocity in a musical line.
`j`	Optional. A single positive integer, which represents the position of the velocity in a chord.

Value

A list of class `Velocity`.

See Also

- `+.Music()` for adding a `Velocity` to a `Music` object
- `Dynamic()` for adding dynamic markings

Examples

```
# Create a `Velocity`
velocity <- Velocity(10)
velocity

# Add it to a `Music`
music <- Music() + Meter(4, 4) + Line(c("C4", "D4")) + velocity
music

# Generate the music score
if (interactive()) {
  show(music)
}
```

Index