LilyPond

The music typesetter

User manual

The LilyPond development team

Copyright © 1999–2007 by the authors

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections. A copy of the license is included in the section entitled "GNU Free Documentation License".

(For LilyPond version 2.11.39)

Table of Contents

1 Musical notation 1

1.1 Pitches 1
1.1.1 Writing pitches 1
1.1.1.1 Absolute octave entry 1
1.1.1.2 Relative octave entry 2
1.1.1.3 Accidentals 4
1.1.1.4 Note names in other languages 6
1.1.2 Changing multiple pitches 7
1.1.2.1 Octave checks 7
1.1.2.2 Transpose 8
1.1.3 Displaying pitches 11
1.1.3.1 Clef 11
1.1.3.2 Key signature 14
1.1.3.3 Ottava brackets 16
1.1.3.4 Instrument transpositions 17
1.1.3.5 Automatic accidentals 19
1.1.3.6 Ambitus 24
1.1.4 Note heads 26
1.1.4.1 Special note heads 26
1.1.4.2 Easy notation note heads 26
1.1.4.3 Shape note heads 27
1.1.4.4 Improvisation 28
1.2 Rhythms 29
1.2.1 Writing rhythms 29
1.2.1.1 Durations 30
1.2.1.2 Tuplets 31
1.2.1.3 Scaling durations 34
1.2.1.4 Ties 35
1.2.2 Writing rests 37
1.2.2.1 Rests 37
1.2.2.2 Skips 38
1.2.2.3 Full measure rests 39
1.2.3 Displaying rhythms 42
1.2.3.1 Time signature 42
1.2.3.2 Upbeats 44
1.2.3.3 Unmetered music 45
1.2.3.4 Polymetric notation 45
1.2.3.5 Automatic note splitting 48
1.2.4 Beams 49
1.2.4.1 Automatic beams 49
1.2.4.2 Setting automatic beam behavior 51
1.2.4.3 Manual beams 54
1.2.4.4 Feathered beams 54
1.2.5 Bars 55
1.2.5.1 Bar lines 55
1.2.5.2 Bar numbers 58
1.2.5.3 Bar and bar number checks 61
1.2.5.4 Rehearsal marks 62

1.2.6 Special rhythmic concerns ... 63
1.2.6.1 Grace notes ... 63
1.2.6.2 Aligning to cadenzas ... 66
1.2.6.3 Time administration ... 67
1.3 Expressive marks ... 68
1.3.1 Attached to notes ... 68
1.3.1.1 Articulations and ornamentations ... 68
1.3.1.2 Dynamics ... 70
1.3.2 Curves ... 73
1.3.2.1 Slurs ... 73
1.3.2.2 Phrasing slurs ... 74
1.3.2.3 Breath marks ... 74
1.3.2.4 Falls and doits ... 75
1.3.3 Lines ... 75
1.3.3.1 Glissando ... 75
1.3.3.2 Arpeggio ... 76
1.3.3.3 Trills ... 78
1.4 Repeats ... 79
1.4.1 Writing repeats ... 79
1.4.1.1 Repeat syntax ... 80
1.4.1.2 Normal repeats ... 80
1.4.1.3 Manual repeat commands ... 82
1.4.2 Other repeats ... 83
1.4.2.1 Tremolo repeats ... 83
1.4.2.2 Measure repeats ... 84
1.5 Simultaneous notes ... 85
1.5.1 Single voice ... 86
1.5.1.1 Chorded notes ... 86
1.5.1.2 Clusters ... 86
1.5.2 Multiple voices ... 86
1.5.2.1 Collision resolution ... 87
1.5.2.2 Automatic part combining ... 89
1.5.2.3 Writing music in parallel ... 91
1.6 Staff notation ... 92
1.6.1 Displaying staves ... 93
1.6.1.1 System start delimiters ... 93
1.6.1.2 Staff symbol ... 95
1.6.1.3 Hiding staves ... 97
1.6.2 Writing parts ... 98
1.6.2.1 Metronome marks ... 98
1.6.2.2 Instrument names ... 99
1.6.2.3 Quoting other voices ... 101
1.6.2.4 Formatting cue notes ... 102
1.7 Editorial annotations ... 104
1.7.1 Inside the staff ... 105
1.7.1.1 Selecting notation font size ... 105
1.7.1.2 Fingering instructions ... 106
1.7.1.3 Hidden notes ... 107
1.7.1.4 Coloring objects ... 107
1.7.1.5 Parentheses ... 109
1.7.1.6 Stems ... 109
1.7.2 Outside the staff ... 110
1.7.2.1 Balloon help ... 110
1.7.2.2 Grid lines ... 110

1.7.2.3 Blank music sheet ... 111
1.7.2.4 Analysis brackets ... 112
1.8 Text ... 113
1.8.1 Writing text ... 113
1.8.1.1 Overview of text entry ... 113
1.8.1.2 Text scripts ... 114
1.8.1.3 Text spanners ... 115
1.8.1.4 Text marks ... 116
1.8.2 Text markup ... 119
1.8.2.1 Text markup introduction ... 119
1.8.2.2 Nested scores ... 122
1.8.2.3 Page wrapping text ... 122
1.8.2.4 Font selection ... 123
1.8.3 Special text concerns ... 124
1.8.3.1 New dynamic marks ... 124
1.8.3.2 Text and line spanners ... 125

2 Specialist notation ... 129
2.1 Vocal music ... 129
2.1.1 Simple lyrics ... 129
2.1.1.1 Setting simple songs ... 129
2.1.1.2 Entering lyrics ... 130
2.1.2 Aligning lyrics to a melody ... 132
2.1.2.1 Automatic syllable durations ... 132
2.1.2.2 Another way of entering lyrics ... 133
2.1.2.3 Assigning more than one syllable to a single note ... 133
2.1.2.4 More than one note on a single syllable ... 134
2.1.2.5 Extenders and hyphens ... 135
2.1.3 Vocals and variables ... 135
2.1.3.1 Working with lyrics and variables ... 135
2.1.4 Flexibility in placement ... 136
2.1.4.1 Lyrics to multiple notes of a melisma ... 136
2.1.4.2 Divisi lyrics ... 137
2.1.4.3 Switching the melody associated with a lyrics line ... 138
2.1.4.4 Lyrics independent of notes ... 139
2.1.5 Spacing vocals ... 139
2.1.5.1 Spacing lyrics ... 139
2.1.6 More about stanzas ... 140
2.1.6.1 Adding stanza numbers ... 141
2.1.6.2 Adding dynamics marks ... 141
2.1.6.3 Adding singer names ... 141
2.1.6.4 Printing stanzas at the end ... 142
2.1.6.5 Printing stanzas at the end in multiple columns ... 143
2.2 Chords ... 144
2.2.1 Chords sections ... 144
2.2.1.1 A lead sheet ... 145
2.2.1.2 Introducing chord names ... 146
2.2.1.3 Chords mode ... 146
2.2.1.4 Printing chord names ... 148
2.3 Piano music ... 151
2.3.1 Piano sections ... 151
2.3.1.1 Automatic staff changes ... 152
2.3.1.2 Manual staff switches ... 152
2.3.1.3 Pedals ... 153

2.3.1.4 Staff switch lines ... 154
2.3.1.5 Cross staff stems ... 155
2.4 Percussion ... 155
2.4.1 Percussion sections ... 155
2.4.1.1 Showing melody rhythms ... 155
2.4.1.2 Entering percussion ... 156
2.4.1.3 Percussion staves ... 156
2.4.1.4 Ghost notes ... 159
2.5 Guitar ... 160
2.5.1 Guitar sections ... 160
2.5.1.1 String number indications ... 160
2.5.1.2 Tablatures basic ... 160
2.5.1.3 Non-guitar tablatures ... 161
2.5.1.4 Banjo tablatures ... 162
2.5.1.5 Fret diagrams ... 163
2.5.1.6 Right hand fingerings ... 163
2.5.1.7 Other guitar issues ... 164
2.6 Orchestral strings ... 165
2.6.1 Orchestral strings sections ... 165
2.6.1.1 Artificial harmonics (strings) ... 165
2.7 Bagpipes ... 165
2.7.1 Bagpipe sections ... 165
2.7.1.1 Bagpipe definitions ... 165
2.7.1.2 Bagpipe example ... 166
2.8 Ancient notation ... 167
2.8.1 Alternative note signs for ancient music ... 168
2.8.1.1 Ancient note heads ... 168
2.8.1.2 Ancient accidentals ... 168
2.8.1.3 Ancient rests ... 169
2.8.1.4 Ancient clefs ... 169
2.8.1.5 Ancient flags ... 171
2.8.1.6 Ancient time signatures ... 172
2.8.2 Additional note signs for ancient music ... 173
2.8.2.1 Ancient articulations ... 173
2.8.2.2 Custodes ... 174
2.8.2.3 Divisiones ... 175
2.8.2.4 Ligatures ... 176
2.8.2.5 White mensural ligatures ... 176
2.8.2.6 Gregorian square neumes ligatures ... 177
2.8.3 Pre-defined contexts ... 183
2.8.3.1 Gregorian Chant contexts ... 183
2.8.3.2 Mensural contexts ... 184
2.8.4 Musica ficta accidentals ... 185
2.8.5 Figured bass ... 185

3 Input syntax ... 189

3.1 Input files ... 189
3.1.1 File structure ... 189
3.1.2 A single music expression ... 190
3.1.3 Multiple scores in a book ... 191
3.1.4 Extracting fragments of notation ... 192
3.1.5 Including LilyPond files ... 192
3.1.6 Text encoding ... 193
3.1.7 Different editions from one source ... 193
3.2 Common syntax issues TODO name? ... 194
3.2.1 Controlling direction ... 194
3.2.2 Distances and measurements MAYBE MOVE ... 195
3.3 Other stuffs TODO move? ... 195
3.3.1 Displaying LilyPond notation ... 195
3.3.2 Skipping corrected music ... 195
3.3.3 context list FIXME ... 196
3.3.4 another thing FIXME ... 197
3.3.5 Input modes FIXME ... 197

4 Non-musical notation ... 198

4.1 Titles and headers ... 198
4.1.1 Creating titles ... 198
4.1.2 Custom titles ... 201
4.1.3 Reference to page numbers ... 202
4.1.4 Table of contents ... 203
4.2 MIDI output ... 204
4.2.1 Creating MIDI files ... 205
4.2.2 MIDI block ... 206
4.2.3 MIDI instrument names ... 206
4.2.4 What goes into the MIDI? FIXME ... 206
4.2.4.1 Repeats and MIDI ... 206
4.3 other midi ... 207

5 Spacing issues ... 208

5.1 Paper and pages ... 208
5.1.1 Paper size ... 208
5.1.2 Page formatting ... 208
5.2 Music layout ... 213
5.2.1 Setting the staff size ... 213
5.2.2 Score layout ... 214
5.3 Displaying spacing ... 214
5.4 Breaks ... 215
5.4.1 Line breaking ... 215
5.4.2 Page breaking ... 217
5.4.3 Optimal page breaking ... 217
5.4.4 Optimal page turning ... 217
5.4.5 Minimal page breaking ... 218
5.4.6 Explicit breaks ... 218
5.4.7 Using an extra voice for breaks ... 220
5.5 Vertical spacing ... 222
5.5.1 Vertical spacing inside a system ... 222
5.5.2 Vertical spacing between systems ... 224
5.5.3 Explicit staff and system positioning ... 226

5.5.4 Two-pass vertical spacing ... 232
5.5.5 Vertical collision avoidance ... 233
5.6 Horizontal Spacing ... 234
5.6.1 Horizontal spacing overview ... 234
5.6.2 New spacing area ... 236
5.6.3 Changing horizontal spacing ... 236
5.6.4 Line length ... 238
5.6.5 Proportional notation ... 239
5.7 Page layout MOVED FROM LM ... 245
5.7.1 Introduction to layout ... 245
5.7.2 Global sizes ... 245
5.7.3 Line breaks ... 246
5.7.4 Page breaks ... 246
5.7.5 Fitting music onto fewer pages ... 246

6 Changing defaults ... 249
6.1 Interpretation contexts ... 249
6.1.1 Contexts explained ... 249
6.1.2 Creating contexts ... 250
6.1.3 Changing context properties on the fly ... 251
6.1.4 Modifying context plug-ins ... 253
6.1.5 Layout tunings within contexts ... 254
6.1.6 Changing context default settings ... 256
6.1.7 Defining new contexts ... 257
6.1.8 Aligning contexts ... 259
6.1.9 Vertical grouping of grobs ... 259
6.2 The `\override` command ... 259
6.2.1 Constructing a tweak ... 259
6.2.2 Navigating the program reference ... 260
6.2.3 Layout interfaces ... 260
6.2.4 Determining the grob property ... 261
6.2.5 Objects connected to the input ... 262
6.2.6 Using Scheme code instead of `\tweak` ... 263
6.2.7 `\set` vs. `\override` ... 264
6.2.8 Difficult tweaks ... 264

7 Interfaces for programmers ... 266
7.1 Music functions ... 266
7.1.1 Overview of music functions ... 266
7.1.2 Simple substitution functions ... 266
7.1.3 Paired substitution functions ... 268
7.1.4 Mathematics in functions ... 268
7.1.5 Void functions ... 269
7.1.6 Functions without arguments ... 269
7.1.7 Overview of available music functions ... 269
7.2 Programmer interfaces ... 272
7.2.1 Input variables and Scheme ... 272
7.2.2 Internal music representation ... 273
7.3 Building complicated functions ... 274
7.3.1 Displaying music expressions ... 274
7.3.2 Music properties ... 275
7.3.3 Doubling a note with slurs (example) ... 276
7.3.4 Adding articulation to notes (example) ... 277

7.4 Markup programmer interface ... 279
7.4.1 Markup construction in Scheme ... 279
7.4.2 How markups work internally ... 280
7.4.3 New markup command definition ... 280
7.4.4 New markup list command definition ... 282
7.5 Contexts for programmers ... 282
7.5.1 Context evaluation ... 282
7.5.2 Running a function on all layout objects ... 283
7.6 Scheme procedures as properties ... 283

Appendix A Literature list ... 285

Appendix B Notation manual tables ... 286
B.1 Chord name chart ... 286
B.2 MIDI instruments ... 287
B.3 List of colors ... 288
B.4 The Feta font ... 290
B.5 Note head styles ... 305
B.6 Overview of text markup commands ... 306
B.7 Overview of text markup list commands ... 314
B.8 List of articulations ... 314
B.9 All context properties ... 316
B.10 Layout properties ... 325
B.11 Identifiers ... 338
B.12 Scheme functions ... 341

Appendix C Cheat sheet ... 359

Appendix D GNU Free Documentation License ... 363
D.0.1 ADDENDUM: How to use this License for your documents ... 368

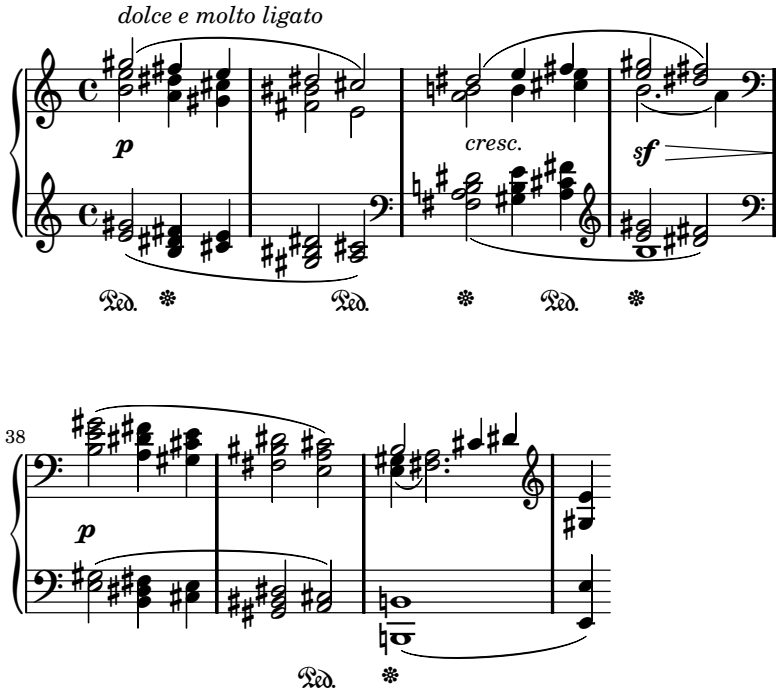
Appendix E LilyPond command index ... 369

Appendix F LilyPond index ... 373

1 Musical notation

This chapter explains how to create musical notation.

1.1 Pitches



This section discusses how to specify the pitch of notes. There are three steps to this process: input, modification, and output.

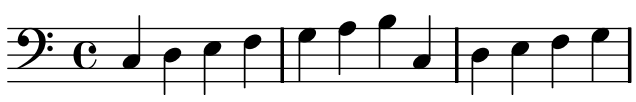
1.1.1 Writing pitches

This section discusses how to input pitches. There are two different ways to place notes in octaves: absolute and relative mode. In most cases, relative mode will be more convenient.

1.1.1.1 Absolute octave entry

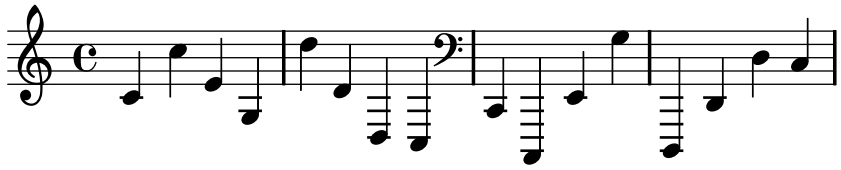
A pitch name is specified using lowercase letters `a` through `g`. The note names `c` to `b` are engraved in the octave below middle C.

```
\clef bass
c d e f
g a b c
d e f g
```



Other octaves may be specified with a single quote (`'`) or comma (`,`) character. Each `'` raises the pitch by one octave; each `,` lowers the pitch by an octave.

```
\clef treble
c' c'' e' g
d'' d' d c
\clef bass
c, c,, e, g
d,, d, d c
```



See also

Music Glossary: ⟨undefined⟩ [Pitch names], page ⟨undefined⟩.

Snippets: Pitches .

1.1.1.2 Relative octave entry

When octaves are specified in absolute mode it is easy to accidentally put a pitch in the wrong octave. Relative octave mode reduces these errors since most of the time it is not necessary to indicate any octaves at all. Furthermore, in absolute mode a single mistake may be difficult to spot, while in relative mode a single error puts the rest of the piece off by one octave.

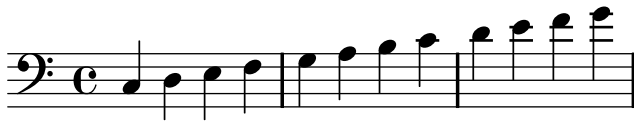
`\relative` *`startpitch musicexpr`*

In relative mode, each note is assumed to be as close to the previous note as possible. This means that the octave of pitches in *musicexpr* is calculated as follows:

- If no octave changing mark is used on a pitch, its octave is calculated so that the interval with the previous note is less than a fifth. This interval is determined without considering accidentals.
- An octave changing mark `'` or `,` can be added to respectively raise or lower a pitch by an extra octave, relative to the pitch calculated without an octave mark.
- Multiple octave changing marks can be used. For example, `''` and `,,` will alter the pitch by two octaves.
- The pitch of the first note is relative to *`startpitch`*. *startpitch* is specified in absolute octave mode, and it is recommended that it be a octave of `c`.

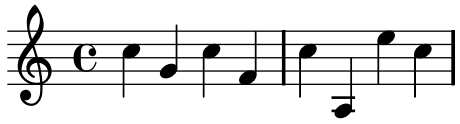
Here is the relative mode shown in action:

```
\relative c {
  \clef bass
  c d e f
  g a b c
  d e f g
}
```



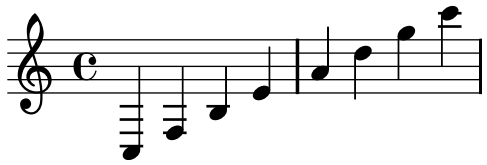
Octave changing marks are used for intervals greater than a fourth:

```
\relative c'' {
  c g c f,
  c' a, e'' c
}
```



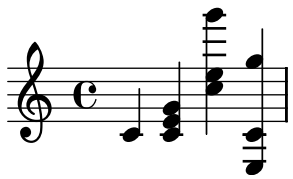
A note sequence without a single octave mark can nevertheless span large intervals:

```
\relative c {
  c f b e
  a d g c
}
```



If the preceding item is a chord, the first note of the chord is used as the reference point for the octave placement of a following note or chord. Inside chords, the next note is always relative to the preceding one.

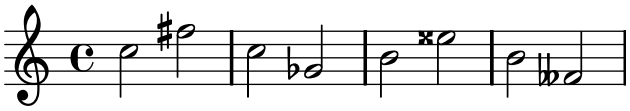
```
\relative c' {
  c
  <c e g>
  % the C is now an octave higher
  <c' e g'>
  % the C returns to the original pitch
  <c, e, g''>
}
```



As explained above, the octave of pitches is calculated only with the note names, regardless of any alterations. Therefore, an E-double-sharp following a B will be placed higher, while an F-double-flat will be placed lower. In other words, a double-augmented fourth is considered a smaller interval than a double-diminished fifth, regardless of the number of semitones that each interval contains.

```
\relative c'' {
  c2 fis
  c2 ges
  b2 eisis
  b2 feses
```

```
}
```



See also

Music Glossary: ⟨undefined⟩ [fifth], page ⟨undefined⟩, ⟨undefined⟩ [interval], page ⟨undefined⟩, ⟨undefined⟩ [Pitch names], page ⟨undefined⟩.

Notation Reference: Section 1.1.2.1 [Octave checks], page 7.

Snippets: Pitches .

Known issues and warnings

The relative conversion will not affect `\transpose`, `\chordmode` or `\relative` sections in its argument. To use relative mode within transposed music, an additional `\relative` must be placed inside `\transpose`.

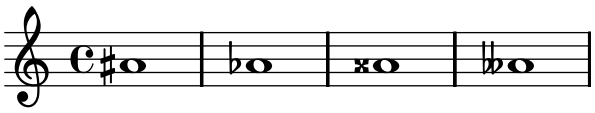
If no *startpitch* is specified for `\relative`, then `c'` is assumed. However, this is a deprecated option and may disappear in future versions, so its use is discouraged.

1.1.1.3 Accidentals

> **Note:** New users are sometimes confused about accidentals and key signatures. In LilyPond, note names are the raw input; key signatures and clefs determine how this raw input is displayed. An unaltered note like `c` means 'C natural', regardless of the key signature or clef. For more information, see learning manual, ⟨undefined⟩ [Accidentals and key signatures], page ⟨undefined⟩ .

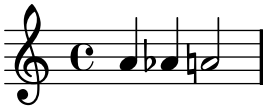
A *sharp* pitch is made by adding `is` to the note name, and a *flat* pitch by adding `es`. As you might expect, a *double sharp* or *double flat* is made by adding `isis` or `eses`. This syntax is derived from Dutch note naming conventions. To use other names for accidentals, see Section 1.1.1.4 [Note names in other languages], page 6.

```
ais1 aes aisis aeses
```



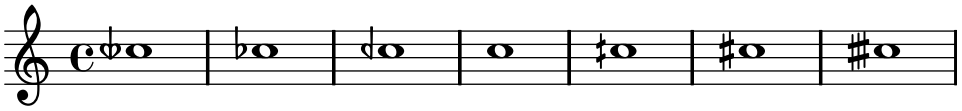
A natural will cancel the effect of an accidental or key signature. However, naturals are not encoded into the note name syntax with a suffix; a natural pitch is shown as a simple note name:

```
a4 aes a2
```



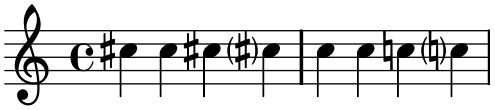
Quarter tones may be added; the following is a series of Cs with increasing pitches

```
ceseh1 ces ceh c cih cis cisih
```



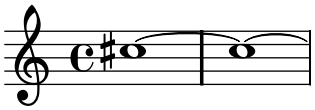
Normally accidentals are printed automatically, but you may also print them manually. A reminder accidental can be forced by adding an exclamation mark `!` after the pitch. A cautionary accidental (i.e., an accidental within parentheses) can be obtained by adding the question mark `?` after the pitch. These extra accidentals can also be used to produce natural signs.

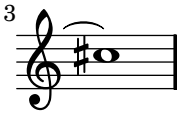
```
cis cis cis! cis? c c c! c?
```



Accidentals on tied notes are only printed at the beginning of a new system:

```
cis1 ~ cis ~
\break
cis
```

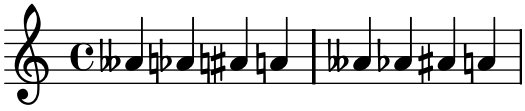




Selected Snippets

In accordance with standard typesetting rules, a natural sign is printed before a sharp or flat if a previous accidental on the same note needs to be canceled. To change this behavior, set the extraNatural property to "false" in the Staff context.

```
\relative {
  aeses'4 aes ais a
  \set Staff.extraNatural = ##f
  aeses4 aes ais a
}
```



See also

Music Glossary: ⟨undefined⟩ [sharp], page ⟨undefined⟩, ⟨undefined⟩ [flat], page ⟨undefined⟩, ⟨undefined⟩ [double sharp], page ⟨undefined⟩, ⟨undefined⟩ [double flat], page ⟨undefined⟩, ⟨undefined⟩ [Pitch names], page ⟨undefined⟩, ⟨undefined⟩ [quarter-tone], page ⟨undefined⟩.

Learning Manual: learning manual, ⟨undefined⟩ [Accidentals and key signatures], page ⟨undefined⟩ .

Notation Reference: Section 1.1.3.5 [Automatic accidentals], page 19, Section 2.8.4 [Musica ficta accidentals], page 185, Section 1.1.1.4 [Note names in other languages], page 6.

Snippets: Pitches .

Known issues and warnings

There are no generally accepted standards for denoting quarter-tone accidentals, so LilyPond's symbol does not conform to any standard.

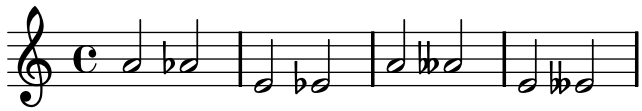
1.1.1.4 Note names in other languages

There are predefined sets of note names for various other languages. To use them, include the language-specific init file. For example, to use English notes names, add `\include "english.ly"` to the top of the input file. The available language files and the note names they define are:

Language	Note names	sharp	flat	double sharp	double flat
nederlands.ly	c d e f g a bes b	-is	-es	-isis	-eses
english.ly	c d e f g a bf b	-s/-sharp	-f/-flat	-ss/-x/-sharpsharp	-ff/-flatflat
deutsch.ly	c d e f g a b h	-is	-es	-isis	-eses
norsk.ly	c d e f g a b h	-iss/-is	-ess/-es	-ississ/-isis	-essess/-eses
svenska.ly	c d e f g a b h	-iss	-ess	-ississ	-essess
suomi.ly	c d e f g a b h	-is	-es	-isis	-eses
italiano.ly	do re mi fa sol la sib si	-d	-b	-dd	-bb
catalan.ly	do re mi fa sol la sib si	-d/-s	-b	-dd/-ss	-bb
espanol.ly	do re mi fa sol la sib si	-s	-b	-ss	-bb
portugues.ly	do re mi fa sol la sib si	-s	-b	-ss	-bb
vlaams.ly	do re mi fa sol la sib si	-k	-b	-kk	-bb

In Dutch, `aes` is contracted to `as`, but both forms are accepted in LilyPond. Similarly, both `es` and `ees` are accepted. This also applies to `aeses` / `ases` and `eeses` / `eses`. Sometimes only these contracted names are defined in the corresponding language files.

```
a2 as e es a ases e eses
```



Some music uses microtones whose alterations are fractions of a 'normal' sharp or flat. The note names for quarter-tones defined in the various language files are listed in the following table. Here the prefixes *semi-* and *sesqui-* mean 'half' and 'one and a half', respectively. For the other languages, no special names have been defined yet.

Language	Note names	semi-sharp	semi-flat	sesqui-sharp	sesqui-flat
nederlands.ly	c d e f g a bes b	-ih	-eh	-isih	-eseh
english.ly	c d e f g a bf b	-qs	-qf	-tqs	-tqf
deutsch.ly	c d e f g a b h	-ih	-eh	-isih	-eseh
italiano.ly	do re mi fa sol la sib si	-sd	-sb	-dsd	-bsb
portugues.ly	do re mi fa sol la sib si	-sqt	-bqt	-stqt	-btqt

See also

Music Glossary: ⟨undefined⟩ [Pitch names], page ⟨undefined⟩.

Snippets: Pitches .

1.1.2 Changing multiple pitches

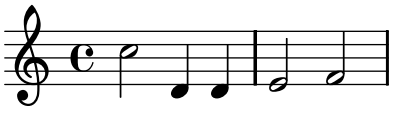
This section discusses how to modify pitches.

1.1.2.1 Octave checks

In relative mode, it is easy to forget an octave changing mark. Octave checks make such errors easier to find by displaying a warning and correcting the octave if a note is found in an unexpected octave.

To check the octave of a specific note, specify the absolute octave after the `=` symbol. This example will generate a warning (and change the pitch) because the second note is the absolute octave `d''` instead of `d'` as indicated by the octave correction.

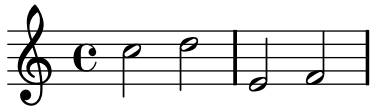
```
\relative c'' {
  c2 d='4 d
  e2 f
}
```



The octave of notes may also be checked with the `\octave` *controlpitch* command. *controlpitch* is specified in absolute mode. This checks that the interval between the previous note and the *controlpitch* is within a fourth (i.e. the normal calculation of relative mode). If this check fails, a warning is printed, but the previous note is not changed. Future notes are relative to the *controlpitch*.

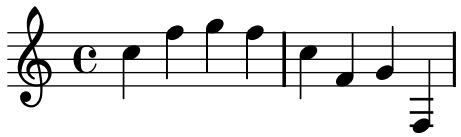
```
\relative c'' {
  c2 d
  \octave c'
  e2 f
```

```
}
```



Compare the two bars below. The first and third `\octave` check fail, but the second one does not fail.

```
\relative c'' {
  c4 f g f

  c4
  \octave c'
  f
  \octave c'
  g
  \octave c'
  f
}
```



See also

Snippets: Pitches .

1.1.2.2 Transpose

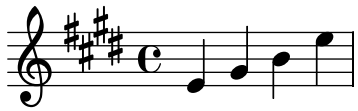
A music expression can be transposed with `\transpose`. The syntax is

`\transpose` *`frompitch topitch musicexpr`*

This means that *musicexpr* is transposed by the interval between the pitches *frompitch* and *topitch*: any note with pitch *frompitch* is changed to *topitch* and any other note is transposed by the same interval. Both pitches are entered in absolute mode.

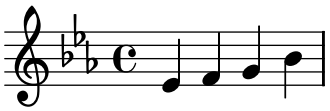
Consider a piece written in the key of D-major. It can be transposed up to E-major; note that the key signature is automatically transposed as well.

```
\transpose d e {
  \relative c' {
    \key d \major
    d4 fis a d
  }
}
```



If a part written in C (normal concert pitch) is to be played on the A clarinet (for which an A is notated as a C and thus sounds a minor third lower than notated), the appropriate part will be produced with:

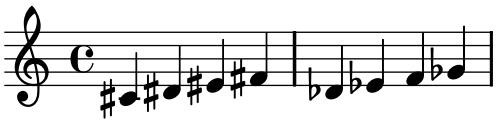
```
\transpose a c' {
  \relative c' {
    \key c \major
    c4 d e g
  }
}
```



Note that we specify `\key c \major` explicitly. If we do not specify a key signature, the notes will be transposed but no key signature will be printed.

`\transpose` distinguishes between enharmonic pitches: both `\transpose c cis` or `\transpose c des` will transpose up a semitone. The first version will print sharps and the notes will remain on the same scale step, the second version will print flats on the scale step above.

```
mus = \relative c' { c d e f }
\new Staff {
  \transpose c cis { \mus }
  \transpose c des { \mus }
}
```



`\transpose` may also be used in a different way, to input written notes for a transposing instrument. The previous examples show how to enter pitches in C (or *concert pitch*) and typeset them for a transposing instrument, but the opposite is also possible if you for example have a set of instrumental parts and want to print a conductor's score. For example, when entering music for a B-flat trumpet that begins on a notated E (concert D), one would write:

```
musicInBflat = { e4 ... }
\transpose c bes, \musicInBflat
```

To print this music in F (e.g., rearranging to a French horn) you could wrap the existing music with another `\transpose`:

```
musicInBflat = { e4 ... }
\transpose f c' { \transpose c bes, \musicInBflat }
```

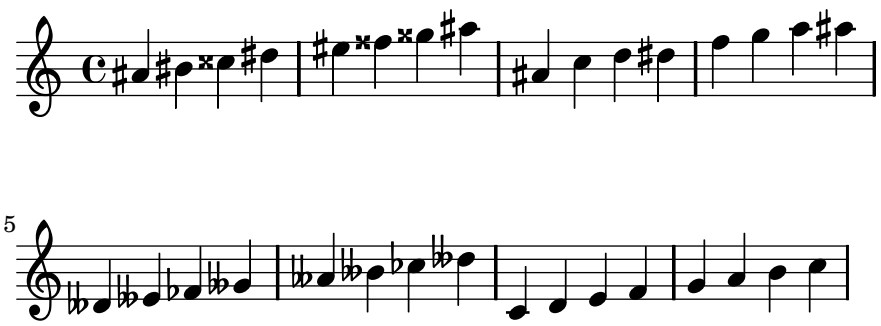
For more information about transposing instruments, see Section 1.1.3.4 [Instrument transpositions], page 17.

Selected Snippets

There is a way to enforce enharmonic modifications for notes in order to have the minimum number of accidentals. In that case, “Double accidentals should be removed, as well as E-sharp (-> F), bC (-> B), bF (-> E), B-sharp (-> C).”, as proposed by a request for a new feature. In this manner, the most natural enharmonic notes are chosen in this example.

```
#(define  (naturalise-pitch p)
  (let* ((o (ly:pitch-octave p))
         (a (* 4 (ly:pitch-alteration p)))
    ; alteration, a, in quarter tone steps, for historical reasons
         (n (ly:pitch-notename p)))

    (cond
     ((and (> a 1) (or (eq? n 6) (eq? n 2)))
      (set! a (- a 2))
      (set! n (+ n 1)))
     ((and (< a -1) (or (eq? n 0) (eq? n 3)))
      (set! a (+ a 2))
      (set! n (- n 1))))

    (cond
     ((> a 2) (set! a (- a 4)) (set! n (+ n 1)))
     ((< a -2) (set! a (+ a 4)) (set! n (- n 1))))

    (if (< n 0) (begin (set!  o (- o 1)) (set! n (+ n 7))))
    (if (> n 6) (begin (set!  o (+ o 1)) (set! n (- n 7))))

    (ly:make-pitch o n (/ a 4))))

#(define (naturalise music)
  (let* ((es (ly:music-property music 'elements))
         (e (ly:music-property music 'element))
         (p (ly:music-property music 'pitch)))

    (if (pair? es)
        (ly:music-set-property!
         music 'elements
         (map (lambda (x) (naturalise x)) es)))

    (if (ly:music? e)
        (ly:music-set-property!
         music 'element
         (naturalise e)))

    (if (ly:pitch? p)
        (begin
          (set! p (naturalise-pitch p))
          (ly:music-set-property! music 'pitch p)))

    music))

music =  \relative c' { c4 d  e f g a b  c }
```

```
naturaliseMusic =
#(define-music-function (parser location m)
                                          (ly:music?)
                         (naturalise m))

\score {
   \new Staff {
     \transpose c ais \music
     \naturaliseMusic \transpose c ais \music
    \break
    \transpose c deses \music
    \naturaliseMusic \transpose c deses \music
  }
  \layout { ragged-right = ##t}
}
```



See also

Notation Reference: Section 1.1.3.4 [Instrument transpositions], page 17.

Snippets: Pitches .

Internals Reference: `TransposedMusic`.

Known issues and warnings

The relative conversion will not affect `\transpose`, `\chordmode` or `\relative` sections in its argument. To use relative mode within transposed music, an additional `\relative` must be placed inside `\transpose`.

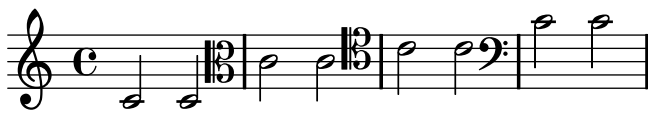
1.1.3 Displaying pitches

This section discusses how to alter the output of pitches.

1.1.3.1 Clef

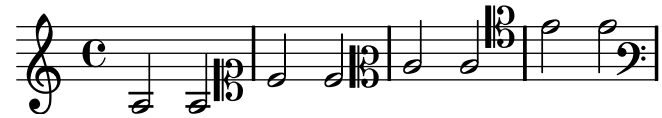
The clef is set with the `\clef` *clefname* command. Middle C is shown in every example.

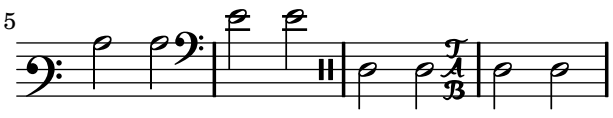
```
\clef treble
c2 c
\clef alto
c2 c
\clef tenor
c2 c
\clef bass
c2 c
```



Other clefs include:

```
\clef french
c2 c
\clef soprano
c2 c
\clef mezzosoprano
c2 c
\clef baritone
c2 c

\break

\clef varbaritone
c2 c
\clef subbass
c2 c
\clef percussion
c2 c
\clef tab
c2 c
```



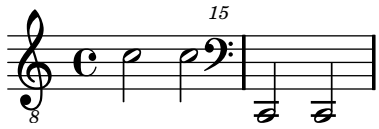


Further supported clefs are described under Section 2.8.1.4 [Ancient clefs], page 169.

By adding `_8` or `^8` to the clef name, the clef is transposed one octave down or up, respectively, and `_15` and `^15` transposes by two octaves. The argument *clefname* must be enclosed in quotes when it contains underscores or digits.

```
\clef "treble_8"
c2 c
```

```
\clef "bass^15"
c2 c
```



Selected Snippets

The command `\clef "treble_8"` is equivalent to setting `clefGlyph`, `clefPosition` (which controls the Y position of the clef), `middleCPosition` and `clefOctavation`. A clef is printed when any of these properties are changed.

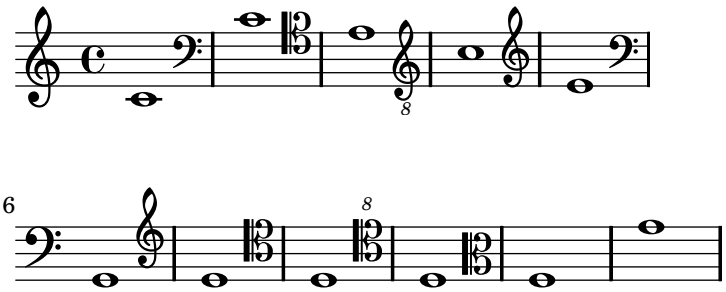
Note that changing the glyph, the position of the clef, or the octavation, does not in itself change the position of subsequent notes on the staff: the position of middle C must also be specified to do this. The positional parameters are relative to the staff centre line, positive numbers displacing upwards, counting 1 for each line and space. The clefOctavation value would normally be set to 7, -7, 15 or -15, but other values are not invalid.

When a clef change takes place at a line break the new clef symbol is printed at both the end of the previous line and the beginning of the new line by default. If the warning clef at the end of the previous line is not required it can be suppressed by setting the explicitClefVisibility Staff property to the value `end-of-line-invisible`. The default behaviour can be recovered with `\unset Staff.explicitClefVisibility`.

The following examples show the possibilities when setting these properties manually. On the first line, the manual changes preserve the standard relative positioning of clefs and notes, whereas on the second line, they do not.

```
{
  % The default treble clef
  c'1
  % The standard bass clef
  \set Staff.clefGlyph = #"clefs.F"
  \set Staff.clefPosition = #2
  \set Staff.middleCPosition = #6
  c'
  % The baritone clef
  \set Staff.clefGlyph = #"clefs.C"
  \set Staff.clefPosition = #4
  \set Staff.middleCPosition = #4
  c'
  % The standard choral tenor clef
  \set Staff.clefGlyph = #"clefs.G"
  \set Staff.clefPosition = #-2
  \set Staff.clefOctavation = #-7
  \set Staff.middleCPosition = #1
  c'
  % A non-standard clef
  \set Staff.clefPosition = #0
  \set Staff.clefOctavation = #0
  \set Staff.middleCPosition = #-4
  c' \break
```

```
  % The following clef changes do not preserve
  % the normal relationship between notes and clefs:

  \set Staff.clefGlyph = #"clefs.F"
  \set Staff.clefPosition = #2
  c'
  \set Staff.clefGlyph = #"clefs.G"
  c'
  \set Staff.clefGlyph = #"clefs.C"
  c'
  \set Staff.clefOctavation = #7
  c'
  \set Staff.clefOctavation = #0
  \set Staff.clefPosition = #0
  c'

  % Here we go back to the normal clef:

  \set Staff.middleCPosition = #4
  c'
}
```



See also

Notation Reference: Section 2.8.1.4 [Ancient clefs], page 169.

Snippets: Pitches .

Internals Reference: `Clef`.

1.1.3.2 Key signature

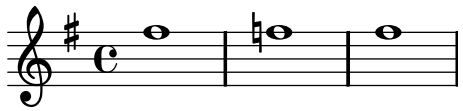
> **Note:** New users are sometimes confused about accidentals and key signatures. In LilyPond, note names are the raw input; key signatures and clefs determine how this raw input is displayed. An unaltered note like `c` means 'C natural', regardless of the key signature or clef. For more information, see learning manual, ⟨undefined⟩ [Accidentals and key signatures], page ⟨undefined⟩ .

The key signature indicates the tonality in which a piece is played. It is denoted by a set of alterations (flats or sharps) at the start of the staff. Setting or changing the key signature is done with the `\key` command:

`\key` *`pitch mode`*

Here, *mode* should be `\major` or `\minor` to get a key signature of *pitch*-major or *pitch*-minor, respectively. You may also use the standard mode names, also called 'church modes': `\ionian`, `\dorian`, `\phrygian`, `\lydian`, `\mixolydian`, `\aeolian`, and `\locrian`.

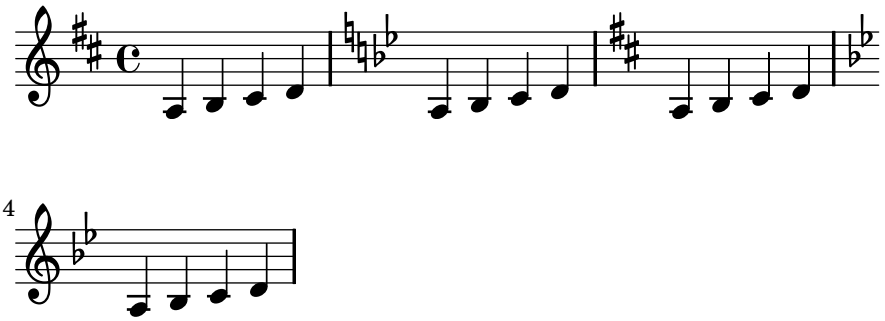
```
\key g \major
fis1
f
fis
```



Selected Snippets

When the key signature changes, natural signs are automatically printed to cancel any accidentals from previous key signatures. This may be altered by setting to "false" the printKeyCancellation property in the Staff context.

```
\relative {
  \key d \major
  a b cis d
  \key g \minor
  a bes c d
  \set Staff.printKeyCancellation = ##f
  \key d \major
  a b cis d
  \key g \minor
  a bes c d
}
```



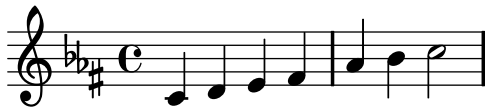
The commonly used `\key` command sets the `keySignature` property, in the `Staff` context. However, non-standard key signatures can be specified by setting this property directly. The format of this command is a list: `\set Staff.keySignature = #'(((octave . step) . alter) ((octave . step) . alter) ...)` where, for each element in the list, octave specifies the octave (0 being the octave from middle C to the B above), step specifies the note within the octave

(0 means C and 6 means B), and alter is ,SHARP ,FLAT ,DOUBLE-SHARP etc. (Note the leading comma.)

However, for each item in the list, you can also use the alternative format (step . alter), which specifies that the same alteration should hold in all octaves.

Here is an example of a possible key signature for generating a whole-tone scale:

```
\relative c' {
  \set Staff.keySignature =
    #`(((0 .  3) . ,SHARP) ((0 . 5) . ,FLAT) ((0 . 6) . ,FLAT))
  c d e fis aes bes c2
}
```



See also

Music Glossary: ⟨undefined⟩ [church mode], page ⟨undefined⟩, ⟨undefined⟩ [scordatura], page ⟨undefined⟩.

Learning Manual: learning manual, ⟨undefined⟩ [Accidentals and key signatures], page ⟨undefined⟩ .

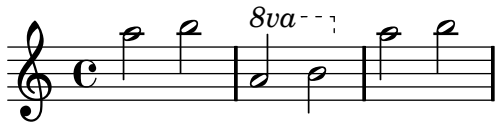
Snippets: Pitches .

Internals Reference: `KeyCancellation`, `KeySignature`, `Key_engraver`.

1.1.3.3 Ottava brackets

Ottava brackets introduce an extra transposition of an octave for the staff:

```
a'2 b
#(set-octavation 1)
a b
#(set-octavation 0)
a b
```

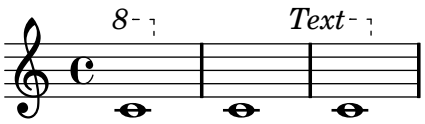


The `set-octavation` function also takes -1 (for 8va bassa), 2 (for 15ma), and -2 (for 15ma bassa) as arguments.

Selected Snippets

Internally, the set-octavation function sets the properties `ottavation` (e.g., to `"8va"` or `"8vb"`) and `middleCPosition`. To override the text of the bracket, set `ottavation` after invoking `set-octavation`, like in the following example.

```
{
  #(set-octavation 1)
  \set Staff.ottavation = #"8"
  c''1
  #(set-octavation 0)
  c'1
  #(set-octavation 1)
  \set Staff.ottavation = #"Text"
  c''1
}
```



See also

Music Glossary: ⟨undefined⟩ [octavation], page ⟨undefined⟩.

Snippets: Pitches .

Internals Reference: `OttavaBracket`.

1.1.3.4 Instrument transpositions

When typesetting scores that involve transposing instruments, some parts can be typeset in a different pitch than the *concert pitch*. In these cases, the key of the *transposing instrument* should be specified; otherwise the MIDI output and cues in other parts will produce incorrect pitches. For more information about quotations, see Section 1.6.2.3 [Quoting other voices], page 101.

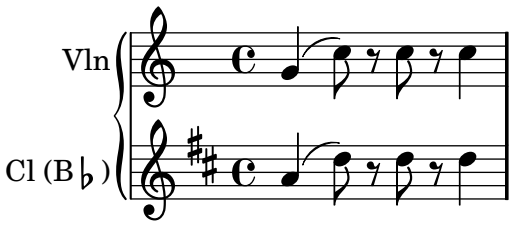
`\transposition` *pitch*

The pitch to use for `\transposition` should correspond to the real sound heard when a `c'` written on the staff is played by the transposing instrument. This pitch is entered in absolute mode, so an instrument that produces a real sound which is one tone higher than the printed music should use `\transposition d'`. `\transposition` should *only* be used if the pitches are *not* being entered in concert pitch.

Here are a few notes for violin and B-flat clarinet where the parts have been entered using the notes and key as they appear in each part of the conductor's score. The two instruments are playing in unison.

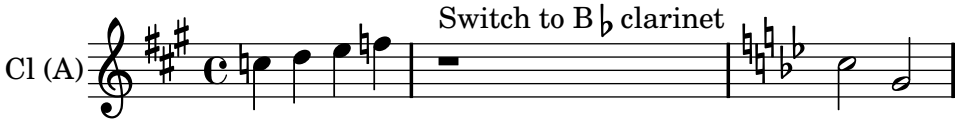
```
\new GrandStaff <<
  \new Staff = "Vln" {
    \relative c'' {
      \set Staff.instrumentName = "Vln"
      \set Staff.midiInstrument="violin"
      % strictly speaking not necessary, but a good reminder
```

```
      \transposition c'

      \key c \major
      g4( c8) r c r c4
    }
  }
  \new Staff = "clarinet" {
    \relative c'' {
      \set Staff.instrumentName = \markup {Cl (B\flat)}
      \set Staff.midiInstrument="clarinet"
      \transposition bes

      % not concert pitch
      \key d \major
      a4( d8) r d r d4
    }
  }
>>
```



The `\transposition` may be changed during a piece. For example, a clarinetist may switch from an A clarinet to a B-flat clarinet.

```
\set Staff.instrumentName = "Cl (A)"
\key a \major
\transposition a
c d e f
\textLengthOn
r1^\markup{Switch to B\flat clarinet}

\key bes \major
\transposition bes
c2 g
```



See also

Music Glossary: ⟨undefined⟩ [concert pitch], page ⟨undefined⟩, ⟨undefined⟩ [transposing instrument], page ⟨undefined⟩.

Notation Reference: Section 1.6.2.3 [Quoting other voices], page 101, Section 1.1.2.2 [Transpose], page 8.

Snippets: Pitches .

1.1.3.5 Automatic accidentals

There are many different conventions on how to typeset accidentals. LilyPond provides a function to specify which accidental style to use. This function is called as follows

```
\new Staff <<
  #(set-accidental-style 'voice)
  { ... }
>>
```

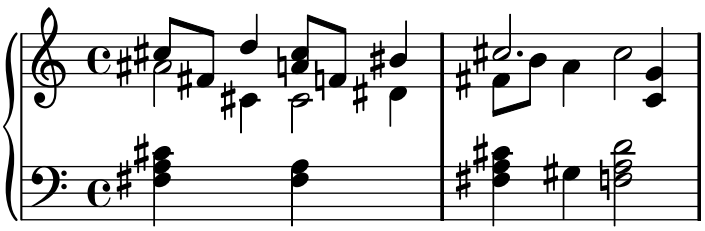
The accidental style applies to the current `Staff` by default (with the exception of the styles `piano` and `piano-cautionary`, which are explained below). Optionally, the function can take a second argument that determines in which scope the style should be changed. For example, to use the same style in all staves of the current `StaffGroup`, use

```
#(set-accidental-style 'voice 'StaffGroup)
```

The following accidental styles are supported. To demonstrate each style, we use the following example:

```
musicA = {
  <<
    \relative c' {
      cis'8 fis, d'4 <a cis>8 f bis4 |
      cis2. <c, g'>4 |
    }
    \\
    \relative c' {
      ais'2 cis, |
      fis8 b a4 cis2 |
    }
  >>
}

musicB = {
  \clef bass
  \new Voice {
    \voiceTwo \relative c' {
      <fis, a cis>4
      \change Staff = up
      cis'
      \change Staff = down
      <fis, a>
      \change Staff = up
      dis' |
      \change Staff = down
      <fis, a cis>4 gis <f a d>2 |
    }
  }
}

\new PianoStaff {
```

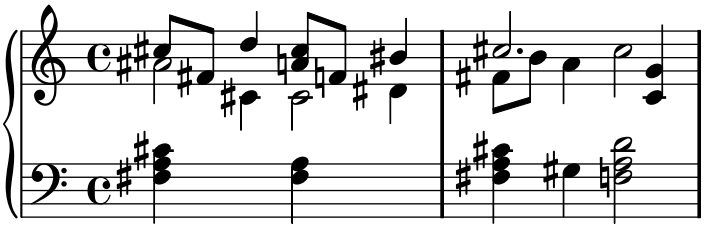
```
    <<
      \context Staff = "up" {
        %%% change the next line as desired:
        #(set-accidental-style 'default)
        \musicA
      }
      \context Staff = "down" {
        %%% change the next line as desired:
        #(set-accidental-style 'default)
        \musicB
      }
    >>
  }
```



Note that the last lines of this example can be replaced by the following, as long as the same accidental style should be used in both staves.

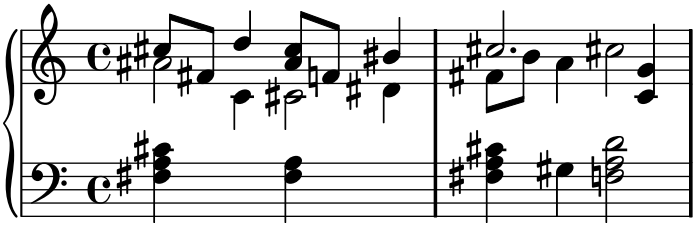
```
  \new PianoStaff {
    <<
      \context Staff = "up" {
        %%% change the next line as desired:
        #(set-accidental-style 'default 'Score)
        \musicA
      }
      \context Staff = "down" {
        \musicB
      }
    >>
  }
```

`default` This is the default typesetting behavior. It corresponds to eighteenth-century common practice: accidentals are remembered to the end of the measure in which they occur and only on their own octave. Thus, in the example below, no natural signs are printed before the `b` in the second measure or the last `c`:



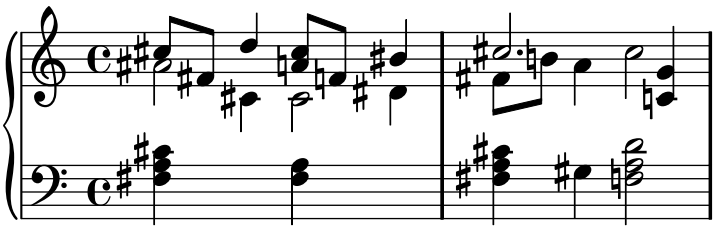
`voice` The normal behavior is to remember the accidentals on Staff-level. In this style, however, accidentals are typeset individually for each voice. Apart from that, the rule is similar to `default`.

As a result, accidentals from one voice do not get canceled in other voices, which is often an unwanted result: in the following example, it is hard to determine whether the second `a` should be played natural or sharp. The `voice` option should therefore be used only if the voices are to be read solely by individual musicians. If the staff is to be used by one musician (e.g., a conductor or in a piano score) then `modern` or `modern-cautionary` should be used instead.



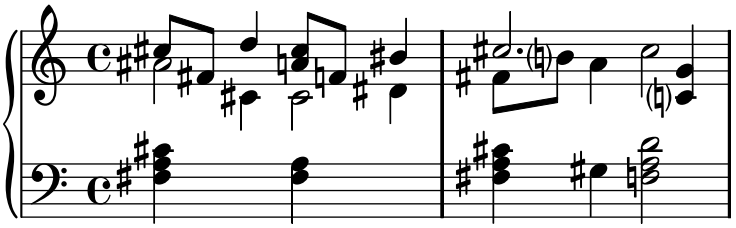
`modern`

This rule corresponds to the common practice in the twentieth century. It prints the same accidentals as `default`, with two exceptions that serve to avoid ambiguity: after temporary accidentals, cancellation marks are printed also in the following measure (for notes in the same octave) and, in the same measure, for notes in other octaves. Hence the naturals before the `b` and the `c` in the second measure of the upper staff:



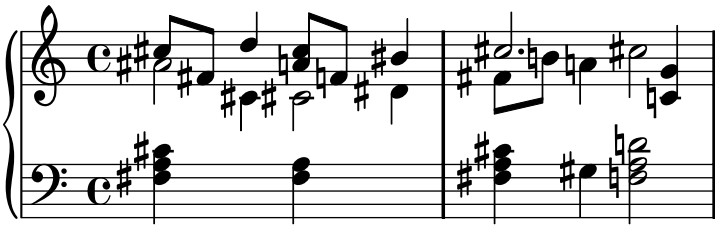
`modern-cautionary`

This rule is similar to `modern`, but the 'extra' accidentals (the ones not typeset by `default`) are typeset as cautionary accidentals. They are by default printed with parentheses, but they can also be printed in reduced size by defining the `cautionary-style` property of `AccidentalSuggestion`.



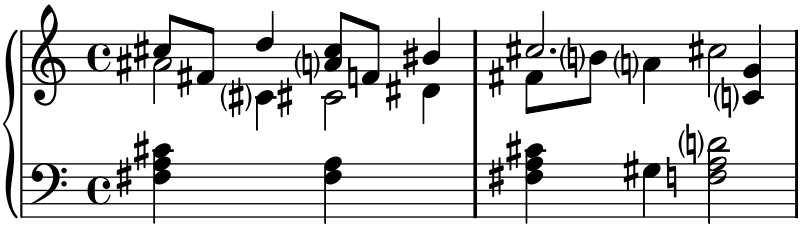
`modern-voice`

This rule is used for multivoice accidentals to be read both by musicians playing one voice and musicians playing all voices. Accidentals are typeset for each voice, but they *are* canceled across voices in the same `Staff`. Hence, the `a` in the last measure is canceled because the previous cancellation was in a different voice, and the `d` in the lower staff is canceled because of the accidental in a different voice in the previous measure:



`modern-voice-cautionary`

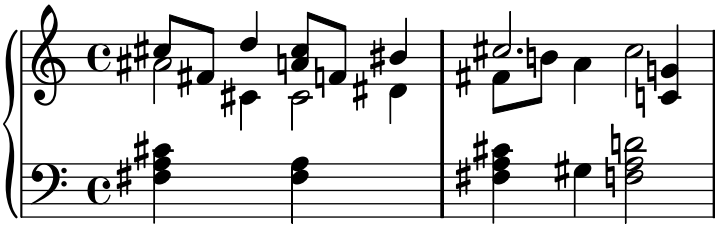
This rule is the same as `modern-voice`, but with the extra accidentals (the ones not typeset by `voice`) typeset as cautionaries. Even though all accidentals typeset by `default` *are* typeset with this rule, some of them are typeset as cautionaries.



`piano`

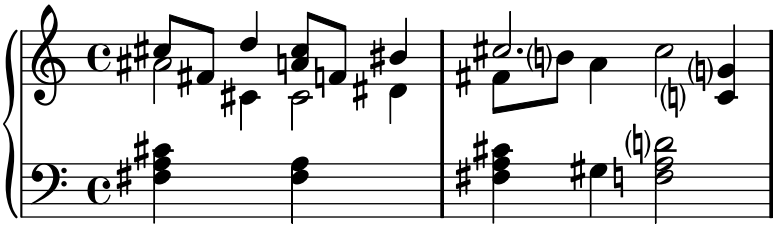
This rule reflects twentieth-century practice for piano notation. Its behavior is very similar to `modern` style, but here accidentals also get canceled across the staves in the same GrandStaff or PianoStaff, hence all the cancellations of the final notes.

This accidental style applies to the current `GrandStaff` or `PianoStaff` by default.



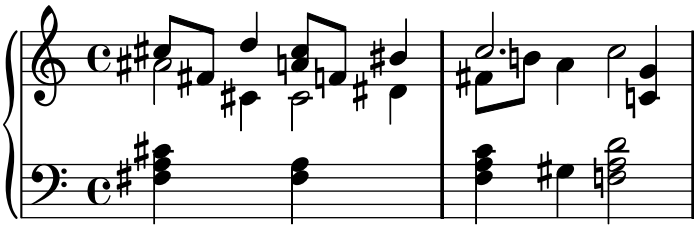
`piano-cautionary`

Same as `#(set-accidental-style 'piano)` but with the extra accidentals typeset as cautionaries.



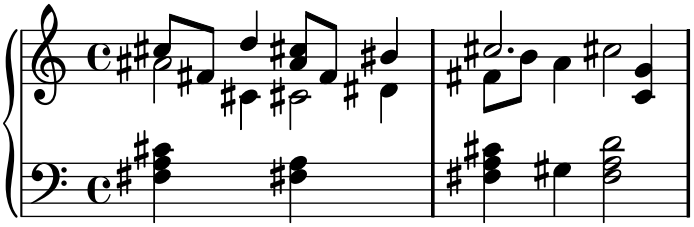
`no-reset`

This is the same as `default` but with accidentals lasting 'forever' and not only within the same measure:



`forget`

This is the opposite of `no-reset`: Accidentals are not remembered at all – and hence all accidentals are typeset relative to the key signature, regardless of what was before in the music:

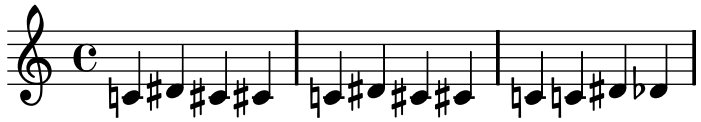


Selected Snippets

In early XXth century works, starting with Schönberg, Berg and Webern (the "second" Viennese school), every pitch in the twelve-tone scale has to be regarded as equal, without any hierarchy such as the classical (tonal) degrees. Therefore, these composers print one accidental for each note, even at natural pitches, to emphasize their new approach to music theory and language. This snippet shows how to achieve such notation rules with LilyPond.

```
webernAccidentals = {
    % the 5s are just "a value different from any accidental"
    \set Staff.keySignature = #'((0 . 5) (1 . 5) (2 . 5) (3 . 5)
                                 (4 . 5) (5 . 5) (6 . 5))
    \set Staff.extraNatural = ##f
    #(set-accidental-style 'forget)
}

\layout {
  \context { \Staff \remove Key_engraver }
}
\score {
```

```
  {
    \webernAccidentals
    c' dis' cis' cis'
    c' dis' cis' cis'
    c' c' dis' des'
  }
}
```



See also

Snippets: Pitches .

Internals Reference: `Accidental_engraver`, `Accidental`, `AccidentalSuggestion`, `AccidentalPlacement`, `GrandStaff` and `PianoStaff`, `Staff`.

Known issues and warnings

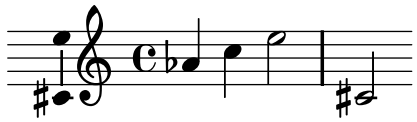
Simultaneous notes are considered to be entered in sequential mode. This means that in a chord the accidentals are typeset as if the notes in the chord happen one at a time, in the order in which they appear in the input file. This is a problem when accidentals in a chord depend on each other, which does not happen for the default accidental style. The problem can be solved by manually inserting `!` and `?` for the problematic notes.

1.1.3.6 Ambitus

The term *ambitus* or *ambit* denotes a range of pitches for a given voice in a part of music. It may also denote the pitch range that a musical instrument is capable of playing. Ambits are printed on vocal parts so that performers can easily determine if it matches their capabilities.

Ambits are denoted at the beginning of a piece near the initial clef. The range is graphically specified by two note heads that represent the lowest and highest pitches. Accidentals are only printed if they are not part of the key signature.

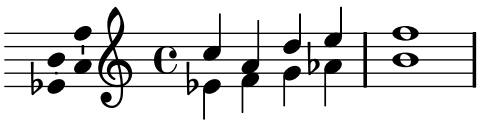
```
\layout {
  \context {
    \Voice
    \consists Ambitus_engraver
  }
}

\relative c'' {
  aes c e2 cis,2
}
```



Selected Snippets

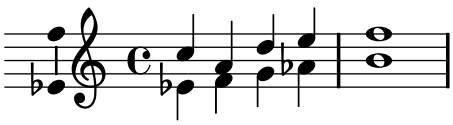
Ambits can be added per voice. In that case, the ambitus must be moved manually to prevent collisions.

```
\new Staff <<
  \new Voice \with {
    \consists "Ambitus_engraver"
  } \relative c'' {
    \override Ambitus #'X-offset = # 2.0
    \voiceOne
    c4 a d e f1
  }
  \new Voice \with {
    \consists "Ambitus_engraver"
  } \relative c' {
    \voiceTwo
        es4 f g as b1
  }
>>
```



If you have multiple voices in a single staff and you want a single ambitus per staff rather than per voice, add the `Ambitus_engraver` to the `Staff` context rather than to the `Voice` context.

```
\new Staff \with {
  \consists "Ambitus_engraver"
  }
<<
  \new Voice \relative c'' {
      \voiceOne
      c4 a d e f1
    }
  \new Voice \relative c' {
      \voiceTwo
      es4 f g as b1
    }
>>
```



See also

Music Glossary: ⟨undefined⟩ [ambitus], page ⟨undefined⟩.

Snippets: Pitches .

Internals Reference: `Ambitus`, `AmbitusLine`, `AmbitusNoteHead`, `AmbitusAccidental`, `Ambitus_engraver`, `Staff`, `Voice`.

Known issues and warnings

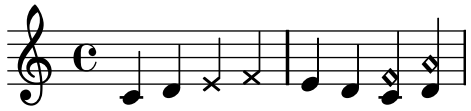
There is no collision handling in the case of multiple per-voice ambitus.

1.1.4 Note heads

1.1.4.1 Special note heads

Different note heads are used by various instruments for various meanings – crosses are used for 'parlato' with vocalists, stopped notes on guitar; diamonds are used for harmonics on string instruments, etc. There is a shorthand (`\harmonic`) for diamond shapes; the other note head styles are produced by tweaking the property:

```
c4 d
\override NoteHead #'style = #'cross
e f
\revert NoteHead #'style
e d <c f\harmonic> <d a'\harmonic>
```



To see all note head styles, see Section B.5 [Note head styles], page 305.

See also

Snippets: Pitches .

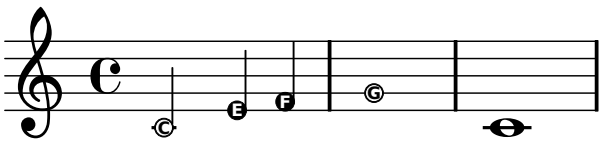
Notation Reference: Section B.5 [Note head styles], page 305.

Internals Reference: `NoteHead`, `LedgerLineSpanner`.

1.1.4.2 Easy notation note heads

The 'easy play' note head includes a note name inside the head. It is used in music for beginners.

```
#(set-global-staff-size 26)
\relative c' {
  \easyHeadsOn
  c2 e4 f
  g1
  \easyHeadsOff
  c,1
}
```



The command `\easyHeadsOn` overrides settings for the `NoteHead` object. These settings can be reverted with the command `\easyHeadsOff`. To make the letters readable, it has to be printed in a large font size. To print with a larger font, see Section 5.2.1 [Setting the staff size], page 213.

Predefined commands

`\easyHeadsOn`, `\easyHeadsOff`

See also

Notation Reference: Section 5.2.1 [Setting the staff size], page 213.

Snippets: Pitches ,

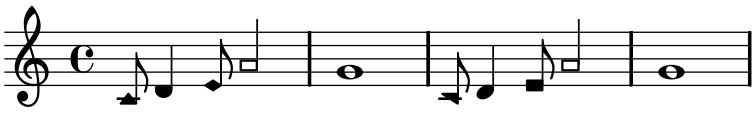
Internals Reference: `NoteHead`.

1.1.4.3 Shape note heads

In shape note head notation, the shape of the note head corresponds to the harmonic function of a note in the scale. This notation was popular in nineteenth-century American song books.

Shape note heads can be produced by setting `\aikenHeads` or `\sacredHarpHeads`, depending on the style desired.

```
\aikenHeads
c8 d4 e8 a2 g1
\sacredHarpHeads
c,8 d4 e8 a2 g1
```



Shapes are typeset according to the step in the scale, where the base of the scale is determined by the `\key` command.

Selected Snippets

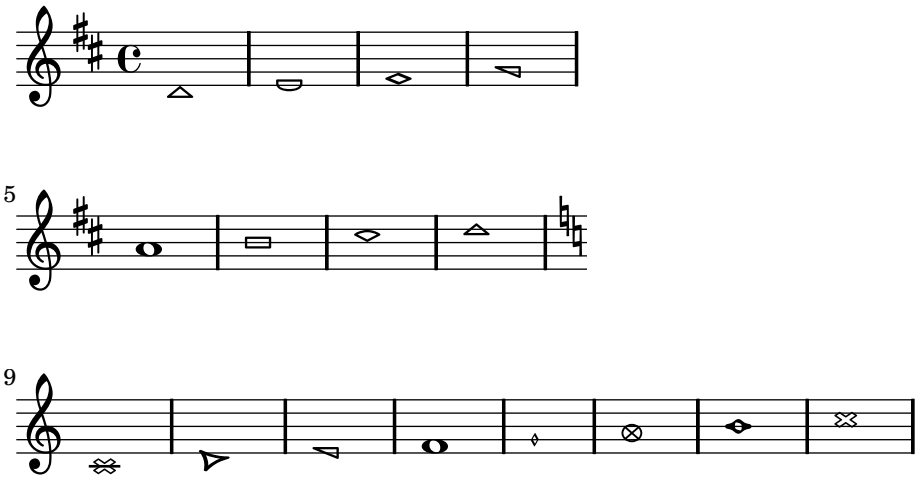
The shapeNoteStyles property gives you the ability to define various note heads styles for each step of the scale (as defined by the key signature or the "tonic" property).

This property requires a set of symbols, which can be purely arbitrary (geometrical expressions such as triangle, cross, xcircle etc. are allowed) or based on old American engraving tradition (you can use some latin note names as well).

That said, if you're trying to imitate old American song books, you may also want to try LilyPond's predefined note heads styles, through shortcut commands such as \aikenHeads or \sacredHarpHeads.

This example shows different ways to obtain shape note heads, and demonstrates the ability to transpose a melody without losing the correspondance between harmonic functions and note heads styles.

```
fragment = {
  \key c \major
  c1 d e f g a b c
  \break
}

\score {
  \new Staff {
    \transpose c d
    \relative {
      \set shapeNoteStyles = ##(do re mi fa #f la ti)
      \fragment \break
    }

    \relative {
      \set shapeNoteStyles  = ##(cross triangle fa #f mensural xcircle diamond)
      \fragment
    }
  }
}
```



See also

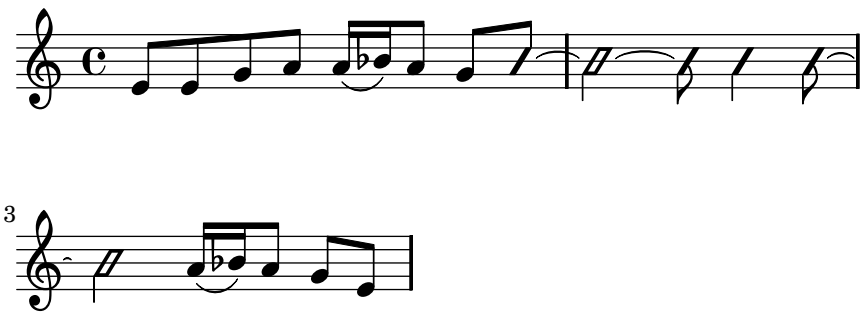
Snippets: Pitches .

1.1.4.4 Improvisation

Improvisation is sometimes denoted with slashed note heads, where the performer may choose any pitch but should play the specified rhythm. Such note heads can be created:

```
\new Voice \with {
  \consists Pitch_squash_engraver
} \transpose c c' {
  e8 e g a a16( bes) a8 g
  \improvisationOn
```

```
    e8
    ~e2~e8 f4 fis8
    ~fis2
    \improvisationOff
    a16(bes) a8 g e
}
```



Predefined commands

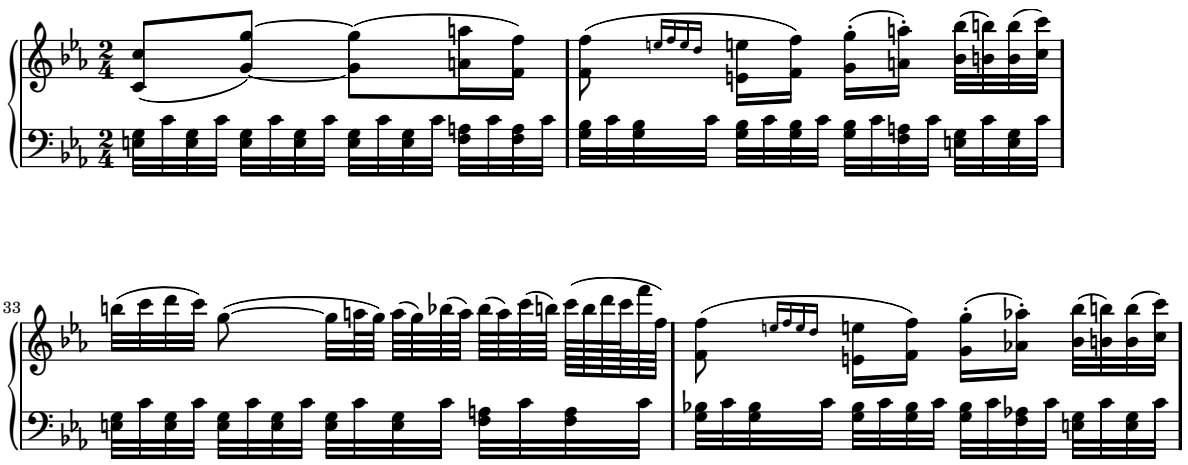
`\improvisationOn`, `\improvisationOff`

See also

Snippets: Pitches .

Internals Reference: `Pitch_squash_engraver`, `Voice`.

1.2 Rhythms



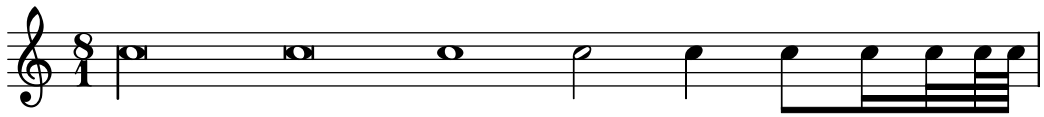
This section discusses rhythms, rests, durations, beaming and bars.

1.2.1 Writing rhythms

1.2.1.1 Durations

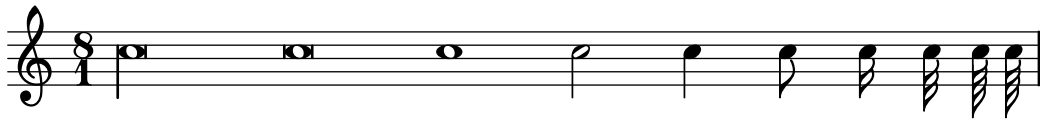
Durations are designated by numbers and dots. Durations are entered as their reciprocal values. For example, a quarter note is entered using a `4` (since it is a 1/4 note), and a half note is entered using a `2` (since it is a 1/2 note). For notes longer than a whole you must use the `\longa` (a double breve) and `\breve` commands. Durations as short as 64th notes may be specified. Shorter values are possible, but only as beamed notes.

```
\time 8/1
c\longa c\breve c1 c2
c4 c8 c16 c32 c64 c64
```



Here are the same durations with automatic beaming turned off.

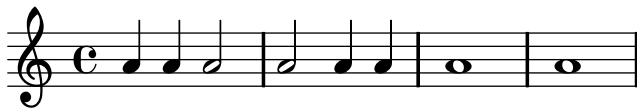
```
\time 8/1
\autoBeamOff
c\longa c\breve c1 c2
c4 c8 c16 c32 c64 c64
```



A note with the duration of a quadruple breve may be entered with `\maxima`, but this is supported only within ancient music notation. For details, see Section 2.8 [Ancient notation], page 167.

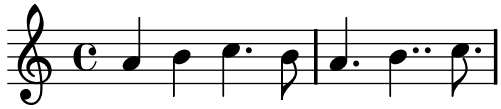
If the duration is omitted, it is set to the previously entered duration. The default for the first note is a quarter note.

```
a a a2 a a4 a a1 a
```



To obtain dotted note lengths, place a dot (`.`) after the duration. Double-dotted notes are specified by appending two dots, and so on.

```
a4 b c4. b8 a4. b4.. c8.
```



Some durations cannot be represented with just binary durations and dots; they can be represented only by tying two or more notes together. For details, see Section 1.2.1.4 [Ties], page 35.

For ways of specifying durations for the syllables of lyrics and ways of aligning lyrics to notes, see Section 2.1 [Vocal music], page 129.

Optionally, notes can be spaced strictly proportionately to their duration. For details of this and other settings which control proportional notation, see Section 5.6.5 [Proportional notation], page 239.

Predefined commands

Dots are normally moved up to avoid staff lines, except in polyphonic situations. The following commands may be used to force a particular direction manually, and to return to the default behaviour:

`\dotsUp`, `\dotsDown`, `\dotsNeutral`.

See also

Music Glossary: ⟨undefined⟩ [breve], page ⟨undefined⟩, ⟨undefined⟩ [longa], page ⟨undefined⟩, ⟨undefined⟩ [note value], page ⟨undefined⟩ ⟨undefined⟩ [Duration names notes and rests], page ⟨undefined⟩.

Notation Reference: Section 1.2.4.1 [Automatic beams], page 49, Section 1.2.1.4 [Ties], page 35, Section 1.2.1 [Writing rhythms], page 29, Section 1.2.2 [Writing rests], page 37, Section 2.1 [Vocal music], page 129, Section 2.8 [Ancient notation], page 167, Section 5.6.5 [Proportional notation], page 239.

Snippets: Rhythms

Internals Reference: `Dots`, `DotColumn`.

Known issues and warnings

There is no fundamental limit to rest durations (both in terms of longest and shortest), but the number of glyphs is limited: rests from 128th to maxima (8 x whole) may be printed.

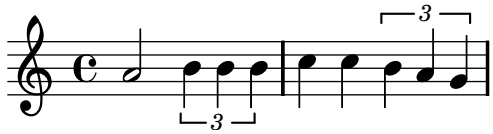
1.2.1.2 Tuplets

Tuplets are made out of a music expression by multiplying all durations with a fraction:

```
\times fraction musicexpr
```

The duration of *musicexpr* will be multiplied by the fraction. The fraction's denominator will be printed over or under the notes, optionally with a bracket. The most common tuplet is the triplet in which 3 notes have the length of 2, so the notes are 2/3 of their written length.

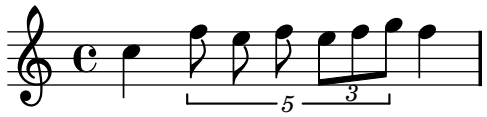
```
a2 \times 2/3 {b4 b b}
c4 c \times 2/3 {b4 a g}
```



Tuplets may be nested:

```
\relative c'' {
  \autoBeamOff
  c4 \times 4/5 { f8 e f \times 2/3 {e[ f g] } } f4 |
```

```
}
```



Predefined commands

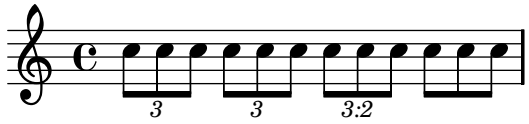
`\tupletUp`, `\tupletDown`, `\tupletNeutral`.

Selected Snippets

For more information about `make-moment`, see Section 1.2.6.3 [Time administration], page 67.

By default, only the numerator of the tuplet number is printed over the tuplet bracket, i.e. the denominator of the argument to the `\times` command. Alternatively, *num*:*den* of the tuplet number may be printed, or the tuplet number may be suppressed altogether.

```
\times 2/3 { c8 c c } \times 2/3 { c8 c c }
\override TupletNumber #'text = #tuplet-number::calc-fraction-text
\times 2/3 { c8 c c }
\override TupletNumber #'transparent = ##t
\times 2/3 { c8 c c }
```

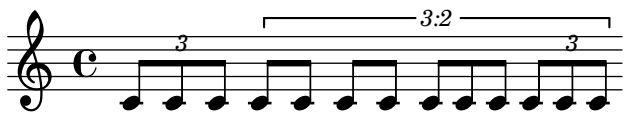


Tuplets may extend over bar lines, but they will inhibit a line break unless the `Forbid_line_break_engraver` is removed from the `Voice context`.

Modifying nested tuplets

If nested tuplets do not begin at the same moment their appearance may be modified in the usual way with `\override` commands:

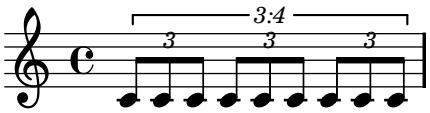
```
\times 2/3 { c8[ c c]}
\once \override TupletNumber #'text = #tuplet-number::calc-fraction-text
\times 2/3 {
  c[ c]
  c[ c]
  \once \override TupletNumber #'transparent = ##t
  \times 2/3 { c8[ c c] }
\times 2/3 { c8[ c c]}
}
```



However, if the nested tuplets begin at the same musical moment, `\override` commands cannot be applied to just one of them – they apply to both. So to change the appearance of nested tuplets beginning at the same musical moment individually, the `\tweak` function must be used (see Section 6.2.5 [Objects connected to the input], page 262). The `\tweak` function is applied to the following `\times` command as it appears in the input stream, and so can distinguish between separate `\times` commands even if their tuplets begin at the same musical moment.

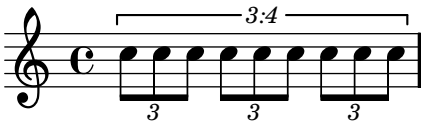
In this example, the `\tweak` command is used to specify fraction text for the outer `TupletNumber` and denominator text for the `TupletNumber` of the first of the three inner tuplets.

```
\tweak #'text #tuplet-number::calc-fraction-text
\times 4/3 {
   \tweak #'text #tuplet-number::calc-denominator-text
   \times 2/3 { c8[ c8 c8] }
   \times 2/3 { c8[ c8 c8] }
   \times 2/3 { c8[ c8 c8] }
}
```



In the next example, `\tweak` and `\override` work together to specify `TupletBracket` direction. The first `\tweak` positions the `TupletBracket` of the outer tuplet above the staff. The second `\tweak` positions the `TupletBracket` of the first of the three inner tuplets below the staff. Note that the `\tweak` command needs to be used only for events that begin at the same music moment: the outer tuplet and the first of the three inner tuplets. To position the `TupletBracket`s of the second and third of the inner tuplets below the staff, we can use `\override` in the usual way.

```
\tweak #'text #tuplet-number::calc-fraction-text
\tweak #'direction #up
\times 4/3 {
   \tweak #'direction #down
   \times 2/3 { c8[ c8 c8] }
   \override TupletBracket #'direction = #down
   \times 2/3 { c8[ c8 c8] }
   \times 2/3 { c8[ c8 c8] }
}
```



Modifying tuplet bracket length

Tuplet brackets can be made to run to prefatory matter or the next note. Default tuplet brackets end at the right edge of the final note of the tuplet; full-length tuplet brackets extend farther to the right, either to cover all the non-rhythmic notation up to the following note, or to cover only the whitespace before the next item of notation, be that a clef, time signature, key signature, or another note. The example shows how to switch tuplets to full length mode and how to modify what material they cover.

```
\new RhythmicStaff {
  % Set tuplets to be extendable ..
  \set tupletFullLength = ##t
  % .. to cover all items up to the next note
  \set tupletFullLengthNote = ##t
  \time 2/4
  \times 2/3 { c4 c c }
  % .. or to cover just whitespace
  \set tupletFullLengthNote = ##f
  \time 4/4
  \times 4/5 { c4 c1 }
  \time 3/4
  c4
}
```

Compressing music

`\compressMusic` works similarly to \times, but does not create a tuplet bracket. One application is in polymetric notation, as shown in the following example. See Section 1.2.3.4 [Polymetric notation], page 45.

See also

Music Glossary: ⟨undefined⟩ [triplet], page ⟨undefined⟩, ⟨undefined⟩ [tuplet], page ⟨undefined⟩, ⟨undefined⟩ [polymetric], page ⟨undefined⟩.

Notation Reference: Section 1.2.6.3 [Time administration], page 67, Section 6.2.5 [Objects connected to the input], page 262, Section 1.2.3.4 [Polymetric notation], page 45.

Snippets: Rhythms .

Internals Reference: `TupletBracket`, `TupletNumber`, `TimeScaledMusic`.

Known issues and warnings

Lines may be broken within a tuplet with `\bar "" \break`, but the tuplet bracket does not correctly carry over.

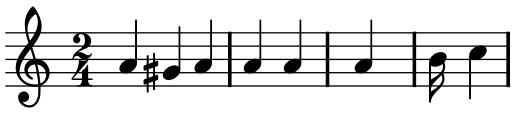
1.2.1.3 Scaling durations

You can alter the length of a duration by a fraction *N/M* by appending `*N/M` (or `*N` if *M=1*). This will not affect the appearance of the notes or rests produced, but the altered duration will be used in calculating the position within the measure and setting the duration in the MIDI output. Multiplying factors may be combined such as `*M*N`.

In the following example, the first three notes take up exactly two beats, but no triplet bracket is printed.

```
\time 2/4
a4*2/3 gis4*2/3 a4*2/3
a4 a4 a4*2
```

```
b16*4 c4
```



The duration of skip or spacing notes may also be modified by a multiplier. This is useful for skipping many measures, e.g., `s1*23`.

See also

Notation Reference: Section 1.2.1.2 [Tuplets], page 31, Section 1.2.2.2 [Skips], page 38, Section 1.2.3.4 [Polymetric notation], page 45.

Snippets: Rhythms

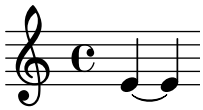
1.2.1.4 Ties

A tie connects two adjacent note heads of the same pitch. The tie in effect extends the length of a note.

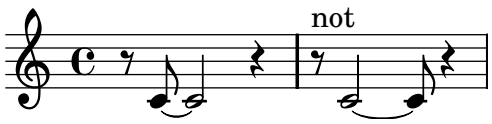
> **Note:** Ties should not be confused with *slurs*, which indicate articulation, or *phrasing slurs*, which indicate musical phrasing. A tie is just a way of extending a note duration, similar to the augmentation dot.

A tie is entered using the tilde symbol ~

```
e' ~ e'
```



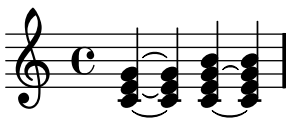
Ties are used either when the note crosses a bar line, or when dots cannot be used to denote the rhythm. Ties should also be used when note values cross larger subdivisions of the measure:



If you need to tie a lot of notes across bar lines, it may be easier to use automatic note splitting (see Section 1.2.3.5 [Automatic note splitting], page 48). This mechanism automatically splits long notes, and ties them across bar lines.

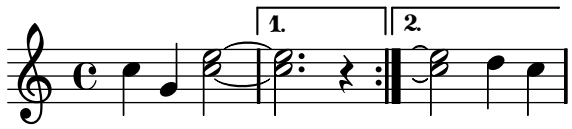
When a tie is applied to a chord, all note heads whose pitches match are connected. When no note heads match, no ties will be created. Chords may be partially tied by placing the tie inside the chord.

```
 <c e g> ~ <c e g>
<c~ e g~ b> <c e g b>
```



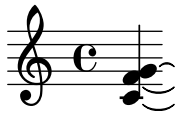
When a second alternative of a repeat starts with a tied note, you have to repeat the tie. This can be achieved with `\repeatTie`,

```
\repeat volta 2 { c g <c e>2 ~ }
\alternative {{ <c e>2. r4 } {<c e>2\repeatTie d4 c }}
```



L.v. ties (*laissez vibrer*) indicate that notes must not be damped at the end. It is used in notation for piano, harp and other string and percussion instruments. They can be entered using `\laissezVibrer`:

```
<c f g>\laissezVibrer
```

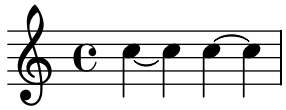


The vertical placement of ties may be controlled; see Section 3.2.1 [Controlling direction], page 194.

DELETE THIS ? `\tieDown` (see example below). `\tieNeutral` reverts to the default behaviour again.

However, as with other music elements of this kind, there is a convenient shorthand for forcing tie directions. By adding `_` or `^` before the tilde, the direction is also set:

```
c4_~ c c^~ c)
```



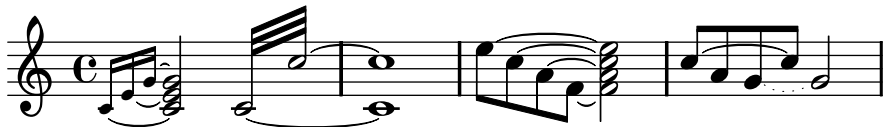
Predefined commands

`\tieUp`, `\tieDown`, `\tieNeutral`, `\tieDotted`, `\tieDashed`, `\tieSolid`.

Selected Snippets

Ties are sometimes used to write out arpeggios. In this case, two tied notes need not be consecutive. This can be achieved by setting the `tieWaitForNote` property to true. The same feature is also useful, for example, to tie a tremolo to a chord, but in principle, it can also be used for ordinary, consecutive notes:

```
\set tieWaitForNote = ##t
\grace { c16[~ e~ g]~ } <c, e g>2
\repeat tremolo 8 { c32~ c'~ } <c c,>1
e8~ c~ a~ f~ <e' c a f>2
\tieUp c8~ a \tieDown \tieDotted g~ c g2
```



Ties may be engraved manually by changing the `tie-configuration` property of the `TieColumn` object. The first number indicates the distance from the center of the staff in staff-spaces, and the second number indicates the direction (1=up, -1=down).

```
<c e g>2~ <c e g> |
\override TieColumn #'tie-configuration =
  #'((0.0 . 1) (-2.0 . 1) (-4.0 . 1))
<c e g>~ <c e g> |
```

See also

Music Glossary: ⟨undefined⟩ [tie], page ⟨undefined⟩,

Snippets: Rhythms

Internals Reference: `LaissezVibrerTie` `LaissezVibrerTieColumn`

Example files:

Notation Reference: Section 1.2.3.5 [Automatic note splitting], page 48.

Snippets: Rhythms

Internals Reference: `Tie`.

Known issues and warnings

Switching staves when a tie is active will not produce a slanted tie.

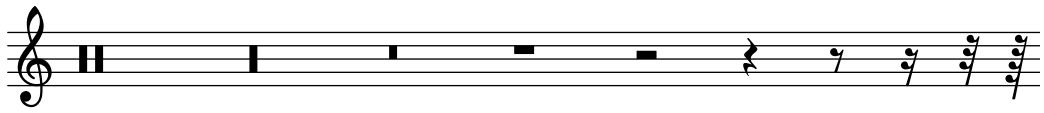
Changing clefs or octavations during a tie is not really well-defined. In these cases, a slur may be preferable.

1.2.2 Writing rests

1.2.2.1 Rests

Rests are entered like notes with the note name `r`:

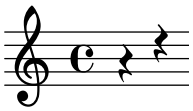
```
\new Staff {
  \time 16/1
  \override Staff.TimeSignature #'stencil = ##f
  r\maxima
  r\longa r\breve r1 r2
  r4 r8 r16 r32 r64
}
```



Whole measure rests, centered in middle of the measure, must be entered as multi-measure rests. They can be used for a single measure as well as many measures and are discussed in Section 1.2.2.3 [Full measure rests], page 39.

To explicitly specify a rest's vertical position, write a note followed by `\rest`. A rest will be placed in the position where the note would appear. This allows for precise manual formatting of polyphonic music, since the automatic rest collision formatter will leave these rests alone.

```
a4\rest d4\rest
```



See also

Notation Reference: Section 1.2.2.3 [Full measure rests], page 39.

Snippets: Rhythms .

Internals Reference: `Rest`.

Known issues and warnings

There is no fundamental limit to rest durations (both in terms of longest and shortest), but the number of glyphs is limited: there are rests from 128th to maxima (8 x whole).

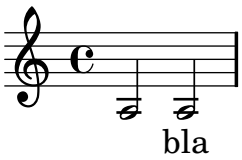
1.2.2.2 Skips

An invisible rest (also called a 'skip') can be entered like a note with note name `s` or with `\skip` *`duration`*

```
a4 a4 s4 a4 \skip 1 a4
```

The `s` syntax is only available in note mode and chord mode. In other situations, for example, when entering lyrics, one must use the `\skip` command:

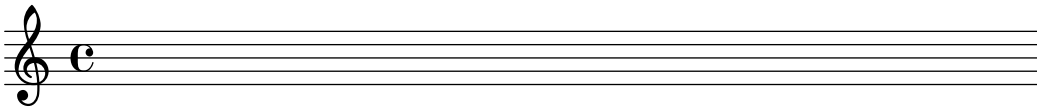
```
<<
  { a2 a2 }
  \new Lyrics \lyricmode { \skip 2 bla2 }
>>
```



The skip command is merely an empty musical placeholder. It does not produce any output, not even transparent output.

The `s` skip command does create `Staff` and `Voice` when necessary, similar to note and rest commands. For example, the following results in an empty staff.

```
{ s4 }
```



See also

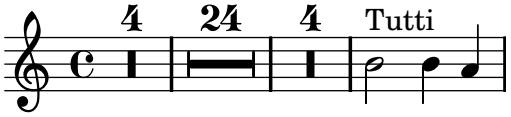
Snippets: Rhythms

Internals Reference: `SkipMusic`.

1.2.2.3 Full measure rests

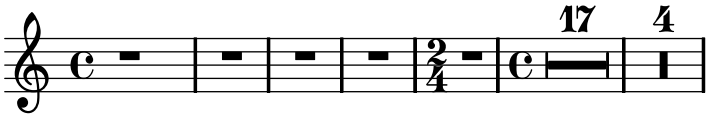
Rests for one or more full measures are entered using `R` followed by a duration (see Section 1.2.1.1 [Durations], page 30). The duration should correspond to an integral number of measures, otherwise a barcheck warning is printed. A ⟨undefined⟩ [multi-measure rest], page ⟨undefined⟩ is used principally to indicate that a part in a multi-part score should be silent:

```
\set Score.skipBars = ##t
R1*4
R1*24
R1*4
b2^"Tutti" b4 a4
```



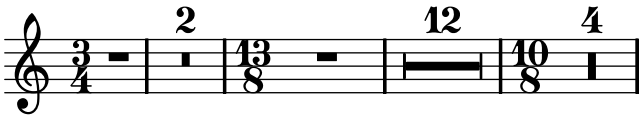
A multi-measure rest can be expanded in the printed score to show all the rest measures explicitly, or, as above, it can be condensed to a single measure containing a multi-measure rest symbol, with the number of measures of rest printed above the measure. This expansion is controlled by the property `Score.skipBars`. If this is set to true, empty measures will be condensed to a single measure.

```
\time 4/4 r1 | R1 | R1*2 |
\time 2/4 R2 |
\time 4/4
\set Score.skipBars = ##t
R1*17 | R1*4 |
```



The `1` in `R1` is similar to the duration notation used for notes and is the length of a measure in 2/2 or 4/4 time. The duration in a multi-measure rest must always be an integral number of measure-lengths, so in other time signatures augmentation dots or fractions must be used:

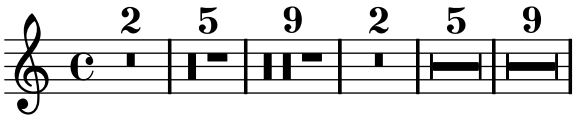
```
\set Score.skipBars = ##t
\time 3/4
R2. | R2.*2
\time 13/8
R1*13/8 | R1*13/8*12 |
\time 10/8
R4*5*4 |
```



An `R` spanning a single measure is printed as either a whole or breve rest, centered in the measure, regardless of the time signature.

If there are 10 or fewer measures of rest, LilyPond prints a series of longa and breve rests (called in German Kirchenpausen - 'church rests') within the staff and prints a simple line otherwise. This default number of 10 may be changed by overriding `MultiMeasureRest.expand-limit`.

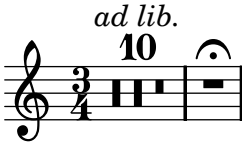
```
\set Score.skipBars = ##t
R1*2 | R1*5 | R1*9
\override MultiMeasureRest #'expand-limit = 3
R1*2 | R1*5 | R1*9
```



Note that unlike ordinary rests, the vertical position on the staff of the multi-measure rest symbol of either form cannot be changed.

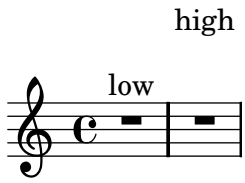
Text can be added to multi-measure rests by using the *note*-`markup` syntax described in Section 1.8.2 [Text markup], page 119. The variable `\fermataMarkup` is provided for adding fermatas.

```
\set Score.skipBars = ##t
\time 3/4
R2.*10^\markup { \italic "ad lib." }
R2.^\fermataMarkup
```



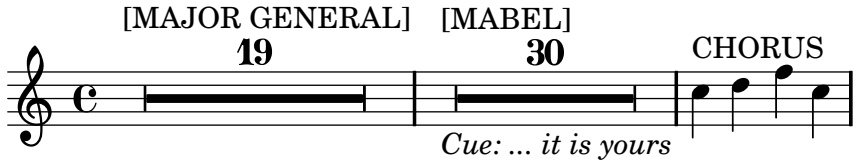
> **Note:** Text attached to a multi-measure rest is created by `MultiMeasureRestText`, not `TextScript`. Overrides must be directed to the correct object, or they will be ignored. See the following example.

```
\override TextScript #'padding = #5
R1^"low"
\override MultiMeasureRestText #'padding = #5
R1^"high"
```



Text attached to a multi-measure rest will be centered above or below it. Long text attached in this way does not cause the measure to expand, and may collide with text in adjacent measures. Long text is better attached to a zero-length skip note preceding the rest, preceded by `\textLengthOn` (turn off again with `\textLengthOff`), since this will cause the measure to expand to accommodate the length of the text:

```
\set Score.skipBars = ##t
\textLengthOn
s1*0^\markup {[MAJOR GENERAL]}
R1*19
s1*0^\markup {[MABEL] }
s1*0_\markup {\italic {Cue: ... it is yours}}
R1*30
\textLengthOff
c4^\markup {CHORUS} d f c
```



Text attached to a skip note in this way is left-aligned to the position where the note would be placed in the bar, and placed above the bar count numeral, but because the bar length is determined by the length of the text, the text will appear to be centered. If two (or more) texts are attached to skip notes in a bar the bar length is determined by the longer text, and the shorter text is then clearly left-aligned, as shown in the second bar above. If the shorter text of two marks is short enough to fit it will be placed alongside and to the left of the bar count numeral.

See also

Notation Reference: Section 1.2.1.1 [Durations], page 30, Section 1.8 [Text], page 113, Section 1.8.2 [Text markup], page 119, Section 1.8.1.2 [Text scripts], page 114.

Snippets: Rhythms

Internals Reference: `MultiMeasureRest`.

The layout object `MultiMeasureRestNumber` is for the default number, and `MultiMeasureRestText` for user specified texts.

Known issues and warnings

If an attempt is made to use fingerings (e.g., `R1*10-4`) to put numbers over multi-measure rests, the fingering numeral (4) may collide with the bar counter numeral (10).

There is no way to automatically condense multiple rests into a single multi-measure rest. Multi-measure rests do not take part in rest collisions.

Be careful when entering multi-measure rests followed by whole notes. The following will enter two notes lasting four measures each:

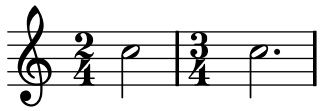
```
R1*4 cis cis
```

1.2.3 Displaying rhythms

1.2.3.1 Time signature

The time signature is set with the `\time` command:

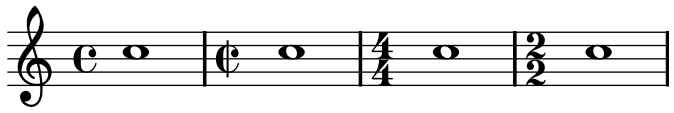
```
\time 2/4 c2 \time 3/4 c2.
```



Selected Snippets

The symbol that is printed can be customized with the `style` property. Setting it to `#'()` uses fraction style for 4/4 and 2/2 time,

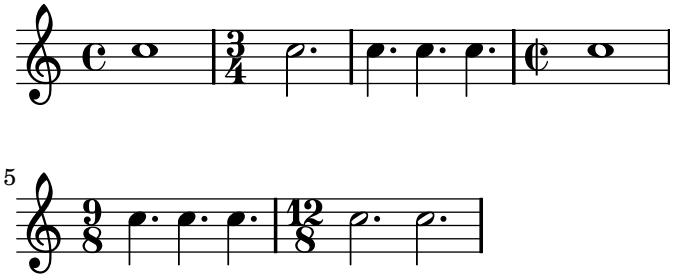
```
\time 4/4 c1
\time 2/2 c1
\override Staff.TimeSignature #'style = #'()
\time 4/4 c1
\time 2/2 c1
```



A time signature symbol is normally printed whenever the time signature changes. If this takes place at the end of a line a warning time signature sign is printed at the end of the line and again at the start of a new line. This default behaviour can be modified by setting the value of the `break-visibility` property. This takes three values which may be set to `#t` or `#f` to specify whether the corresponding time signature is visible or not. The order of the three values is `end of line visible`, `middle of line visible`, `beginning of line visible`.

```
% Do not print any time signatures at end of line
\override Staff.TimeSignature #'break-visibility = ##(#f #t #t)
\time 4/4 c1
```

```
\time 3/4 c2.
% Do not print the following 9/8 time signature
\once \override Staff.TimeSignature #'break-visibility = ##(#t #f #t)
\time 9/8 c4. c c
\time 2/2 c1
\break
\time 9/8 c4. c c
\time 12/8 c2. c2.
```

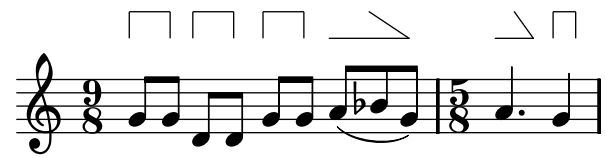


There are many more options for its layout. See Section 2.8.1.6 [Ancient time signatures], page 172, for more examples.

`\time` sets the properties `timeSignatureFraction`, `beatLength`, and `measureLength` in the `Timing` context, which is normally aliased to `Score`. The property `measureLength` determines where bar lines should be inserted, and how automatic beams should be generated. Changing the value of `timeSignatureFraction` also causes the symbol to be printed.

More options are available through the Scheme function `set-time-signature`, which takes three arguments: the number of beats, the beat length, and the internal grouping of beats in the measure. If the `Measure_grouping_engraver` is included, the function will also create `MeasureGrouping` signs. Such signs ease reading rhythmically complex modern music. In the following example, the 9/8 measure is subdivided in 2, 2, 2 and 3. This is passed to `set-time-signature` as the third argument `(2 2 2 3)`:

```
\score {
  \relative c'' {
    #(set-time-signature 9 8 '(2 2 2 3))
    g8[ g] d[ d] g[ g] a8[( bes g]) |
    #(set-time-signature 5 8 '(3 2))
    a4. g4
  }
  \layout {
    \context {
      \Staff
      \consists "Measure_grouping_engraver"
    }
  }
}
```



See also

Snippets: Rhythms

Internals Reference: `TimeSignature`, and `Timing_translator`.

Examples:

Known issues and warnings

Automatic beaming does not use the measure grouping specified with `set-time-signature`.

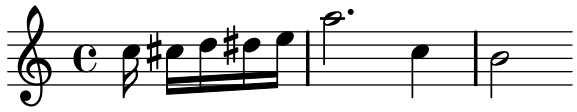
1.2.3.2 Upbeats

Partial or pickup measures, such as an anacrusis or upbeat, are entered using the `\partial` command, with the syntax

```
\partial duration
```

where `duration` is the rhythmic length to be added before the next measure:

```
\partial 16*5 c16 cis d dis e | a2. c,4 | b2
```



Internally, this is translated into

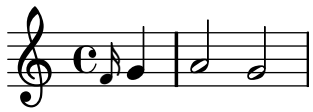
```
\set Timing.measurePosition = -length of duration
```

The property `measurePosition` contains a rational number indicating how much of the measure has passed at this point. Note that this is a negative number; `\partial 4` is internally translated to mean "there is a quarter note left in the bar."

Known issues and warnings

This command does not take into account grace notes at the start of the music. When a piece starts with grace notes in the pickup, then the `\partial` should follow the grace notes:

```
\grace f16
\partial 4
g4
a2 g2
```



`\partial` is only intended to be used at the beginning of a piece. If you use it after the beginning, some odd warnings may occur.

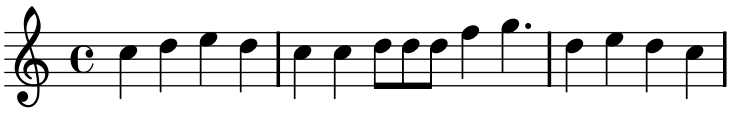
See also

Snippets: Rhythms

1.2.3.3 Unmetered music

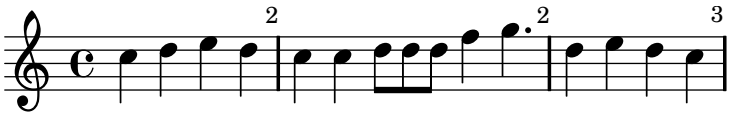
Bar lines and bar numbers are calculated automatically. For unmetered music (cadenzas, for example), this is not desirable. To turn off automatic bar lines and bar numbers, use the commands `\cadenzaOn` and `\cadenzaOff`.

```
c4 d e d
\cadenzaOn
c4 c d8 d d f4 g4.
\cadenzaOff
\bar "|"
d4 e d c
```



Bar numbering is resumed at the end of the cadenza as if the cadenza were not there:

```
\override Score.BarNumber #'break-visibility = ##( #t #t #t )
c4 d e d
\cadenzaOn
c4 c d8 d d f4 g4.
\cadenzaOff
\bar "|"
d4 e d c
```



Known issues and warnings

LilyPond will only insert line breaks and page breaks at a bar line. Unless the unmetered music ends before the end of the staff line, you will need to insert invisible bar lines with

```
\bar ""
```

to indicate where breaks can occur.

See also

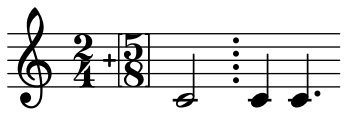
Snippets: Rhythms

1.2.3.4 Polymetric notation

Music Glossary: ⟨undefined⟩ [polymetric], page ⟨undefined⟩ ⟨undefined⟩ [polymetric time signature], page ⟨undefined⟩ ⟨undefined⟩ [meter], page ⟨undefined⟩

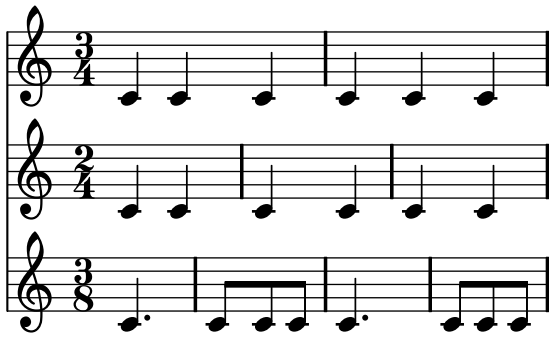
Double time signatures are not supported explicitly, but they can be faked. In the next example, the markup for the time signature is created with a markup text. This markup text is inserted in the `TimeSignature` grob. See also

```
% create 2/4 + 5/8
tsMarkup = \markup {
  \override #'(baseline-skip . 2) \number {
    \column { "2" "4" }
    \vcenter "+"
    \bracket \column { "5" "8" }
  }
}

{
  \override Staff.TimeSignature #'stencil =
    #ly:text-interface::print
  \override Staff.TimeSignature #'text = #tsMarkup
  \time 3/2
  c'2 \bar ":" c'4 c'4.
}
```



Each staff can also have its own time signature. This is done by moving the `Timing_translator` to the `Staff` context.

```
\layout {
  \context {
    \Score
    \remove "Timing_translator"
    \remove "Default_bar_line_engraver"
  }
  \context {
    \Staff
    \consists "Timing_translator"
    \consists "Default_bar_line_engraver"
  }
}
%Now, each staff has its own time signature.

\relative c' <<
  \new Staff {
    \time 3/4
    c4 c c | c c c |
  }
  \new Staff {
    \time 2/4
    c4 c | c c | c c
  }
  \new Staff {
    \time 3/8
    c4. c8 c c c4. c8 c c
  }
>>
```



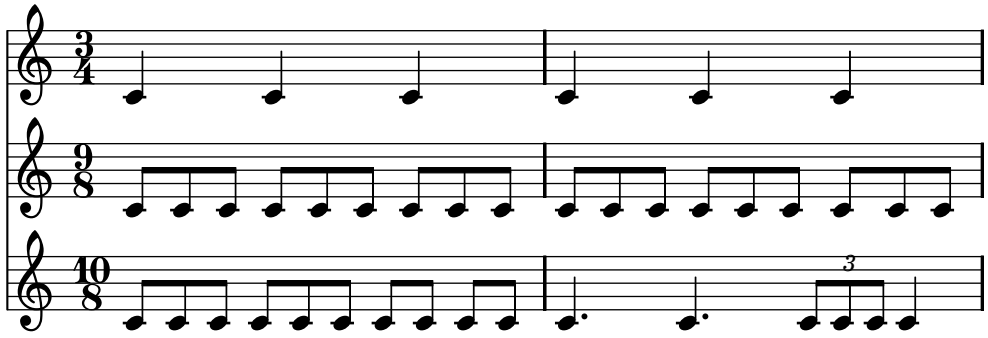
A different form of polymetric notation is where note lengths have different values across staves, but the measures are all the same length.

This notation can be created by setting a common time signature for each staff but replacing it manually using `timeSignatureFraction` to the desired fraction. Then the printed durations in each staff are scaled to the common time signature. The latter is done with `\compressMusic`, which is used in a similar way to `\times`, but does not create a tuplet bracket. The syntax is

```
\compressMusic
#'(numerator . denominator) musicexpr
```

In this example, music with the time signatures of 3/4, 9/8, and 10/8 are used in parallel. In the second staff, shown durations are multiplied by 2/3, so that 2/3 * 9/8 = 3/4, and in the third staff, shown durations are multiplied by 3/5, so that 3/5 * 10/8 = 3/4.

```
\relative c' { <<
  \new Staff {
    \time 3/4
    c4 c c | c c c |
  }
  \new Staff {
    \time 3/4
    \set Staff.timeSignatureFraction = #'(9 . 8)
    \compressMusic #'(2 . 3)
      \repeat unfold 6 { c8[ c c] }
  }
  \new Staff {
    \time 3/4
    \set Staff.timeSignatureFraction = #'(10 . 8)
    \compressMusic #'(3 . 5) {
      \repeat unfold 2 { c8[ c c] }
      \repeat unfold 2 { c8[ c] }
      | c4. c4. \times 2/3 { c8 c c } c4
    }
  }
>> }
```



Known issues and warnings

When using different time signatures in parallel, the spacing is aligned vertically, but bar lines distort the regular spacing.

See also

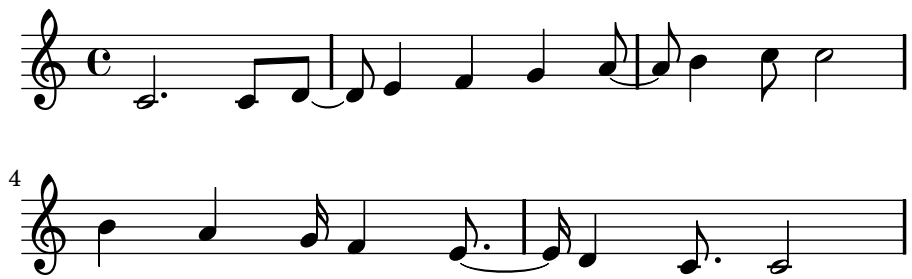
Snippets: Rhythms ,

Internals Reference: `TimeSignature`, `Timing-translator`, `Staff`.

1.2.3.5 Automatic note splitting

Long notes which overrun bar lines can be converted automatically to tied notes. This is done by replacing the `Note_heads_engraver` by the `Completion_heads_engraver`. In the following examples, notes crossing the bar line are split and tied.

```
\new Voice \with {
  \remove "Note_heads_engraver"
  \consists "Completion_heads_engraver"
} {
  c2. c8 d4 e f g a b c8 c2 b4 a g16 f4 e d c8. c2
}
```



This engraver splits all running notes at the bar line, and inserts ties. One of its uses is to debug complex scores: if the measures are not entirely filled, then the ties exactly show how much each measure is off.

If you want to allow line breaking on the bar lines where `Completion_heads_engraver` splits notes, you must also remove `Forbid_line_break_engraver`.

Known issues and warnings

Not all durations (especially those containing tuplets) can be represented exactly with normal notes and dots, but the engraver will not insert tuplets.

`Completion_heads_engraver` only affects notes; it does not split rests.

See also

Snippets: Rhythms

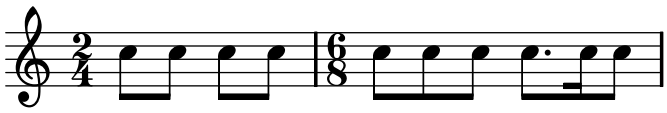
Internals Reference: `Note_heads_engraver`, `Completion_heads_engraver`, `Forbid_line_break_engraver`.

1.2.4 Beams

1.2.4.1 Automatic beams

By default, beams are inserted automatically:

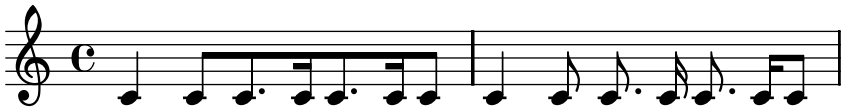
```
\time 2/4 c8 c c c
\time 6/8 c c c c8. c16 c8
```



If these automatic decisions are not satisfactory, beaming can be entered explicitly; see Section 1.2.4.3 [Manual beams], page 54. It is also possible to define beaming patterns that differ from the defaults; see Section 1.2.4.2 [Setting automatic beam behavior], page 51. The default beaming rules are defined in '`scm/auto-beam.scm`'.

Automatic beaming may be turned off and on with `\autoBeamOff` and `\autoBeamOn` commands:

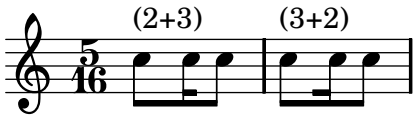
```
c4 c8 c8. c16 c8. c16 c8
\autoBeamOff
c4 c8 c8. c16 c8.
\autoBeamOn
c16 c8
```



Selected Snippets

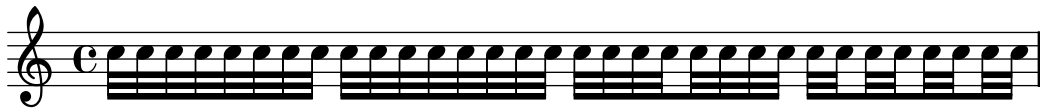
Beaming patterns may be altered with the `beatGrouping` property,

```
\time 5/16
\set beatGrouping = #'(2 3)
c8[^"(2+3)" c16 c8]
\set beatGrouping = #'(3 2)
c8[^"(3+2)" c16 c8]
```



The beams of consecutive 16th (or shorter) notes are, by default, not sub-divided. That is, the three (or more) beams stretch unbroken over entire groups of notes. This behaviour can be modified to sub-divide the beams into sub-groups by setting the property `subdivideBeams`. When set, multiple beams will be sub-divided at intervals defined by the current value of `beatLength` by reducing the multiple beams to just one beam between the sub-groups. Note that `beatLength` lives in the `Score` context and defaults to a quarter note. It must be set to a fraction giving the duration of the beam sub-group using the `make-moment` function, as shown here:

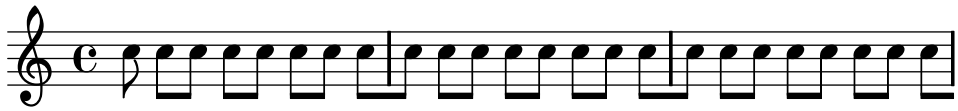
```
c32[ c c c c c c c]
\set subdivideBeams = ##t
c32[ c c c c c c c]
% Set beam sub-group length to an eighth note
\set Score.beatLength = #(ly:make-moment 1 8)
c32[ c c c c c c c]
% Set beam sub-group length to a sixteenth note
\set Score.beatLength = #(ly:make-moment 1 16)
c32[ c c c c c c c]
```

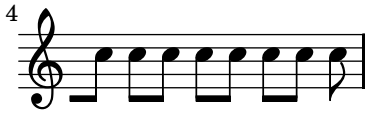


For more information about `make-moment`, see Section 1.2.6.3 [Time administration], page 67.

Line breaks are normally forbidden when beams cross bar lines. This behavior can be changed by setting the `breakable` property: `\override Beam #'breakable = ##t`.

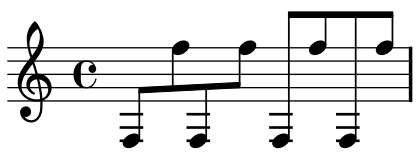
```
\override Beam #'breakable = ##t
c8 \repeat unfold 15 {c[ c] } c
```





Kneed beams are inserted automatically when a large gap is detected between the note heads. This behavior can be tuned through the `auto-knee-gap` property. A kneed beam is drawn if the gap is larger than the value of `auto-knee-gap` plus the width of the beam object (which depends on the duration of the notes and the slope of the beam). By default `auto-knee-gap` is set to 5.5 staff spaces.

```
f8 f''8 f8 f''8
\override Beam #'auto-knee-gap = #6
f8 f''8 f8 f''8
```



See also

Notation Reference: Section 1.2.4.3 [Manual beams], page 54, Section 1.2.4.2 [Setting automatic beam behavior], page 51.

Snippets: Rhythms

Internals Reference: `Beam`.

Known issues and warnings

Automatically kneed cross-staff beams cannot be used together with hidden staves. See Section 1.6.1.3 [Hiding staves], page 97.

Beams can collide with note heads and accidentals in other voices

1.2.4.2 Setting automatic beam behavior

In normal time signatures, automatic beams can start on any note but can end in only a few positions within the measure: beams can end on a beat, or at durations specified by the properties in `autoBeamSettings`. The properties in `autoBeamSettings` consist of a list of rules for where beams can begin and end. The default `autoBeamSettings` rules are defined in '`scm/auto-beam.scm`'.

In order to add a rule to the list, use

```
#(override-auto-beam-setting '(be p q n m) a b [context])
```

- `be` is either `begin` or `end`.
- `p/q` is the duration of the note for which you want to add a rule. A beam is considered to have the duration of its shortest note. Set `p` and `q` to `'*'` to have this apply to any beam.
- `n/m` is the time signature to which this rule should apply. Set `n` and `m` to `'*'` to have this apply in any time signature.
- `a/b` is the position in the bar at which the beam should begin/end.
- `context` is optional, and it specifies the context at which the change should be made. The default is `'Voice`.

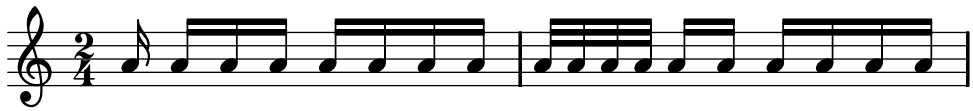
  `#(score-override-auto-beam-setting '(A B C D) E F)` is equivalent to `#(override-auto-beam-setting '(A B C D) E F 'Score)`.

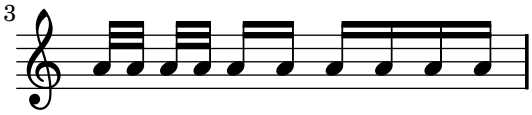
For example, if automatic beams should always end on the first quarter note, use

```
#(override-auto-beam-setting '(end * * * *) 1 4)
```

You can force the beam settings to only take effect on beams whose shortest note is a certain duration

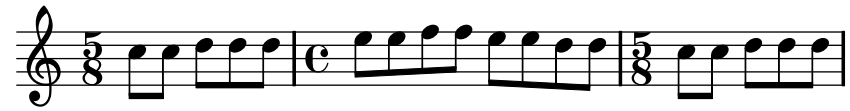
```
\time 2/4
#(override-auto-beam-setting '(end 1 16 * *) 1 16)
a16 a a a a a a a |
a32 a a a a16 a a a a a |
#(override-auto-beam-setting '(end 1 32 * *) 1 16)
a32 a a a a16 a a a a a |
```





You can force the beam settings to only take effect in certain time signatures

```
\time 5/8
#(override-auto-beam-setting '(end * * 5 8) 2 8)
c8 c d d d
\time 4/4
e8 e f f e e d d
\time 5/8
c8 c d d d
```

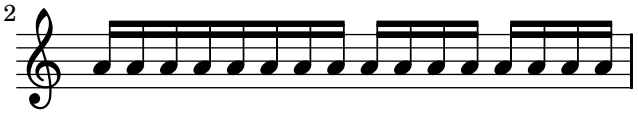


You can also remove a previously set beam-ending rule by using

```
#(revert-auto-beam-setting '(be p q n m) a b [context])
```

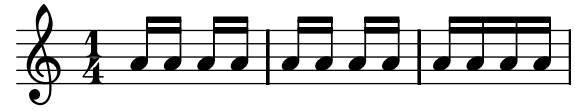
be, p, q, n, m, a, b and context are the same as above. Note that the default rules are specified in '`scm/auto-beam.scm`', so you can revert rules that you did not explicitly create.

```
\time 4/4
a16 a a a a a a a a a a a a a a a a
#(revert-auto-beam-setting '(end 1 16 4 4) 1 4)
a16 a a a a a a a a a a a a a a a a
```



The rule in a revert-auto-beam-setting statement must exactly match the original rule. That is, no wildcard expansion is taken into account.

```
\time 1/4
#(override-auto-beam-setting '(end 1 16 1 4) 1 8)
a16 a a a
#(revert-auto-beam-setting '(end 1 16 * *) 1 8) % this won't revert it!
a a a a
#(revert-auto-beam-setting '(end 1 16 1 4) 1 8) % this will
a a a a
```



If automatic beams should end on every quarter in 5/4 time, specify all endings

```
#(override-auto-beam-setting '(end * * * *) 1 4 'Staff)
#(override-auto-beam-setting '(end * * * *) 1 2 'Staff)
#(override-auto-beam-setting '(end * * * *) 3 4 'Staff)
#(override-auto-beam-setting '(end * * * *) 5 4 'Staff)
...
```

The same syntax can be used to specify beam starting points. In this example, automatic beams can only end on a dotted quarter note

```
#(override-auto-beam-setting '(end * * * *) 3 8)
#(override-auto-beam-setting '(end * * * *) 1 2)
#(override-auto-beam-setting '(end * * * *) 7 8)
```

In 4/4 time signature, this means that automatic beams could end only on 3/8 and on the fourth beat of the measure (after 3/4, that is 2 times 3/8, has passed within the measure).

If any unexpected beam behaviour occurs, check the default automatic beam settings in '`scm/auto-beam.scm`' for possible interference, because the beam endings defined there will still apply on top of your own overrides. Any unwanted endings in the default vales must be reverted for your time signature(s).

For example, to typeset `(3 4 3 2)`-beam endings in 12/8, begin with

```
%%% revert default values in scm/auto-beam.scm regarding 12/8 time
#(revert-auto-beam-setting '(end * * 12 8) 3 8)
#(revert-auto-beam-setting '(end * * 12 8) 3 4)
#(revert-auto-beam-setting '(end * * 12 8) 9 8)

%%% your new values
#(override-auto-beam-setting '(end 1 8 12 8) 3 8)
#(override-auto-beam-setting '(end 1 8 12 8) 7 8)
#(override-auto-beam-setting '(end 1 8 12 8) 10 8)
```

If beams are used to indicate melismata in songs, then automatic beaming should be switched off with `\autoBeamOff`.

Predefined commands

`\autoBeamOff`, `\autoBeamOn`.

Known issues and warnings

If a score ends while an automatic beam has not been ended and is still accepting notes, this last beam will not be typeset at all. The same holds for polyphonic voices, entered with `<< ... \\ ... >>`. If a polyphonic voice ends while an automatic beam is still accepting notes, it is not typeset.

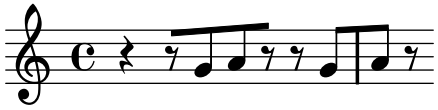
See also

Snippets: Rhythms

1.2.4.3 Manual beams

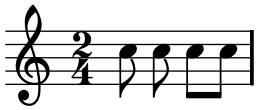
In some cases it may be necessary to override the automatic beaming algorithm. For example, the autobeamer will not put beams over rests or bar lines, and in choral scores the beaming is often set to follow the meter of the lyrics rather than the notes. Such beams can be specified manually by marking the begin and end point with `[` and `]`

```
{
  r4 r8[ g' a r8] r8 g[ | a] r8
}
```



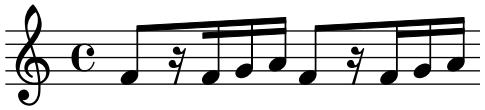
Individual notes may be marked with `\noBeam` to prevent them from being beamed:

```
\time 2/4 c8 c\noBeam c c
```



Even more strict manual control with the beams can be achieved by setting the properties `stemLeftBeamCount` and `stemRightBeamCount`. They specify the number of beams to draw on the left and right side, respectively, of the next note. If either property is set, its value will be used only once, and then it is erased. In this example, the last `f` is printed with only one beam on the left side, i.e. the eigth-note beam of the group as a whole.

```
{
  f8[ r16 f g a]
  f8[ r16
  \set stemLeftBeamCount = #1
  f g a]
}
```



Selected Snippets

1.2.4.4 Feathered beams

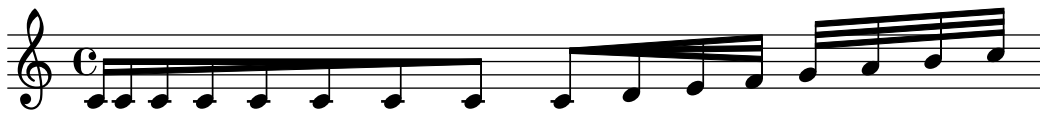
Feathered beams are used to indicate that a small group of notes should be played at an increasing (or decreasing) tempo, without changing the overall tempo of the piece. The extent of the feathered beam must be indicated manually using `[` and `]`, and the beam feathering is turned on by specifying a direction to the Beam property `grow-direction`.

If the placement of the notes and the sound in the MIDI output is to reflect the ritardando or accelerando indicated by the feathered beam the notes must be grouped as a music expression delimited by braces and preceded by a `featheredDurations` command which specifies the ratio between the durations of the first and last notes in the group.

The square brackets show the extent of the beam and the braces show which notes are to have their durations modified. Normally these would delimit the same group of notes, but this is not required: the two commands are independent.

In the following example the eight 16th notes occupy exactly the same time as a half note, but the first note is one half as long as the last one, with the intermediate notes gradually lengthening. The first four 32nd notes gradually speed up, while the last four 32nd notes are at a constant tempo.

```
\override Beam #'grow-direction = #LEFT
\featherDurations #(ly:make-moment 2 1)
{ c16[ c c c c c c c] }
\override Beam #'grow-direction = #RIGHT
\featherDurations #(ly:make-moment 2 3)
{ c32[ d e f] }
% revert to non-feathered beams
\override Beam #'grow-direction = #'()
{ g32[ a b c] }
```



The spacing in the printed output represents the note durations only approximately, but the midi output is exact.

Known issues and warnings

The `\featherDurations` command only works with very short music snippets, and when numbers in the fraction are small.

See also

Snippets: Rhythms

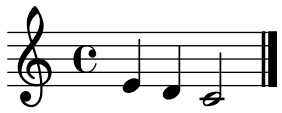
1.2.5 Bars

1.2.5.1 Bar lines

Bar lines delimit measures, and are also used to indicate repeats. Normally, simple bar lines are automatically inserted into the printed output at places based on the current time signature.

The simple bar lines inserted automatically can be changed to other types with the `\bar` command. For example, a closing double bar line is usually placed at the end of a piece:

```
e4 d c2 \bar "|."
```



Note: An incorrect duration can lead to poorly formatted music.

It is not invalid if the final note in a bar does not end on the automatically entered bar line: the note is assumed to carry over into the next bar. But if a long sequence of such carry-over bars appears the music can appear compressed or even flowing off the page. This is because automatic line breaks happen only at the end of complete bars, i.e. where the end of a note coincides with the end of a bar.

Line breaks are also permitted at manually inserted bar lines even within incomplete bars. To allow a line break without printing a bar line, use

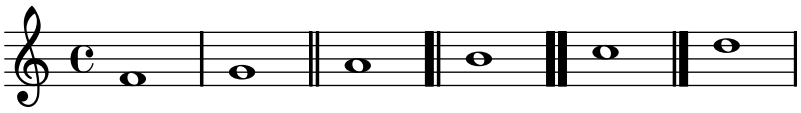
```
\bar ""
```

This will insert an invisible bar line and allow (but not force) a line break to occur at this point. The bar number counter is not increased. To force a line break see Section 5.4.1 [Line breaking], page 215.

This and other special bar lines may be inserted manually at any point. When they coincide with the end of a bar they replace the simple bar line which would have been inserted there automatically. When they do not coincide with the end of a bar the specified bar line is inserted at that point in the printed output. Such insertions do not affect the calculation and placement of subsequent automatic bar lines.

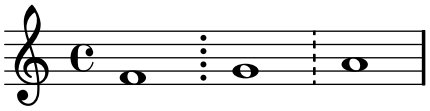
The simple bar line and four types of double bar line are available for manual insertion:

```
f1 \bar "|" g \bar "||" a \bar ".|" b \bar ".|." c \bar "|." d
```



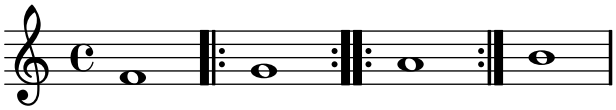
together with dotted and dashed bar lines:

```
f1 \bar ":" g \bar "dashed" a
```



and three types of repeat bar line:

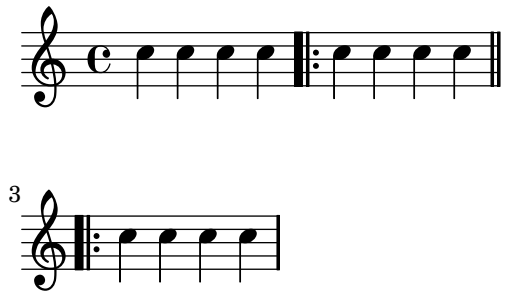
```
f1 \bar "|:" g \bar ":|:" a \bar ":|" b
```



Although the bar line types signifying repeats may be inserted manually they do not in themselves cause LilyPond to recognise a repeated section. Such repeated sections are better entered using the various repeat commands (see Section 1.4 [Repeats], page 79), which automatically print the appropriate bar lines.

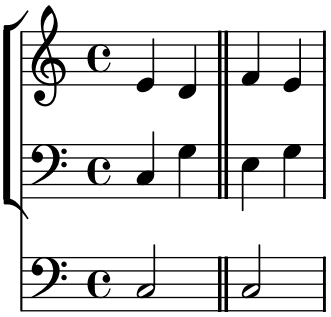
In addition, you can specify `"||:"`, which is equivalent to `"|:"` except at line breaks, where it gives a double bar line at the end of the line and a start repeat at the beginning of the next line.

```
\override Score.RehearsalMark #'padding = #3
c c c c
\bar "||:"
c c c c \break
\bar "||:"
c c c c
```



In scores with many staves, a `\bar` command in one staff is automatically applied to all staves. The resulting bar lines are connected between different staves of a `StaffGroup`, `PianoStaff`, or `GrandStaff`.

```
<<
  \new StaffGroup <<
    \new Staff {
      e'4 d'
      \bar "||"
      f' e'
    }
    \new Staff { \clef bass c4 g e g }
  >>
  \new Staff { \clef bass c2 c2 }
>>
```



Selected Snippets

The command `\bar` *bartype* is a shortcut for `\set Timing.whichBar =` *bartype*. A bar line is created whenever the `whichBar` property is set.

The default bar type used for automatically inserted bar lines is `"|"`. This may be changed at any time with `\set Timing.defaultBarType =` *bartype*.

See also

Notation Reference: Section 5.4.1 [Line breaking], page 215, Section 1.4 [Repeats], page 79, Section 1.6.1.1 [System start delimiters], page 93.

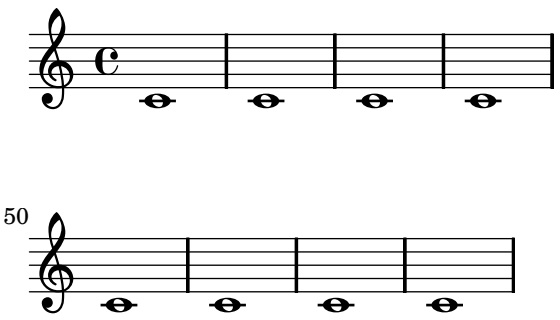
Snippets: Rhythms

Internals Reference: `BarLine` (created at `Staff` level), `SpanBar` (across staves), `Timing_translator` (for Timing properties).

1.2.5.2 Bar numbers

Bar numbers are typeset by default at the start of every line except the first line. The number itself is stored in the `currentBarNumber` property, which is normally updated automatically for every measure. It may also be set manually:

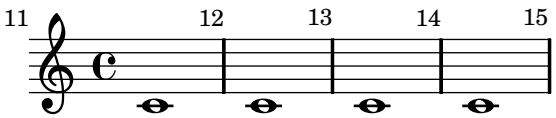
```
c1 c c c
\break
\set Score.currentBarNumber = #50
c1 c c c
```

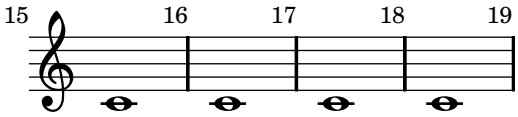


Selected Snippets

Bar numbers can be typeset at regular intervals instead of just at the beginning of every line. To do this the default behaviour must be overridden to permit bar numbers to be printed at places other than the start of a line. This is controlled by the `break-visibility` property of `BarNumber`. This takes three values which may be set to `#t` or `#f` to specify whether the corresponding bar number is visible or not. The order of the three values is `end of line visible`, `middle of line visible`, `beginning of line visible`. In the following example bar numbers are printed at all possible places:

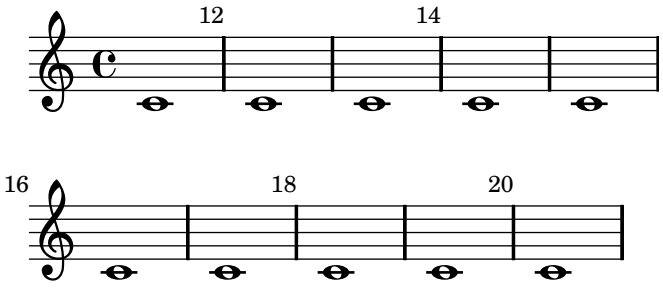
```
\override Score.BarNumber #'break-visibility = ##(#t #t #t)
\set Score.currentBarNumber = #11
\bar ""  % Permit first bar number to be printed
c1 c c c
\break
c c c c
```





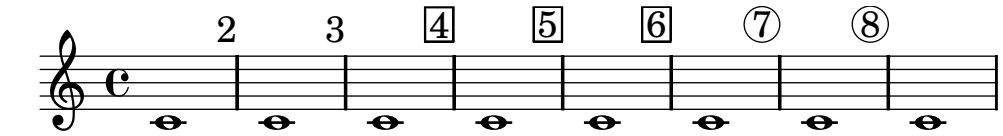
and here the bar numbers are printed every two bars except at the end of the line:

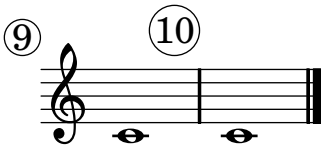
```
\override Score.BarNumber #'break-visibility = ##(#f #t #t)
\set Score.currentBarNumber = #11
\bar ""  % Permit first bar number to be printed
% Print a bar number every 2nd bar
\set Score.barNumberVisibility = #(every-nth-bar-number-visible 2)
c1 c c c c
\break
c c c c c
```



The size of the bar number may be changed. This is illustrated in the following example, which also shows how to enclose bar numbers in boxes and circles, and shows an alternative way of specifying `#(#f #t #t)` for `break-visibility`.

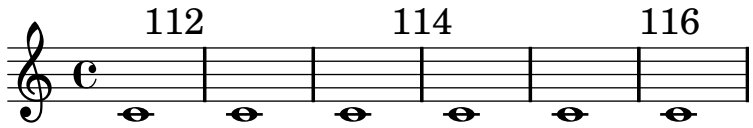
```
% Prevent bar numbers at the end of a line and permit them elsewhere
\override Score.BarNumber #'break-visibility
  = #end-of-line-invisible

% Increase the size of the bar number by 2
\override Score.BarNumber #'font-size = #2
\repeat unfold 3 { c1 } \bar "|"

% Draw a box round the following bar number(s)
\override Score.BarNumber  #'stencil
  = #(make-stencil-boxer 0.1 0.25 ly:text-interface::print)
\repeat unfold 3 { c1 } \bar "|"

% Draw a circle round the following bar number(s)
\override Score.BarNumber  #'stencil
  = #(make-stencil-circler 0.1 0.25 ly:text-interface::print)
\repeat unfold 4 { c1 } \bar "|."
```





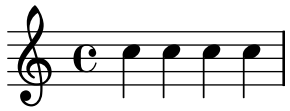
Bar numbers by default are left-aligned to their parent object. This is usually the left edge of a line or, if numbers are printed within a line, the left bar line of the bar. The numbers may also be positioned directly on the bar line or right-aligned to the bar line:

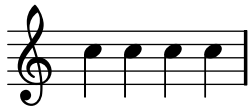
```
\set Score.currentBarNumber = #111
\override Score.BarNumber #'break-visibility = ##(#t #t #t)
% Increase the size of the bar number by 2
\override Score.BarNumber #'font-size = #2
% Print a bar number every 2nd bar
\set Score.barNumberVisibility = #(every-nth-bar-number-visible 2)
c1 c1
% Centre-align bar numbers
\override Score.BarNumber #'self-alignment-X = #0
c1 c1
% Right-align bar numbers
\override Score.BarNumber #'self-alignment-X = #-1
c1 c1
```



Bar numbers can be removed entirely by removing the Bar number engraver from the score context.

```
\layout {
  \context {
    \Score
    \remove "Bar_number_engraver"
  }
}
\relative c''{
  c4 c c c \break
  c4 c c c
}
```





See also

Snippets: Rhythms

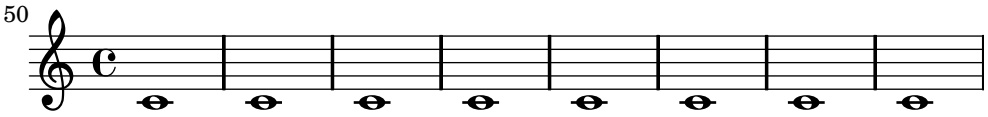
Internals Reference: `BarNumber`.

Known issues and warnings

Bar numbers may collide with the top of the `StaffGroup` bracket, if there is one. To solve this, the `padding` property of `BarNumber` can be used to position the number correctly.

Bar numbers may only be printed at bar lines; to print a bar number at the beginning of a piece, an empty bar line must be inserted there, and a value other than `1` must be placed in `currentBarNumber`:

```
\set Score.currentBarNumber = #50
\bar ""
c1 c c c
c1 c c c
\break
```



1.2.5.3 Bar and bar number checks

Bar checks help detect errors in the entered durations. A bar check may be entered using the bar symbol, `|`, at any place where a bar line is expected to fall. If bar check lines are encountered at other places, a list of warnings is printed in the log file, showing the line numbers and lines in which the bar checks failed. In the next example, the second bar check will signal an error.

```
\time 3/4 c2 e4 | g2 |
```

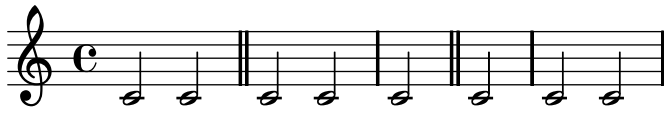
Bar checks can also be used in lyrics, for example

```
\lyricmode {
  \time 2/4
  Twin -- kle | Twin -- kle |
}
```

An incorrect duration can result in a completely garbled score, especially if the score is polyphonic, so a good place to start correcting input is by scanning for failed bar checks and incorrect durations.

It is also possible to redefine the action taken when a bar check or pipe symbol, `|`, is encountered in the input, so that it does something other than a bar check. This is done by assigning a music expression to `pipeSymbol`. In the following example `|` is set to insert a double bar line wherever it appears in the input, rather than checking for end of bar.

```
pipeSymbol = \bar "||"
{
  c'2 c'2 |
  c'2 c'2
  c'2 | c'2
  c'2 c'2
}
```



When copying large pieces of music, it can be helpful to check that the LilyPond bar number corresponds to the original that you are entering from. This can be checked with `\barNumberCheck`, for example,

`\barNumberCheck #123`
will print a warning if the `currentBarNumber` is not 123 when it is processed.

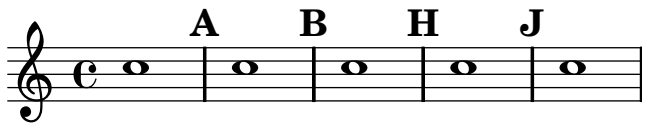
See also

Snippets: Rhythms

1.2.5.4 Rehearsal marks

To print a rehearsal mark, use the `\mark` command

```
c1 \mark \default
c1 \mark \default
c1 \mark #8
c1 \mark \default
c1 \mark \default
```



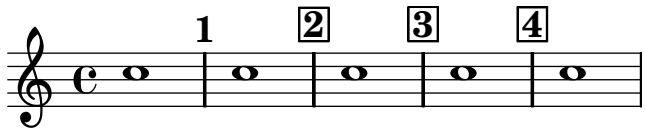
The letter 'I' is skipped in accordance with engraving traditions. If you wish to include the letter 'I', then use

```
\set Score.markFormatter = #format-mark-alphabet
```

The mark is incremented automatically if you use `\mark \default`, but you can also use an integer argument to set the mark manually. The value to use is stored in the property `rehearsalMark`.

The style is defined by the property `markFormatter`. It is a function taking the current mark (an integer) and the current context as argument. It should return a markup object. In the following example, `markFormatter` is set to a pre-defined procedure. After a few measures, it is set to a procedure that produces a boxed number.

```
\set Score.markFormatter = #format-mark-numbers
c1 \mark \default
c1 \mark \default
\set Score.markFormatter = #format-mark-box-numbers
c1 \mark \default
c1 \mark \default
c1
```



The file 'scm/translation-functions.scm' contains the definitions of `format-mark-numbers` (the default format), `format-mark-box-numbers`, `format-mark-letters` and `format-mark-box-letters`. These can be used as inspiration for other formatting functions.

You may use `format-mark-barnumbers`, `format-mark-box-barnumbers`, and `format-mark-circle-barnumbers` to get bar numbers instead of incremented numbers or letters.

Other styles of rehearsal mark can be specified manually

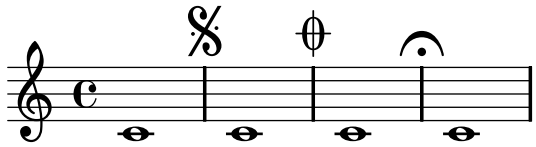
```
\mark "A1"
```

`Score.markFormatter` does not affect marks specified in this manner. However, it is possible to apply a `\markup` to the string.

```
\mark \markup{ \box A1 }
```

Music glyphs (such as the segno sign) may be printed inside a `\mark`

```
c1 \mark \markup { \musicglyph #"scripts.segno" }
c1 \mark \markup { \musicglyph #"scripts.coda" }
c1 \mark \markup { \musicglyph #"scripts.ufermata" }
c1
```



See Section B.4 [The Feta font], page 290, for a list of symbols which may be printed with `\musicglyph`.

For common tweaks to the positioning of rehearsal marks, see Section 1.8.1.4 [Text marks], page 116.

See also

Snippets: Rhythms

This manual: Section B.4 [The Feta font], page 290, Section 1.8.1.4 [Text marks], page 116.

Internals Reference: `RehearsalMark`.

Init files: '`scm/translation-functions.scm`' contains the definition of `format-mark-numbers` and `format-mark-letters`. They can be used as inspiration for other formatting functions.

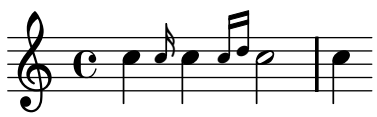
Examples:

1.2.6 Special rhythmic concerns

1.2.6.1 Grace notes

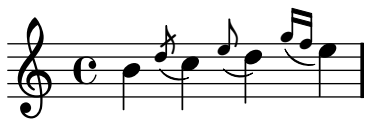
Grace notes are ornaments that are written out. They are made with the `\grace` command. By prefixing this keyword to a music expression, a new one is formed, which will be printed in a smaller font and takes up no logical time in a measure.

```
c4 \grace c16 c4
\grace { c16[ d16] } c2 c4
```



Two special forms of the `\grace` command exist. An *acciaccatura*, which should be played as very short, is denoted by a slurred small note with a slashed stem. The *appoggiatura*, a grace note that takes a fixed fraction of the main note, is denoted as a slurred note in small print without a slash. They are entered with the commands `\acciaccatura` and `\appoggiatura`, as demonstrated in the following example:

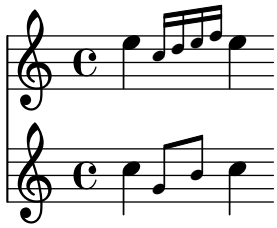
```
b4 \acciaccatura d8 c4
\appoggiatura e8 d4
\acciaccatura { g16[ f] } e4
```



`\acciaccatura` and `\appoggiatura` start a slur, `\grace` does not.

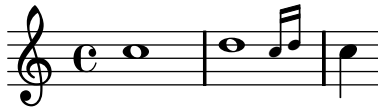
The placement of grace notes is synchronized between different staves. In the following example, there are two sixteenth grace notes for every eighth grace note

```
<< \new Staff { e4 \grace { c16[ d e f] } e4 }
   \new Staff { c4 \grace { g8[ b] } c4 } >>
```



If you want to end a note with a grace, use the `\afterGrace` command. It takes two arguments: the main note, and the grace notes following the main note.

```
c1 \afterGrace d1 { c16[ d] } c4
```



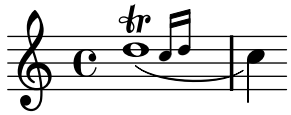
This will put the grace notes after a 'space' lasting 3/4 of the length of the main note. The fraction 3/4 can be changed by setting `afterGraceFraction`, ie.

```
#(define afterGraceFraction (cons 7 8))
```

will put the grace note at 7/8 of the main note.

The same effect can be achieved manually by doing

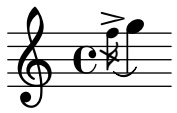
```
\new Voice {
  << { d1^\trill_( }
     { s2 \grace { c16[ d] } } >>
  c4)
}
```



By adjusting the duration of the skip note (here it is a half-note), the space between the main-note and the grace may be adjusted.

A `\grace` music expression will introduce special typesetting settings, for example, to produce smaller type, and set directions. Hence, when introducing layout tweaks, they should be inside the grace expression, for example,

```
\new Voice {
  \acciaccatura {
    \stemDown
    f16->
    \stemNeutral
  }
  g4
}
```



The overrides should also be reverted inside the grace expression.

The layout of grace expressions can be changed throughout the music using the function `add-grace-property`. The following example undefines the `Stem` direction for this grace, so that stems do not always point up.

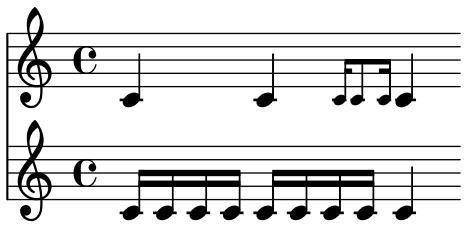
```
\new Staff {
  #(add-grace-property 'Voice 'Stem 'direction '())
  ...
}
```

Another option is to change the variables `startGraceMusic`, `stopGraceMusic`, `startAcciaccaturaMusic`, `stopAcciaccaturaMusic`, `startAppoggiaturaMusic`, `stopAppoggiaturaMusic`. The default values of these can be seen in the file '`ly/grace-init.ly`'. By redefining them other effects may be obtained.

The slash through the stem in *acciaccatura*s can be obtained in other situations by `\override Stem #'stroke-style = #"grace"`.

Selected Snippets

Grace notes may be forced to use align with regular notes in other staves by setting `strict-grace-spacing` to ##t:



See also

Music Glossary: ⟨undefined⟩ [grace notes], page ⟨undefined⟩, ⟨undefined⟩ [acciaccatura], page ⟨undefined⟩, ⟨undefined⟩ [appoggiatura], page ⟨undefined⟩

Snippets: Rhythms

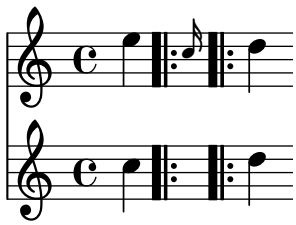
Internals Reference: `GraceMusic`.

Known issues and warnings

A multi-note beamed *acciaccatura* is printed without a slash, and looks exactly the same as a multi-note beamed *appoggiatura*.

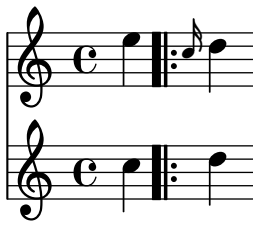
Grace note synchronization can also lead to surprises. Staff notation, such as key signatures, bar lines, etc., are also synchronized. Take care when you mix staves with grace notes and staves without, for example,

```
<< \new Staff { e4 \bar "|:" \grace c16 d4 }
   \new Staff { c4 \bar "|:" d4 } >>
```



This can be remedied by inserting grace skips of the corresponding durations in the other staves. For the above example

```
<< \new Staff { e4 \bar "|:" \grace c16 d4 }
   \new Staff { c4 \bar "|:" \grace s16 d4 } >>
```



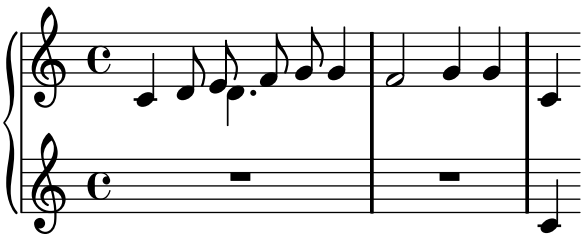
Grace sections should only be used within sequential music expressions. Nesting or juxtaposing grace sections is not supported, and might produce crashes or other errors.

1.2.6.2 Aligning to cadenzas

In an orchestral context, cadenzas present a special problem: when constructing a score that includes a cadenza, all other instruments should skip just as many notes as the length of the cadenza, otherwise they will start too soon or too late.

A solution to this problem is to use the functions `mmrest-of-length` and `skip-of-length`. These Scheme functions take a piece of music as argument, and generate a multi-rest or `\skip`, exactly as long as the piece. The use of `mmrest-of-length` is demonstrated in the following example.

```
cadenza = \relative c' {
  c4 d8 << { e f g } \\ { d4. } >>
  g4 f2 g4 g
}

\new GrandStaff <<
  \new Staff { \cadenza c'4 }
  \new Staff {
    #(ly:export (mmrest-of-length cadenza))
    c'4
  }
>>
```



See also

Snippets: Rhythms

1.2.6.3 Time administration

Time is administered by the `Timing_translator`, which by default is to be found in the `Score` context. An alias, `Timing`, is added to the context in which the `Timing_translator` is placed.

The following properties of `Timing` are used to keep track of timing within the score.

`currentBarNumber`
: The current measure number. For an example showing the use of this property see Section 1.2.5.2 [Bar numbers], page 58.

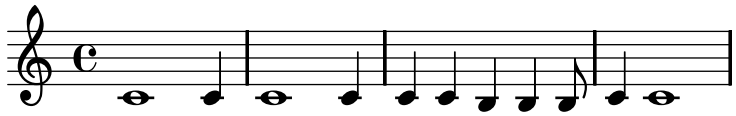
`measureLength`
: The length of the measures in the current time signature. For a 4/4 time this is 1, and for 6/8 it is 3/4. Its value determines when bar lines are inserted and how automatic beams should be generated.

`measurePosition`
: The point within the measure where we currently are. This quantity is reset by subtracting `measureLength` whenever `measureLength` is reached or exceeded. When that happens, `currentBarNumber` is incremented.

`timing`
: If set to true, the above variables are updated for every time step. When set to false, the engraver stays in the current measure indefinitely.

Timing can be changed by setting any of these variables explicitly. In the next example, the default 4/4 time signature is printed, but `measureLength` is set to 5/4. At 4/8 through the third measure, the `measurePosition` is advanced by 1/8 to 5/8, shortening that bar by 1/8. The next bar line then falls at 9/8 rather than 5/4.

```
\set Score.measureLength = #(ly:make-moment 5 4)
c1 c4
c1 c4
c4 c4
\set Score.measurePosition = #(ly:make-moment 5 8)
b4 b4 b8
c4 c1
```



As the example illustrates, `ly:make-moment n m` constructs a duration of n/m of a whole note. For example, `ly:make-moment 1 8` is an eighth note duration and `ly:make-moment 7 16` is the duration of seven sixteenths notes.

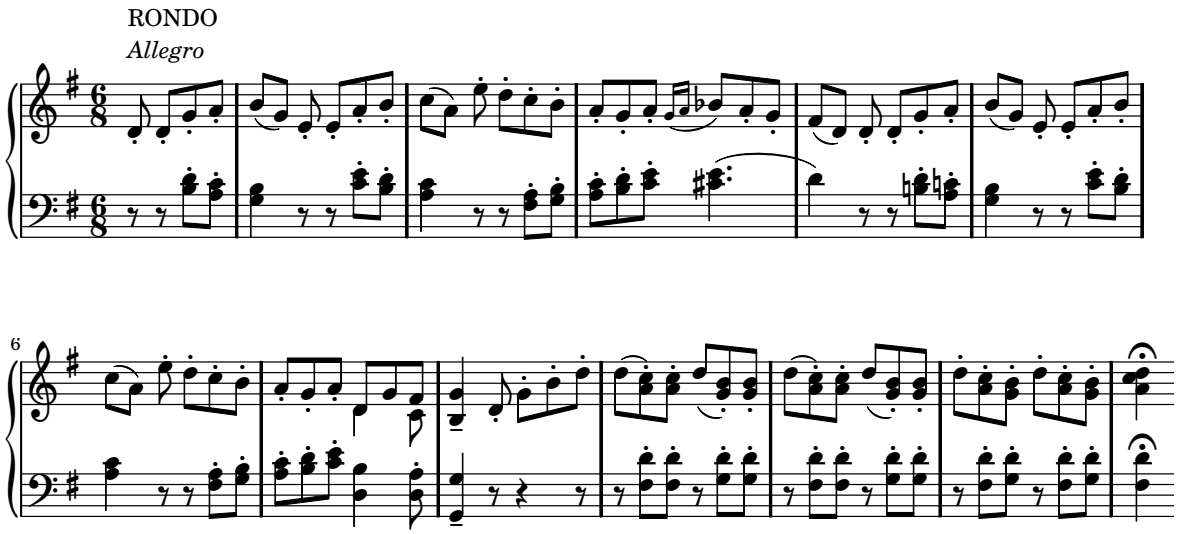
See also

This manual: Section 1.2.5.2 [Bar numbers], page 58, Section 1.2.3.3 [Unmetered music], page 45

Snippets: Rhythms

Internals Reference: `Timing_translator`, `Score`

1.3 Expressive marks



1.3.1 Attached to notes

1.3.1.1 Articulations and ornamentations

A variety of symbols can appear above and below notes to indicate different characteristics of the performance. All these symbols can be attached to a note using the syntax *note**name*.

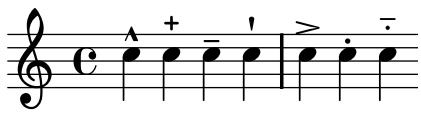
The possible values for *name* are listed in Section B.8 [List of articulations], page 314.

Some of these articulations have shorthands for easier entry. Shorthands are appended to the note name, and their syntax consists of a dash (`-`) followed by a symbol signifying the articulation. The available shorthands are:

- `-^` (*marcato*)
- `-+` (*stopped*)
- `--` (*tenuto*)
- `-|` (*staccatissimo*)
- `->` (*accent*)
- `-.` (*staccato*)
- `-_` (*portato*)

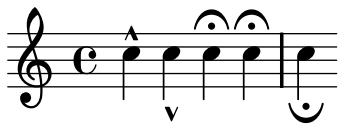
and their corresponding output:

```
c4-^  c-+  c--  c-|
c->   c-.  c-_
```



The marks are automatically placed, but the direction can be forced as well. Like other pieces of LilyPond code, `_` will place them below the staff, and `^` will place them above. This applies both to the shorthands and the fully named articulations. For the shorthands, the dash itself should be omitted; the direction signs replace it:

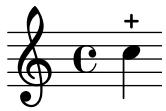
```
c4^^ c4_^
c\fermata c^\fermata c_\fermata
```



Selected Snippets

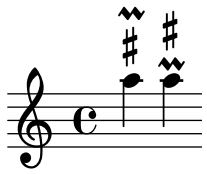
The shorthands are defined in '`ly/script-init.ly`', where the variables `dashHat`, `dashPlus`, `dashDash`, `dashBar`, `dashLarger`, `dashDot`, and `dashUnderscore` are assigned default values. The default values for the shorthands can be modified. For example, to associate the `-+` (`dashPlus`) shorthand with the *trill* symbol instead of the default `+` symbol, assign the value `trill` to the variable `dashPlus`:

```
\relative c'' { c-+ }
dashPlus = "trill"
\relative c'' { c-+ }
```



The vertical ordering of scripts is controlled with the `script-priority` property. The lower this number, the closer it will be put to the note. In this example, the `TextScript` (the sharp symbol) first has the lowest priority, so it is put lowest in the first example. In the second, the *prall trill* (the `Script`) has the lowest, so it is on the inside. When two objects have the same priority, the order in which they are entered decides which one comes first.

```
\once \override TextScript #'script-priority = #-100
a4^\prall^\markup { \sharp }

\once \override Script #'script-priority = #-100
a4^\prall^\markup { \sharp }
```



See also

Snippets: Expressive-marks

Internals Reference: `Script`, `TextScript`.

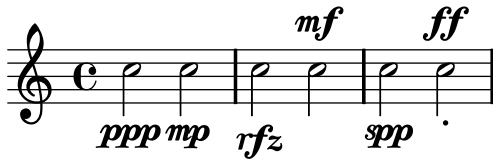
Known issues and warnings

These signs appear in the printed output but have no effect on the MIDI rendering of the music.

1.3.1.2 Dynamics

Absolute dynamic marks are specified using a command after a note, like `c4\ff`. The available dynamic marks are `\ppppp`, `\pppp`, `\ppp`, `\pp`, `\p`, `\mp`, `\mf`, `\f`, `\ff`, `\fff`, `\ffff`, `\fp`, `\sf`, `\sff`, `\sp`, `\spp`, `\sfz`, and `\rfz`. The dynamic marks can be placed above or below the staff with `_` and `^`, just like articulation marks.

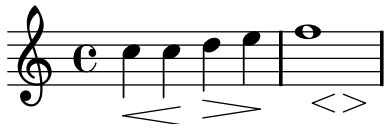
```
c2\ppp c\mp
c\rfz c^\mf
c_\spp c_\staccato^\ff
```



A *crescendo* mark is started with `\<` and terminated with `\!` or an absolute dynamic. A *decrescendo* is started with `\>` and is also terminated with `\!` or an absolute dynamic. `\cr` and `\decr` may be used instead of `\<` and `\>`. They can be engraved either using a graphical sign (a 'hairpin'), or with textual signs.

Because these marks are bound to notes, you must use spacer notes if multiple marks are needed during one note.

```
c\< c\! d\> e\!
<< f1 { s4 s4\< s4\! \> s4\! } >>
```

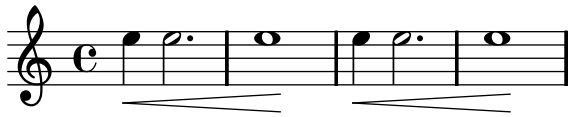


This may give rise to very short hairpins. To lengthen them, use for example `minimum-length` in `Voice.Hairpin`:

```
\override Voice.Hairpin #'minimum-length = #5
```

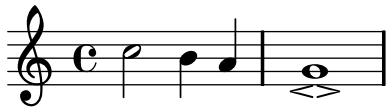
A hairpin normally starts at the left edge of the beginning note and ends on the right edge of the ending note. If the ending note falls on the downbeat, the hairpin ends on the immediately preceding bar line. This may be modified by setting the `hairpinToBarline` property:

```
e4\< e2. e1\!
\set hairpinToBarline = ##f
e4\< e2. e1\!
```



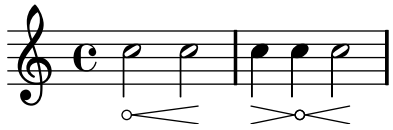
In some situations the `\espressivo` articulation mark may be suitable to indicate a crescendo and decrescendo on the one note:

```
c2 b4 a g1\espressivo
```



Hairpins may be printed with a circled tip (al niente notation) by setting the `circled-tip` property:

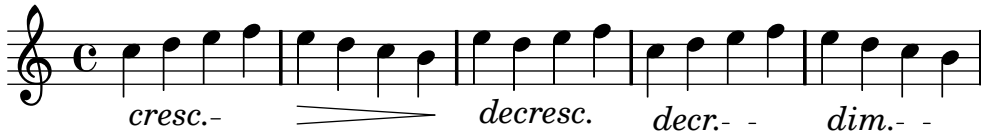
```
\override Hairpin #'circled-tip = ##t
c2\< c\!
c4\> c\< c2\!
```



You can use text saying *cresc.*, *decr.*, or *dim.* instead of hairpins with the commands \setTextCresc, \setTextDim, and \setTextDecresc. The corresponding \setHairpinCresc, \setHairpinDim, and \setHairpinDecresc will revert to hairpins again:

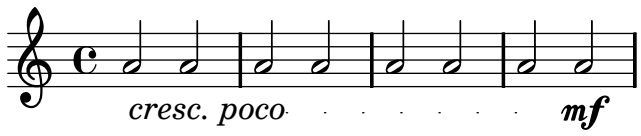
```
\setTextCresc
c\< d e f\!
\setHairpinCresc
e\> d c b\!
\setTextDecresc
```

```
e\> d e f\!
\setTextDecr
c\> d e f\!
\setTextDim
e\> d c b\!
```



You can also supply your own texts and change the style of the spanner line with the properties `\crescendoText`, `\crescendoSpanner`, `\decrescendoText`, and `\decrescendoSpanner`. Available values for the spanner properties are `hairpin`, `line`, `dashed-line`, and `dotted-line`. If unset, a hairpin crescendo is used:

```
\set crescendoText = \markup { \italic "cresc. poco" }
\set crescendoSpanner = #'dotted-line
a'2\< a a a a a a\!\mf
```



To create new dynamic marks or text that should be aligned with dynamics, see Section 1.8.3.1 [New dynamic marks], page 124.

Vertical positioning of dynamics is handled by `DynamicLineSpanner`.

Predefined commands

`\dynamicUp`, `\dynamicDown`, `\dynamicNeutral`.

Selected Snippets

Dynamics that occur at, begin on, or end on the same note will be vertically aligned. If you want to ensure that dynamics are aligned when they do not occur on the same note, you can increase the `staff-padding` property.

```
\override DynamicLineSpanner #'staff-padding = #4
```

You may also use this property if the dynamics are colliding with other notation.

Crescendi and decrescendi that cross a line break will be continued on the second line. If they end on the first note of a new line, nothing will be printed on that line. To change this behavior, use

```
\override Score.Hairpin #'after-line-breaking = ##t
```

Text style dynamic changes (such as *cresc.* and *dim.*) are printed with a dashed line showing their extent. To surpress printing this line, use

```
\override DynamicTextSpanner #'dash-period = #-1.0
```

See also

Music Glossary: ⟨undefined⟩ [hairpin], page ⟨undefined⟩, ⟨undefined⟩ [crescendo], page ⟨undefined⟩, ⟨undefined⟩ [decrescendo], page ⟨undefined⟩.

Learning Manual: learning manual, ⟨undefined⟩ [Articulation and dynamics], page ⟨undefined⟩ .

Snippets: Expressive-marks

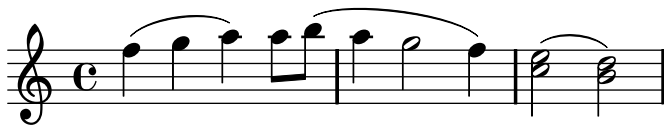
Internals Reference: `DynamicText`, `Hairpin`. Vertical positioning of these symbols is handled by `DynamicLineSpanner`.

1.3.2 Curves

1.3.2.1 Slurs

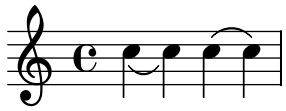
A slur indicates that notes are to be played bound or *legato*. They are entered using parentheses:

```
f( g a) a8 b( a4 g2 f4)
<c e>2( <b d>2)
```



Just as with ties, the direction of a slur can be specified with `\slur`*`DIR`*, where *`DIR`* is either `Up`, `Down`, or `Neutral` (automatically selected). The shorthands are also available: by adding `_` or `^` before the opening parentheses, the direction is also set.

```
c4_( c) c^( c)
```



Only one slur can be printed at once. If you need to print a long slur over a few small slurs, please see Section 1.3.2.2 [Phrasing slurs], page 74.

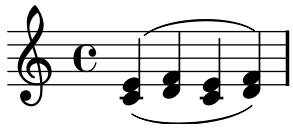
Predefined commands

`\slurUp`, `\slurDown`, `\slurNeutral`, `\slurDashed`, `\slurDotted`, `\slurSolid`.

Selected Snippets

Some composers write two slurs when they want legato chords. This can be achieved in LilyPond by setting `doubleSlurs`,

```
\set doubleSlurs = ##t
<c e>4 ( <d f> <c e> <d f> )
```



See also

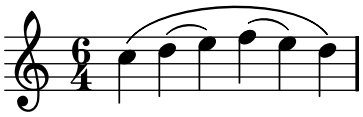
Snippets: Expressive-marks

Internals Reference: `Slur`.

1.3.2.2 Phrasing slurs

A phrasing slur (or phrasing mark) connects notes and is used to indicate a musical sentence. It is written using `\(` and `\)` respectively:

```
\time 6/4 c'\( d( e) f( e) d\)
```



Typographically, the phrasing slur behaves almost exactly like a normal slur. However, they are treated as different objects. A `\slurUp` will have no effect on a phrasing slur; instead, use `\phrasingSlurUp`, `\phrasingSlurDown`, and `\phrasingSlurNeutral`, or use the shorthands `_` and `^`.

You cannot have simultaneous phrasing slurs.

Predefined commands

`\phrasingSlurUp`, `\phrasingSlurDown`, `\phrasingSlurNeutral`.

See also

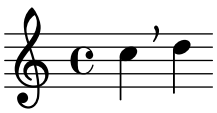
Snippets: Expressive-marks

Internals Reference: `PhrasingSlur`.

1.3.2.3 Breath marks

Breath marks are entered using `\breathe`:

```
c'4 \breathe d4
```



Selected Snippets

The glyph of the breath mark can be tuned by overriding the `text` property of the `BreathingSign` layout object with any markup text. For example,

```
c'4
\override BreathingSign #'text
  = #(make-musicglyph-markup "scripts.rvarcomma")
\breathe
```

```
d4
```

See also

Snippets: Expressive-marks ,
Internals Reference: `BreathingSign`.

1.3.2.4 Falls and doits

Falls and doits can be added to notes using the `\bendAfter` command,

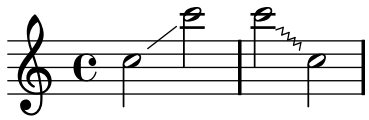
```
\override Score.SpacingSpanner #'shortest-duration-space = #3.0
c4-\bendAfter #+5
c4-\bendAfter #-3
```

1.3.3 Lines

1.3.3.1 Glissando

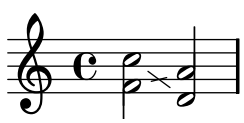
A glissando is a smooth change in pitch. It is denoted by a line or a wavy line between two notes. It is requested by attaching `\glissando` to a note:

```
c2\glissando c'
\override Glissando #'style = #'zigzag
c2\glissando c,
```



Selected Snippets

```
I = \once \override NoteColumn #'ignore-collision = ##t

\relative c' <<
  { \oneVoice \stemDown f2 \glissando \stemNeutral a } \\
  { \oneVoice \I c2 \glissando \I d, }
>>
```



See also

Music Glossary: ⟨undefined⟩ [falls], page ⟨undefined⟩, ⟨undefined⟩ [doits], page ⟨undefined⟩.
Snippets: Expressive-marks .
Internals Reference: `Glissando`.

Known issues and warnings

Printing text over the line (such as *gliss.*) is not supported.

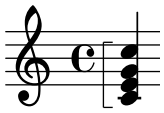
1.3.3.2 Arpeggio

You can specify an arpeggio sign (also known as broken chord) on a chord by attaching an `\arpeggio` to the chord:

```
<c e g c>\arpeggio
```

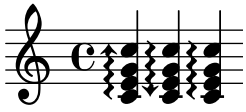
A square bracket on the left is used to indicate that the chord should *not* be arpeggiated:

```
\arpeggioBracket
<c e g c>\arpeggio
```



The direction of the arpeggio can be denoted by adding an arrowhead to the wiggly line. This is done with the commands `arpeggioUp` and `arpeggioDown`. `arpeggioNeutral` reverts to the arrow-less version:

```
\new Voice {
  \arpeggioUp
  <c e g c>\arpeggio
  \arpeggioDown
  <c e g c>\arpeggio
  \arpeggioNeutral
  <c e g c>\arpeggio
}
```



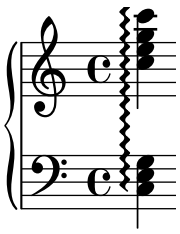
Predefined commands

`\arpeggio`, `\arpeggioUp`, `\arpeggioDown`, `\arpeggioNeutral`, `\arpeggioBracket`.

Selected Snippets

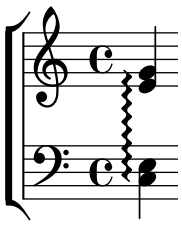
In a `PianoStaff`, it is possible to let an arpeggio cross between the staves by setting the property `PianoStaff.connectArpeggios`.

```
\new PianoStaff <<
  \set PianoStaff.connectArpeggios = ##t
  \new Staff { <c' e g c>\arpeggio }
  \new Staff { \clef bass <c,, e g>\arpeggio }
>>
```



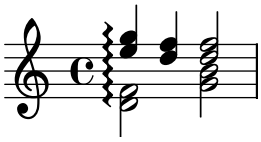
The same can be accomplished in contexts other than `PianoStaff` if the `Span_arpeggio_engraver` is included in the Score context.

```
\score {
  \new StaffGroup {
    \set Score.connectArpeggios = ##t
    <<
      \new Voice \relative c' {
        <e g>4\arpeggio
      }
      \new Voice  \relative c {
        \clef bass
        <c e>4\arpeggio
      }
    >>
  }
  \layout {
    \context {
      \Score
      \consists "Span_arpeggio_engraver"
    }
  }
}
```



Similarly, an arpeggio can be drawn across notes in different voices on the same staff if the `Span_arpeggio_engraver` is moved to the Staff context:

```
\new Staff
\with {
  \consists "Span_arpeggio_engraver"
} \relative c' {
  \set Staff.connectArpeggios = ##t
    <<
    {<e' g>4\arpeggio <d f> <d f>2 }
  \\
  {<d, f>2\arpeggio  <g b>2  }
  >>
}
```



See also

Notation Reference: Section 1.2.1.4 [Ties], page 35, for writing out arpeggios.

Snippets: Expressive-marks

Internals Reference: `Arpeggio`, `PianoStaff`.

Known issues and warnings

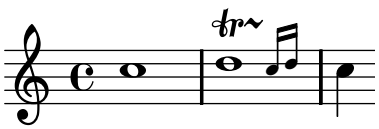
It is not possible to mix connected arpeggios and unconnected arpeggios in one `PianoStaff` at the same point in time.

1.3.3.3 Trills

Short trills are printed with `\trill` like normal articulation; see Section 1.3.1.1 [Articulations and ornamentations], page 68.

Long running trills are made with `\startTrillSpan` and `\stopTrillSpan`. In the following example, it is shown in the common combination with grace notes. If a more precise control of the placement of the grace notes is needed, see Section 1.2.6.1 [Grace notes], page 63.

```
c1 \afterGrace
d1\startTrillSpan { c16[\stopTrillSpan d] }
c4
```



Trills that should be executed on an explicitly specified pitch can be typeset with the command `pitchedTrill`, and the syntax `\pitchedTrill` *mainnote*`\startTrillSpan` *trillnote endnote* `\stopTrillSpan`.

```
\pitchedTrill e2 \startTrillSpan fis
d\stopTrillSpan
```

The first argument is the main note. The pitch of the second is printed as a stemless note head in parentheses.

Predefined commands

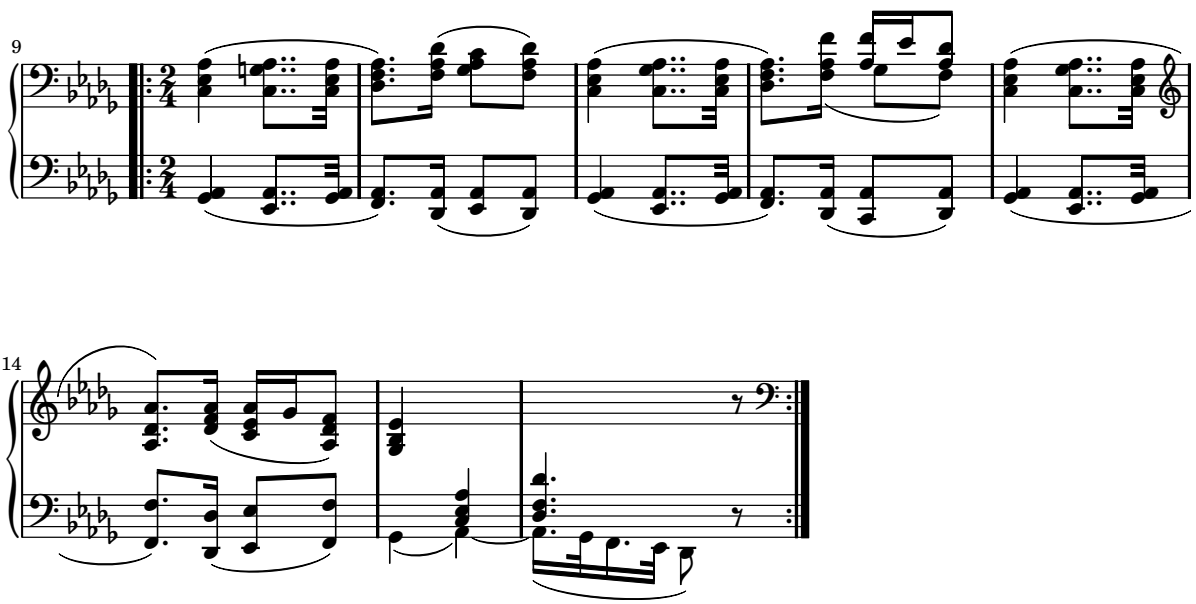
`\startTrillSpan`, `\stopTrillSpan`.

See also

Snippets: Expressive-marks

Internals Reference: `TrillSpanner`.

1.4 Repeats



Repetition is a central concept in music, and multiple notations exist for repetitions.

1.4.1 Writing repeats

Blah blah

1.4.1.1 Repeat syntax

LilyPond has one syntactic construct for specifying different types of repeats. The syntax is

```
\repeat variant repeatcount repeatbody
```

where *repeabody* is a music expression.

The following types of repetition are supported

`volta` The repeated music is not written out but enclosed in repeat bar lines. If the repeat is at the beginning of a piece, a repeat bar line is only printed at the end. Alternative endings (volte) are printed, left to right with brackets. This is the standard notation for repeats with alternatives.

`unfold` The repeated music is fully written out, as many times as specified by *repeatcount*. This is useful when entering repetitious music.

`tremolo` Make tremolo beams.

`percent` Make beat or measure repeats. These look like percent signs. Percent repeats must be declared within a `Voice` context.

Alternative endings are entered with

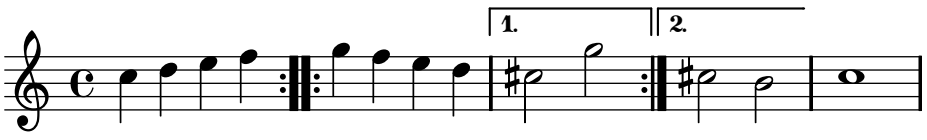
```
\alternative {
  alternative1
  alternative2
  alternative3
  ...
}
```

after a `\repeat volta` or `unfold` block, where each *alternative* is a music expression. If you give less alternatives than *repeatcount*, the first alternative is assumed to be played more than once.

1.4.1.2 Normal repeats

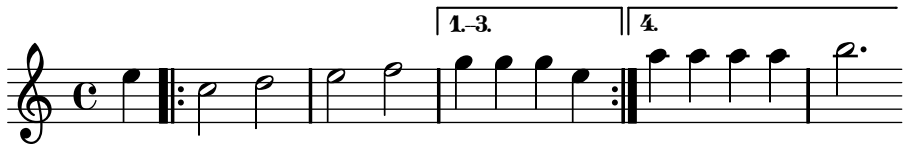
Normal repeats, with or without alternate repeats:

```
\repeat volta 2 { c4 d e f }
\repeat volta 2 { g f e d }
  \alternative {
    { cis2 g' }
    { cis,2 b }
  }
c1
```



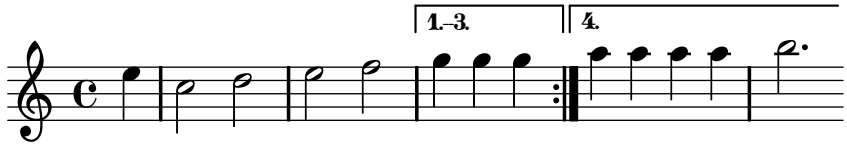
Repeats with upbeats:

```
\new Staff {
  \partial 4 e |
  \repeat volta 4 { c2 d2 | e2 f2 | }
  \alternative { { g4 g g e } { a a a a | b2. } }
}
```



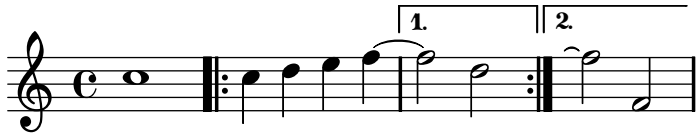
or

```
\new Staff {
  \partial 4
  \repeat volta 4 { e | c2 d2 | e2 f2 | }
  \alternative { { \partial 4*3 g4 g g } { a a a a | b2. } }
}
```



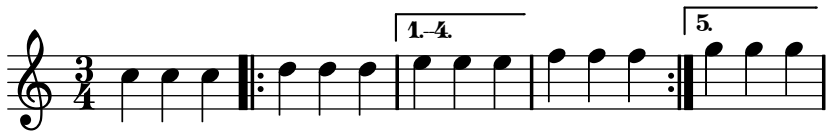
Ties may be added to a second ending:

```
c1
\repeat volta 2 {c4 d e f ~ }
\alternative { {f2 d} {f\repeatTie f,} }
```



By default, the volta brackets will be drawn over all of the alternative music, but it is possible to shorten them by setting `voltaSpannerDuration`. In the next example, the bracket only lasts one measure, which is a duration of 3/4.

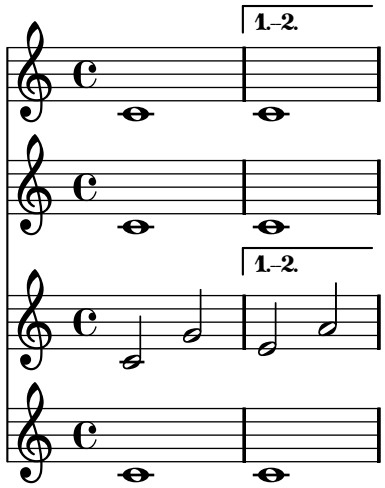
```
\relative c''{
  \time 3/4
  c c c
  \set Score.voltaSpannerDuration = #(ly:make-moment 3 4)
  \repeat volta 5 { d d d }
  \alternative { { e e e f f f }
  { g g g } }
}
```



The `Volta_engraver` by default resides in the Score context, and brackets for the repeat are thus normally only printed over the topmost staff. This can be adjusted by adding `Volta_engraver` to the Staff context where you want the brackets to appear; see Section 6.1.4 [Modifying context plug-ins], page 253 and

```
\score { <<
  \new Staff { \repeat volta 2 { c'1 } \alternative { c' } }
  \new Staff { \repeat volta 2 { c'1 } \alternative { c' } }
  \new Staff \with { \consists Volta_engraver } { c'2 g' e' a' }
```

```
 \new Staff { \repeat volta 2 { c'1 } \alternative { c' } }
>> }
```



If you want to start a repeat at the beginning of a line and have a double bar at the end of the previous line, use

```
... \bar "||:" \break
\repeat volta 2 { ...
```

see Section 1.2.5.1 [Bar lines], page 55 for more information.

See also

Snippets: Repeats

Internals Reference: `VoltaBracket`, `RepeatedMusic`, `VoltaRepeatedMusic`, and `UnfoldedRepeatedMusic`.

Known issues and warnings

A nested repeat like

```
\repeat ...
\repeat ...
\alternative
```

is ambiguous, since it is is not clear to which `\repeat` the `\alternative` belongs. This ambiguity is resolved by always having the `\alternative` belong to the inner `\repeat`. For clarity, it is advisable to use braces in such situations.

Timing information is not remembered at the start of an alternative, so after a repeat timing information must be reset by hand; for example, by setting `Score.measurePosition` or entering `\partial`. Similarly, slurs or ties are also not repeated.

1.4.1.3 Manual repeat commands

The property `repeatCommands` can be used to control the layout of repeats. Its value is a Scheme list of repeat commands.

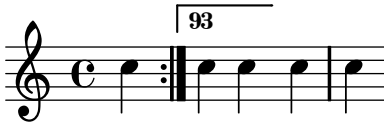
`start-repeat`
: Print a `|:` bar line.

`end-repeat`
: Print a `:|` bar line.

`(volta` *text*`)`
: Print a volta bracket saying *text*. The text can be specified as a text string or as a markup text, see Section 1.8.2 [Text markup], page 119. Do not forget to change the font, as the default number font does not contain alphabetic characters;

`(volta #f)`
: Stop a running volta bracket.

```
c4
  \set Score.repeatCommands = #'((volta "93") end-repeat)
c4 c4
  \set Score.repeatCommands = #'((volta #f))
c4 c4
```



See also

Notation Reference: Section 1.2.5.1 [Bar lines], page 55.

Snippets: Repeats

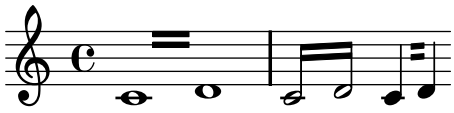
Internals Reference: `VoltaBracket`, `RepeatedMusic`, `VoltaRepeatedMusic`, and `UnfoldedRepeatedMusic`.

1.4.2 Other repeats

1.4.2.1 Tremolo repeats

To place tremolo marks between notes, use `\repeat` with tremolo style:

```
\new Voice \relative c' {
  \repeat tremolo 8 { c16 d16 }
  \repeat tremolo 4 { c16 d16 }
  \repeat tremolo 2 { c16 d16 }
}
```

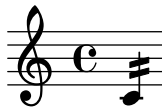


The `\repeat tremolo` syntax expects exactly two notes within the braces, and the number of repetitions must correspond to a note value that can be expressed with plain or dotted notes. Thus, `\repeat tremolo 7` is valid and produces a double dotted note, but `\repeat tremolo 9` is not.

The duration of the tremolo equals the duration of the braced expression multiplied by the number of repeats: `\repeat tremolo 8 { c16 d16 }` gives a whole note tremolo, notated as two whole notes joined by tremolo beams.

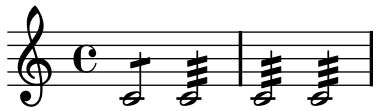
There are two ways to put tremolo marks on a single note. The `\repeat tremolo` syntax can be used even here, in which case the note should not be surrounded by braces:

```
\repeat tremolo 4 c'16
```



The same output can be obtained by adding '`:`[*number*]' after the note. The number indicates the duration of the subdivision, and it must be at least 8. A *number* value of 8 gives one line across the note stem. If the length is omitted, the last value (stored in `tremoloFlags`) is used

```
c'2:8 c':32 | c': c': |
```



Known issues and warnings

Tremolos entered with '`:`[*number*]' do not carry over into the MIDI output.

See also

Notation Reference: Section 1.4.2.1 [Tremolo repeats], page 83.

Internals Reference: `Beam`, `StemTremolo`.

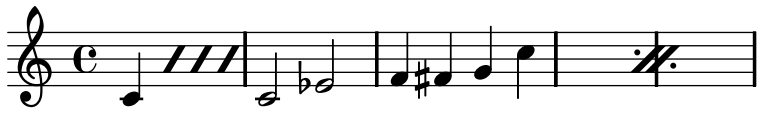
Snippets: Repeats

Elsewhere: `StemTremolo`.

1.4.2.2 Measure repeats

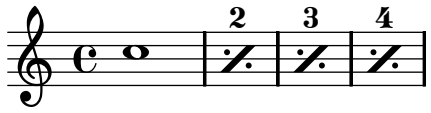
A note pattern can be repeated with the `\repeat percent` ***number*** syntax. The music is printed once, and the pattern is replaced with a special sign. Patterns of one and two measures are replaced by percent-like signs, patterns that divide the measure length are replaced by slashes. Percent repeats must be declared within a `Voice` context.

```
\new Voice \relative c' {
  \repeat percent 4 { c4 }
  \repeat percent 2 { c2 es2 f4 fis4 g4 c4 }
}
```



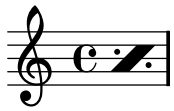
Measure repeats of more than two measures get a counter if you switch on the `countPercentRepeats` property:

```
\new Voice {
  \set countPercentRepeats = ##t
  \repeat percent 4 { c1 }
}
```



Isolated percents can also be printed. This is done by entering a multi-measure rest with a different print function:

```
\override MultiMeasureRest #'stencil
  = #ly:multi-measure-rest::percent
R1
```

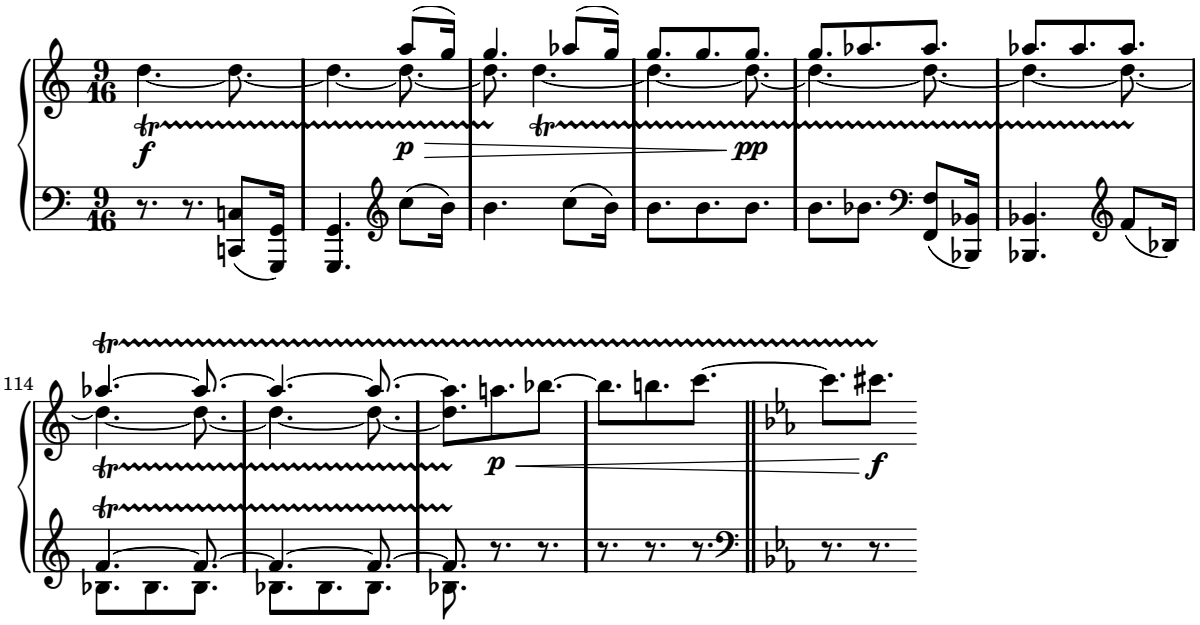


See also

Snippets: Repeats

Internals Reference: `RepeatSlash`, `PercentRepeat`, `DoublePercentRepeat`, `DoublePercentRepeatCounter`, `PercentRepeatCounter`, `PercentRepeatedMusic`.

1.5 Simultaneous notes



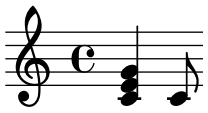
Polyphony in music refers to having more than one voice occurring in a piece of music. Polyphony in LilyPond refers to having more than one voice on the same staff.

1.5.1 Single voice

1.5.1.1 Chorded notes

A chord is formed by a enclosing a set of pitches between `<` and `>`. A chord may be followed by a duration, and a set of articulations, just like simple notes:

```
<c e g>4 <c>8
```



For more information about chords, see Section 2.2.1.2 [Introducing chord names], page 146.

See also

Music Glossary: ⟨undefined⟩ [chord], page ⟨undefined⟩.

Notation Reference: Section 2.2.1.2 [Introducing chord names], page 146.

Snippets: Simultaneous-notes .

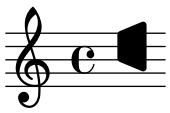
Known issues and warnings

Music expressions like `<< { g8 e8 } a4 >>` are not printed accurately. Use `<g a>8 <e a>8` instead.

1.5.1.2 Clusters

A cluster indicates a continuous range of pitches to be played. They can be denoted as the envelope of a set of notes. They are entered by applying the function `makeClusters` to a sequence of chords, e.g.,

```
\makeClusters { <c e > <b f'> }
```



Ordinary notes and clusters can be put together in the same staff, even simultaneously. In such a case no attempt is made to automatically avoid collisions between ordinary notes and clusters.

See also

Snippets: Simultaneous-notes .

Internals Reference: `ClusterSpanner`, `ClusterSpannerBeacon`, `Cluster_spanner_engraver`.

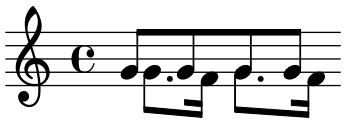
Examples:

1.5.2 Multiple voices

1.5.2.1 Collision resolution

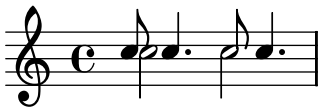
Normally, note heads with a different number of dots are not merged, but when the object property `merge-differently-dotted` is set in the `NoteCollision` object, they are merged:

```
\new Voice << {
  g8 g8
  \override Staff.NoteCollision
    #'merge-differently-dotted = ##t
  g8 g8
} \\ { g8.[ f16] g8.[ f16] } >>
```



Similarly, you can merge half note heads with eighth notes, by setting `merge-differently-headed`:

```
\new Voice << {
  c8 c4.
  \override Staff.NoteCollision
    #'merge-differently-headed = ##t
c8 c4. } \\ { c2 c2 } >>
```



`merge-differently-headed` and `merge-differently-dotted` only apply to opposing stem directions (i.e. Voice 1 & 2).

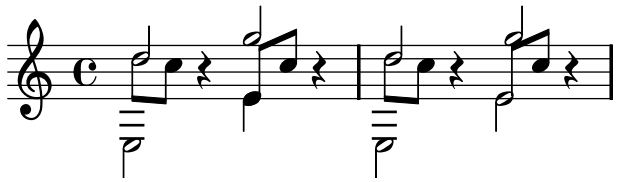
LilyPond also vertically shifts rests that are opposite of a stem, for example

```
\new Voice << c''4 \\ r4 >>
```

If three or more notes line up in the same column, `merge-differently-headed` cannot successfully complete the merge of the two notes that should be merged. To allow the merge to work properly, apply a `\shift` to the note that should not be merged. In the first measure of following example, `merge-differently-headed` does not work (the half-note head is solid). In the second measure, `\shiftOn` is applied to move the top `g` out of the column, and `merge-differently-headed` works properly.

```
\override Staff.NoteCollision #'merge-differently-headed = ##t
<<
  { d=''2 g2 } \\
  { \oneVoice d=''8 c8 r4 e,8 c'8 r4 } \\
  { \voiceFour e,,2 e'2}
>>
<<
```

```
  { d''2 \shiftOn g2 } \\
  { \oneVoice d''8 c8 r4 e,8 c'8 r4 } \\
  { \voiceFour e,,2 e'2}
>>
```



In some instances of complex polyphonic music, you may need additional voices to avoid collisions between notes. Additional voices are added by defining an variable, as shown below:

```
voiceFive = #(context-spec-music (make-voice-props-set 4) 'Voice)

\relative c''' <<
  { \voiceOne g4 ~  \stemDown g32[ f( es d c b a b64 )g] } \\
  { \voiceThree  b4} \\
  { \voiceFive d,} \\
  { \voiceTwo g,}
>>
```

Predefined commands

`\oneVoice`, `\voiceOne`, `\voiceTwo`, `\voiceThree`, `\voiceFour`.

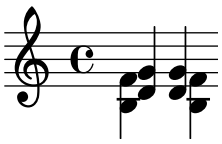
```
\voiceNeutralStyle
\voiceOneStyle
\voiceTwoStyle
\voiceThreeStyle
\voiceFourStyle
```

`\shiftOn`, `\shiftOnn`, `\shiftOnnn`, `\shiftOff`: these commands specify the degree to which chords of the current voice should be shifted. The outer voices (normally: voice one and two) have `\shiftOff`, while the inner voices (three and four) have `\shiftOn`. `\shiftOnn` and `\shiftOnnn` define further shift levels.

When LilyPond cannot cope, the `force-hshift` property of the `NoteColumn` object and pitched rests can be used to override typesetting decisions.

```
\relative <<
{
  <d g>
  <d g>
} \\ {
  <b f'>
  \once \override NoteColumn #'force-hshift = #1.7
```

```
  <b f'>
} >>
```



See also

Snippets: Simultaneous-notes .

Internals Reference: the objects responsible for resolving collisions are `NoteCollision` and `RestCollision`.

Known issues and warnings

When using `merge-differently-headed` with an upstem eighth or a shorter note, and a downstem half note, the eighth note gets the wrong offset.

There is no support for clusters where the same note occurs with different accidentals in the same chord. In this case, it is recommended to use enharmonic transcription, or to use special cluster notation (see Section 1.5.1.2 [Clusters], page 86).

1.5.2.2 Automatic part combining

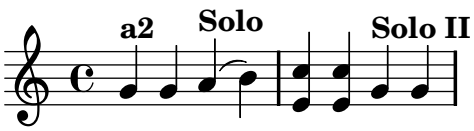
Automatic part combining is used to merge two parts of music onto a staff. It is aimed at typesetting orchestral scores. When the two parts are identical for a period of time, only one is shown. In places where the two parts differ, they are typeset as separate voices, and stem directions are set automatically. Also, solo and *a due* parts are identified and can be marked.

The syntax for part combining is

`\partcombine` *`musicexpr1 musicexpr2`*

The following example demonstrates the basic functionality of the part combiner: putting parts on one staff, and setting stem directions and polyphony.

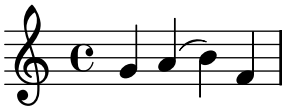
```
\new Staff \partcombine
  \relative g' { g g a( b) c c r r }
  \relative g' { g g r4 r e e g g }
```



The first `g` appears only once, although it was specified twice (once in each part). Stem, slur, and tie directions are set automatically, depending whether there is a solo or unisono. The first part (with context called `one`) always gets up stems, and 'Solo', while the second (called `two`) always gets down stems and 'Solo II'.

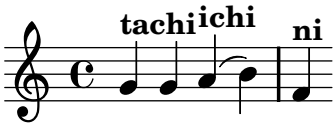
If you just want the merging parts, and not the textual markings, you may set the property `printPartCombineTexts` to false.

```
\new Staff <<
  \set Staff.printPartCombineTexts = ##f
  \partcombine
    \relative g' { g a( b) r }
    \relative g' { g r4 r f }
>>
```



To change the text that is printed for solos or merging, you may set the `soloText`, `soloIIText`, and `aDueText` properties.

```
\new Staff <<
  \set Score.soloText = #"ichi"
  \set Score.soloIIText = #"ni"
  \set Score.aDueText = #"tachi"
  \partcombine
    \relative g' { g4 g a( b) r }
    \relative g' { g4 g r r f }
>>
```



Both arguments to `\partcombine` will be interpreted as `Voice` contexts. If using relative octaves, `\relative` should be specified for both music expressions, i.e.,

```
\partcombine
  \relative ... musicexpr1
  \relative ... musicexpr2
```

A `\relative` section that is outside of `\partcombine` has no effect on the pitches of *musicexpr1* and *musicexpr2*.

See also

Music Glossary: ⟨undefined⟩ [a due], page ⟨undefined⟩.

Snippets: Simultaneous-notes .

Internals Reference: `PartCombineMusic`, `Voice`.

Known issues and warnings

When `printPartCombineTexts` is set, if the two voices play the same notes on and off, the part combiner may typeset `a2` more than once in a measure.

`\partcombine` cannot be inside `\times`.

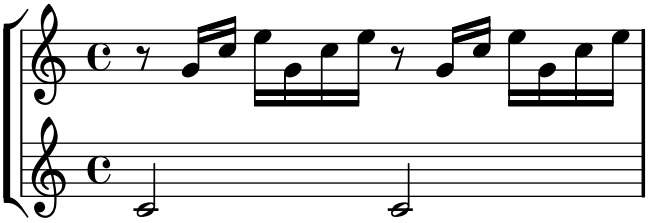
`\partcombine` cannot be inside `\relative`.

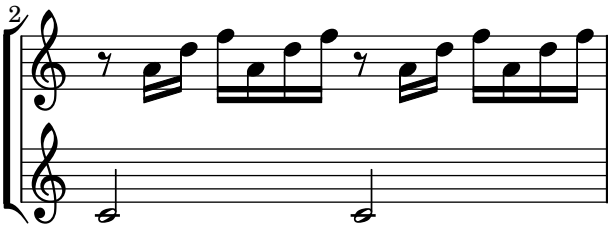
Internally, the `\partcombine` interprets both arguments as `Voices` named `one` and `two`, and then decides when the parts can be combined. Consequently, if the arguments switch to differently named `Voice` contexts, the events in those will be ignored.

1.5.2.3 Writing music in parallel

Music for multiple parts can be interleaved:

```
\parallelMusic #'(voiceA voiceB) {
  r8 g'16[ c''] e''[ g' c'' e''] r8 g'16[ c''] e''[ g' c'' e''] |
  c'2                            c'2                           |
  r8 a'16[ d''] f''[ a' d'' f''] r8 a'16[ d''] f''[ a' d'' f''] |
  c'2                            c'2                           |
}
\new StaffGroup <<
  \new Staff \new Voice \voiceA
  \new Staff \new Voice \voiceB
>>
```

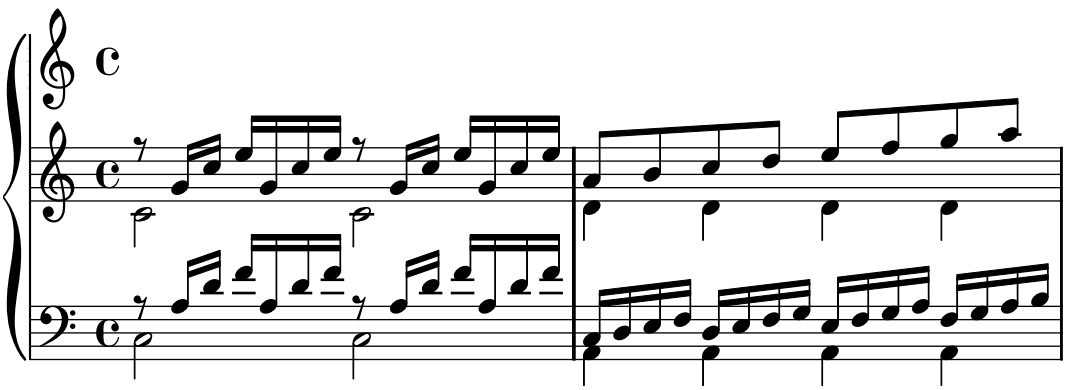




This works quite well for piano music.

```
music = {
  \key c \major
  \time 4/4
  \parallelMusic #'(voiceA voiceB voiceC voiceD) {
    % Bar 1
    r8  g'16[ c''] e''[ g' c'' e'']
      r8 g'16[ c''] e''[ g' c'' e''] |
    c'2
      c'2 |
    r8  a16[ d'] f'[ a d' f']
      r8  a16[ d'] f'[ a d' f'] |
    c2
      c2 |

    % Bar 2
    a'8 b'      c'' d''    e'' f''    g'' a'' |
    d'4         d'         d'         d' |
    c16 d e f   d e f g    e f g a    f g a b |
    a,4         a,4        a,4        a,4 |

    % Bar 3 ...
```

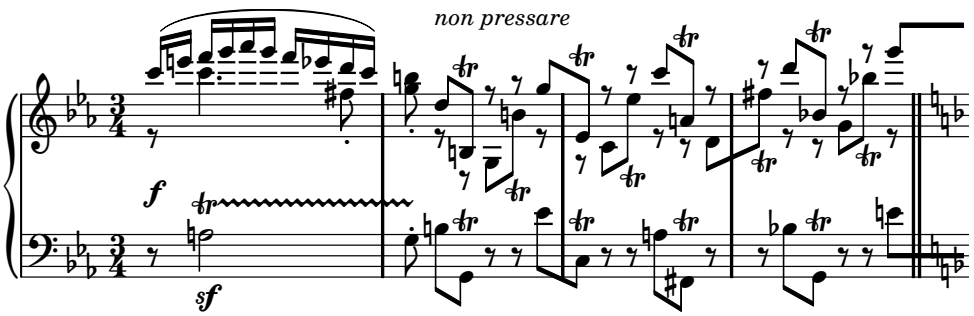
```
  }
}

\score {
  \new PianoStaff <<
    \music
    \new Staff <<
      \voiceA \\
      \voiceB
    >>
    \new Staff {
      \clef bass
      <<
        \voiceC \\
        \voiceD
      >>
    }
  >>
}
```

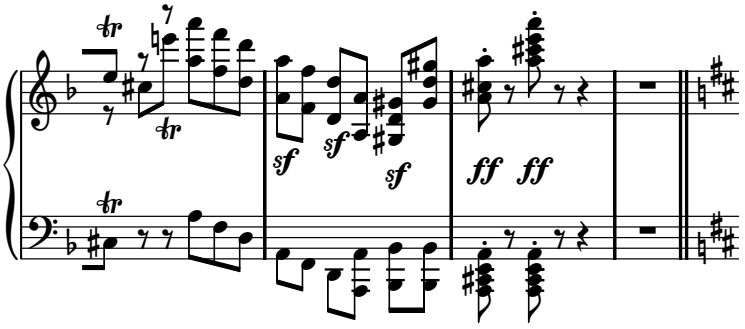


See also

Snippets: Simultaneous-notes

1.6 Staff notation





Notes, dynamic signs, rests, etc., are grouped with a set of horizontal lines, called a staff (plural 'staves'). In LilyPond, these lines are drawn using a separate layout object called `staff symbol`.

Two or more staves can be grouped vertically in a `GrandStaff`, a `StaffGroup`, or a `ChoirStaff`.

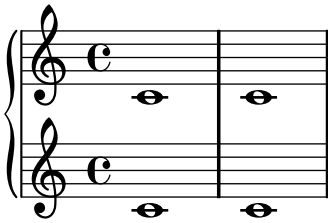
1.6.1 Displaying staves

1.6.1.1 System start delimiters

Many scores consist of more than one staff. These staves can be grouped in several different ways:

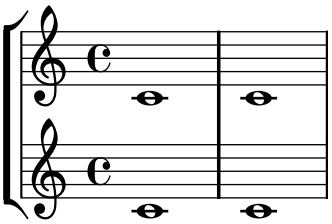
- In a `GrandStaff`, the group is started with a brace at the left, and bar lines are connected between the staves.

```
\new GrandStaff
\relative <<
  \new Staff { c1 c }
  \new Staff { c c }
>>
```



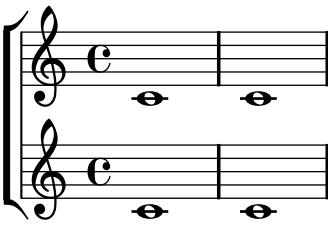
- In a `StaffGroup`, the bar lines will be drawn through all the staves, but the group is started with a bracket.

```
\new StaffGroup
\relative <<
  \new Staff { c1 c }
  \new Staff { c c }
>>
```



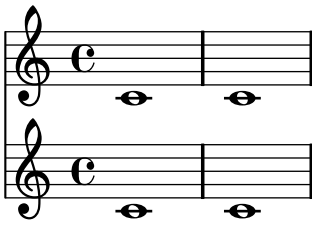
- In a `ChoirStaff`, the group is started with a bracket, but bar lines are not connected.

```
\new ChoirStaff
\relative <<
  \new Staff { c1 c }
  \new Staff { c c }
>>
```



- If no context is specified, the default properties for the score will be used: the group is started with a vertical line, and the bar lines are not connected.

```
\relative <<
  \new Staff { c1 c }
  \new Staff { c c }
>>
```



In addition to these four staff group types, other groupings can be produced by changing various properties. E.g., the 'Mensurstriche' layout common in Renaissance music, with bar lines running between but not through the staves, can be produced from a `StaffGroup` or `GrandStaff` context if the bar lines are made transparent in the `Staff` itself, with the command `\override Staff.BarLine #'transparent = ##t`

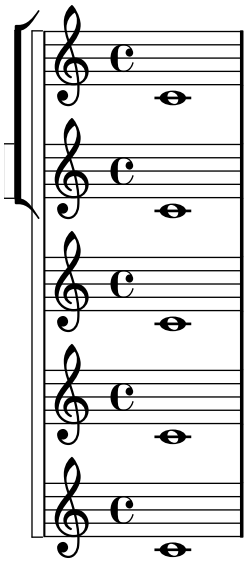
Staff groups can be nested, using the context `InnerStaffGroup` or `InnerChoirStaff`; see

Selected Snippets

More complex nesting can be accomplished using the property `systemStartDelimiterHierarchy`:

```
\new StaffGroup
\relative <<
  \set StaffGroup.systemStartDelimiterHierarchy
    = #'(SystemStartSquare (SystemStartBracket a
                            (SystemStartSquare b)) d)
  \new Staff { c1 }
  \new Staff { c1 }
  \new Staff { c1 }
  \new Staff { c1 }
  \new Staff { c1 }
```

>>



Each staff group context sets the property `systemStartDelimiter` to one of the values `SystemStartBar`, `SystemStartBrace`, and `SystemStartBracket`. A fourth delimiter, `systemStartSquare`, is also available, but must be instantiated manually

To display a bracket even if there is only one staff, see

See also

Music Glossary: ⟨undefined⟩ [brace], page ⟨undefined⟩, ⟨undefined⟩ [staff], page ⟨undefined⟩, ⟨undefined⟩ [staves], page ⟨undefined⟩ ⟨undefined⟩ [bracket], page ⟨undefined⟩.

Snippets: Staff-notation

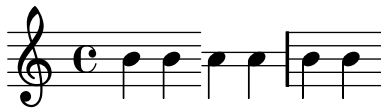
Internals Reference: `ChoirStaff`, `GrandStaff`, `StaffGroup`, `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket`, `systemStartDelimiterHierarchy`.

1.6.1.2 Staff symbol

The layout object which draws the lines of a staff is called `staff symbol`. The staff symbol may be tuned in the number, thickness and distance of lines, using properties. This is demonstrated in the example files

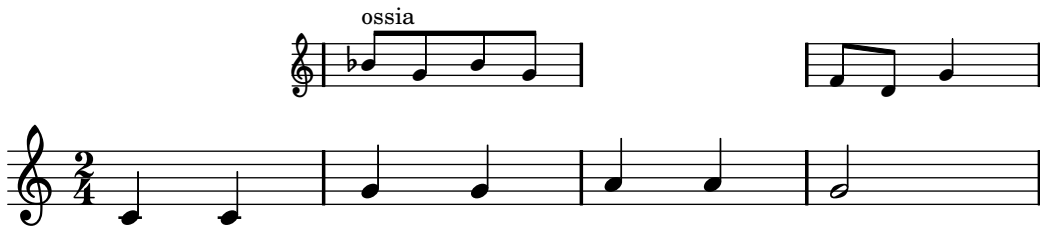
In addition, staves may be started and stopped at will. This is done with `\startStaff` and `\stopStaff`.

```
b4 b
\override Staff.StaffSymbol #'line-count = 2
\stopStaff \startStaff
b b
\revert Staff.StaffSymbol #'line-count
\stopStaff \startStaff
b b
```



In combination with Frenched staves, this may be used to typeset *ossia* sections. An example is shown here

```
<<
  \new Staff \with
  {
    \remove "Time_signature_engraver"
    fontSize = #-2
    \override StaffSymbol #'staff-space = #(magstep -2)
    firstClef = ##f
  }
  \relative c'' {
    \stopStaff
    \skip 2

    \startStaff
    \clef treble
    bes8[^"ossia" g bes g]
    \stopStaff

    s2

    \startStaff
    f8 d g4
  }
  \new Staff  \relative
  {
    \time 2/4
    c4 c g' g a a g2
  }

>>
```



See also

Music Glossary: ⟨undefined⟩ [ossia], page ⟨undefined⟩, ⟨undefined⟩ [staff], page ⟨undefined⟩, ⟨undefined⟩ [Frenched staff], page ⟨undefined⟩.

Snippets: Staff-notation

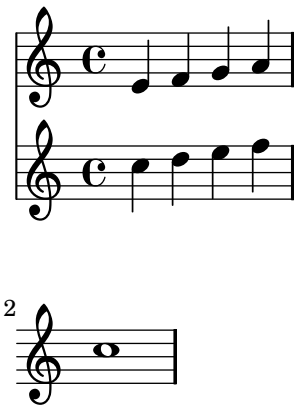
Internals Reference: `StaffSymbol`, `DrumStaff`.

1.6.1.3 Hiding staves

In orchestral scores, staff lines that only have rests are usually removed; this saves some space. This style is called 'French Score'. For `Lyrics`, `ChordNames` and `FiguredBass`, this is switched on by default. When the lines of these contexts turn out empty after the line-breaking process, they are removed.

For normal staves, a specialized `Staff` context is available, which does the same: staves containing nothing (or only multi-measure rests) are removed. The context definition is stored in `\RemoveEmptyStaffContext` variable. Observe how the second staff in this example disappears in the second line

```
\layout {
  \context { \RemoveEmptyStaffContext }
}

{
  \relative c' <<
    \new Staff { e4 f g a \break c1 }
    \new Staff { c4 d e f \break R1 }
  >>
}
```



The first system shows all staves in full. If empty staves should be removed from the first system too, set `remove-first` to true in `VerticalAxisGroup`.

```
\override Score.VerticalAxisGroup #'remove-first = ##t
```

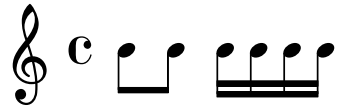
To remove other types of contexts, use `\AncientRemoveEmptyStaffContext` or `\RemoveEmptyRhythmicStaffContext`.

Another application of the `\RemoveEmptyStaffContext` is to make ossia sections, i.e., alternative melodies on a separate piece of staff, with help of a Frenched staff. See Section 1.6.1.2 [Staff symbol], page 95.

You can make the staff lines invisible by removing the `Staff_symbol_engraver` from the `Staff` context.

```
\score {
  \context Staff \relative c'' { c8 c c16 c c c }
  \layout{
    \context {
      \Staff
      \remove Staff_symbol_engraver
    }
```

```
  }
}
```



See also

Notation Reference: Section 1.6.1.2 [Staff symbol], page 95.

Snippets: Staff-notation

Internals Reference: `ChordNames`, `FiguredBass`, `Lyrics`, `Staff`, `VerticalAxisGroup`.

1.6.2 Writing parts

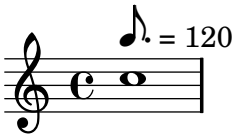
1.6.2.1 Metronome marks

Metronome settings can be entered as follows

```
\tempo duration = per-minute
```

In the MIDI output, they are interpreted as a tempo change. In the layout output, a metronome marking is printed

```
\tempo 8.=120 c''1
```



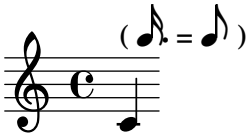
Selected Snippets

To change the tempo in the MIDI output without printing anything, make the metronome marking invisible

```
\once \override
Score.MetronomeMark #'transparent = ##t
```

To print other metronome markings, use these markup commands

```
c4^\markup {
  (
  \smaller \general-align #Y #DOWN \note #"16." #1
  =
  \smaller \general-align #Y #DOWN \note #"8" #1
  ) }
```



For more details, see Section 1.8.2 [Text markup], page 119.

See also

Music Glossary: ⟨undefined⟩ [metronome], page ⟨undefined⟩, ⟨undefined⟩ [metronomic indication], page ⟨undefined⟩, ⟨undefined⟩ [tempo indication], page ⟨undefined⟩, ⟨undefined⟩ [metronome mark], page ⟨undefined⟩.

Notation Reference: Section 1.8.2 [Text markup], page 119.

Snippets: Staff-notation .

Internals Reference: `MetronomeMark`, Section 4.2 [MIDI output], page 204.

Known issues and warnings

Collisions are not checked. If you have notes above the top line of the staff (or notes with articulations, slurs, text, etc), then the metronome marking may be printed on top of musical symbols. If this occurs, increase the padding of the metronome mark to place it further away from the staff.

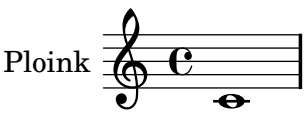
```
\override Score.MetronomeMark #'padding = #2.5
```

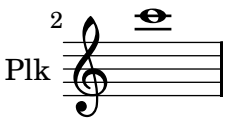
1.6.2.2 Instrument names

In an orchestral score, instrument names are printed at the left side of the staves.

This can be achieved by setting `Staff.instrumentName` and `Staff.shortInstrumentName`, or `PianoStaff.instrumentName` and `PianoStaff.shortInstrumentName`. This will print text before the start of the staff. For the first staff, `instrumentName` is used. If set, `shortInstrumentName` is used for the following staves.

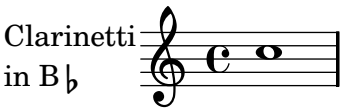
```
\set Staff.instrumentName = "Ploink "
\set Staff.shortInstrumentName = "Plk "
c1
\break
c''
```





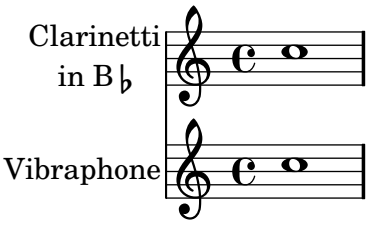
You can also use markup texts to construct more complicated instrument names, for example

```
\set Staff.instrumentName = \markup {
  \column { "Clarinetti"
            \line { "in B" \smaller \flat } } }
c''1
```



As instrument names are centered by default, multi line names are better entered using `\center-align`:

```
{ <<
\new Staff {
  \set Staff.instrumentName = \markup \center-align {
    Clarinetti
    \line { "in B" \smaller \flat }
  }
  c''1
}
\new Staff {
  \set Staff.instrumentName = "Vibraphone"
  c''1
}
>>
}
```



For longer instrument names, it may be useful to increase the `indent` setting in the `\layout` block.

Short instrument names, printed before the systems following the first one, are also centered by default, in a space which width is given by the `short-indent` variable of the `\layout` block.

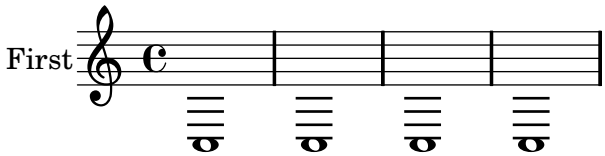
To add instrument names to other contexts (such as `GrandStaff`, `ChoirStaff`, or `StaffGroup`), the engraver must be added to that context.

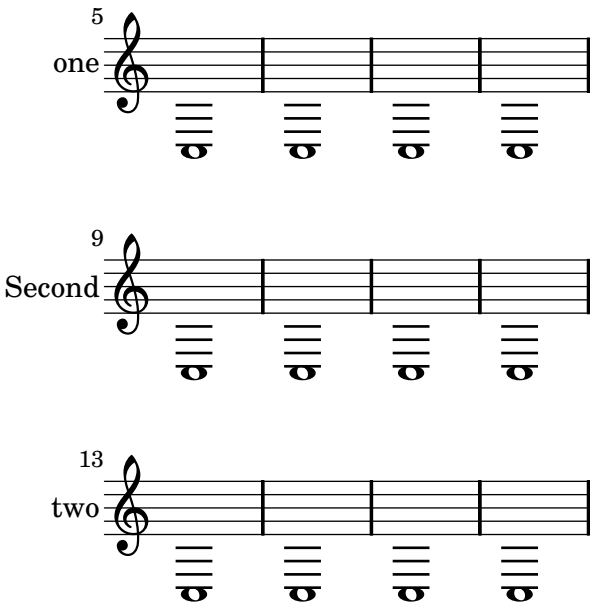
```
\layout{
  \context {\GrandStaff \consists "Instrument_name_engraver"}
}
```

More information about adding and removing engravers can be found in Section 6.1.4 [Modifying context plug-ins], page 253.

Instrument names may be changed in the middle of a piece,

```
\set Staff.instrumentName = "First"
\set Staff.shortInstrumentName = "one"
c1 c c c \break
c1 c c c \break
\set Staff.instrumentName = "Second"
\set Staff.shortInstrumentName = "two"
c1 c c c \break
c1 c c c \break
```





See also

Notation Reference: Section 6.1.4 [Modifying context plug-ins], page 253.

Snippets: Staff-notation

Internals Reference: `InstrumentName`, `PianoStaff`, `Staff`.

1.6.2.3 Quoting other voices

With quotations, fragments of other parts can be inserted into a part directly. Before a part can be quoted, it must be marked especially as quotable. This is done with the `\addQuote` command.

```
\addQuote name music
```

Here, *name* is an identifying string. The *music* is any kind of music. Here is an example of `\addQuote`

```
\addQuote clarinet \relative c' {
  f4 fis g gis
}
```

This command must be entered at toplevel, i.e., outside any music blocks. Typically, one would use an already defined music event as the *music*:

```
clarinet = \relative c' {
  f4 fis g gis
}
\addQuote clarinet { \clarinet }
```

After calling `\addQuote`, the quotation may then be done with `\quoteDuring` or `\cueDuring`,

```
\quoteDuring #name music
```

During a part, a piece of music can be quoted with the `\quoteDuring` command.

```
\quoteDuring #"clarinet" { s2. }
```

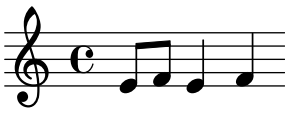
This would cite three quarter notes (the duration of `s2.`) of the previously added `clarinet` voice.

More precisely, it takes the current time-step of the part being printed, and extracts the notes at the corresponding point of the `\addQuote`d voice. Therefore, the argument to `\addQuote` should be the entire part of the voice to be quoted, including any rests at the beginning.

It is possible to use another music expression instead of `s`, thus creating a polyphonic section, but this may not always give the desired result.

Quotations take into account the transposition of both source and target instruments, if they are specified using the `\transposition` command.

```
\addQuote clarinet \relative c' {
  \transposition bes
  f4 fis g gis
}

{
  e'8 f'8 \quoteDuring #"clarinet" { s2 }
}
```



The type of events that are present in the quoted music can be trimmed with the `quotedEventTypes` property. The default value is `(note-event rest-event)`, which means that only notes and rests of the quoted voice end up in the `\quoteDuring`. Setting

```
\set Staff.quotedEventTypes =
       #'(note-event articulation-event dynamic-event)
```

will quote notes (but no rests), together with scripts and dynamics.

Known issues and warnings

Only the contents of the first `Voice` occurring in an `\addQuote` command will be considered for quotation, so *music* can not contain `\new` and `\context Voice` statements that would switch to a different Voice.

Quoting grace notes is broken and can even cause LilyPond to crash.

Quoting nested triplets may result in poor notation.

In earlier versions of LilyPond (pre 2.11), `addQuote` was written entirely in lower-case letters: `\addquote`.

See also

Notation Reference: Section 1.1.3.4 [Instrument transpositions], page 17.

Snippets: Staff-notation

Internals Reference: `QuoteMusic`, `Voice`.

1.6.2.4 Formatting cue notes

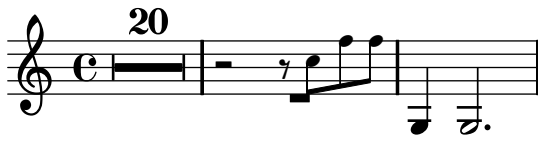
The previous section deals with inserting notes from another voice. There is a more advanced music function called `\cueDuring`, which makes formatting cue notes easier.

The syntax is

```
\cueDuring #name #updown music
```

This will insert notes from the part *name* into a `Voice` called `cue`. This happens simultaneously with *music*, which usually is a rest. When the cue notes start, the staff in effect becomes polyphonic for a moment. The argument *updown* determines whether the cue notes should be notated as a first or second voice.

```
smaller = {
  \set fontSize = #-2
  \override Stem #'length-fraction = #0.8
  \override Beam #'thickness = #0.384
  \override Beam #'length-fraction = #0.8
}

\addQuote clarinet \relative {
  R1*20
  r2 r8 c' f f
}

\new Staff \relative  <<

  % setup a context for cue notes.
  \new Voice = "cue" { \smaller \skip 1*21 }

  \set Score.skipBars = ##t

  \new Voice {
    R1*20
    \cueDuring #"clarinet" #UP {
      R1
    }
    g4 g2.
  }
>>
```



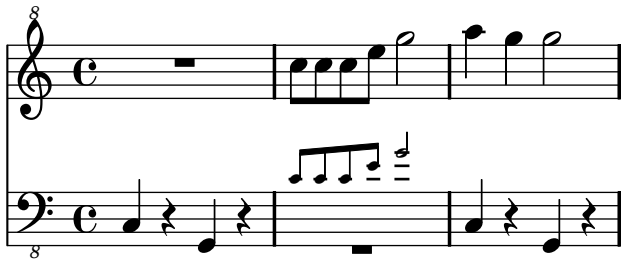
Here are a couple of hints for successful cue notes:

- Cue notes have smaller font sizes.
- The cued part is marked with the instrument playing the cue.
- When the original part takes over again, this should be marked with the name of the original instrument.
- Any other changes introduced by the cued part should also be undone. For example, if the cued instrument plays in a different clef, the original clef should be stated once again.

The macro `\transposedCueDuring` is useful to add cues to instruments which use a completely different octave range (for example, having a cue of a piccolo flute within a contra bassoon part).

```
picc = \relative c''' {
  \clef "treble^8"
```

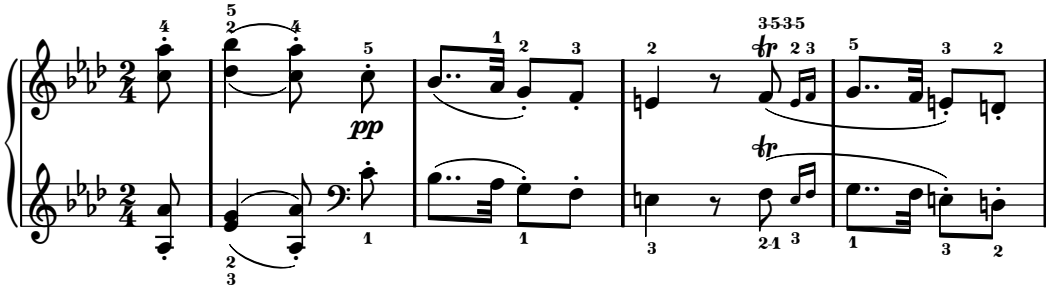
```
  R1 |
  c8 c c e g2 |
  a4 g g2 |
}
\addQuote "picc" { \picc }

cbsn = \relative c, {
  \clef "bass_8"
  c4 r g r
  \transposedCueDuring #"picc" #UP c,, { R1 } |
  c4 r g r |
}

<<
  \context Staff = "picc" \picc
  \context Staff = "cbsn" \cbsn
>>
```

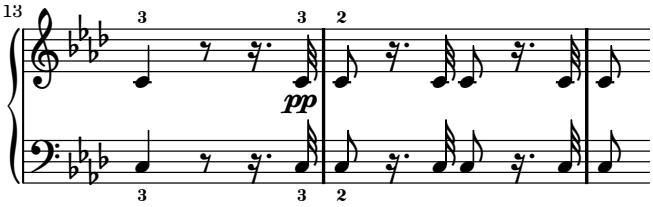


See also

Snippets: Staff-notation .
Internals Reference: `Voice`.

1.7 Editorial annotations



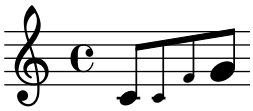


1.7.1 Inside the staff

1.7.1.1 Selecting notation font size

The easiest method of setting the font size of any context is by setting the `fontSize` property.

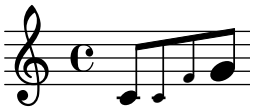
```
c8
\set fontSize = #-4
c f
\set fontSize = #3
g
```



It does not change the size of variable symbols, such as beams or slurs.

Internally, the `fontSize` context property will cause the `font-size` property to be set in all layout objects. The value of `font-size` is a number indicating the size relative to the standard size for the current staff height. Each step up is an increase of approximately 12% of the font size. Six steps is exactly a factor two. The Scheme function `magstep` converts a `font-size` number to a scaling factor. The `font-size` property can also be set directly, so that only certain layout objects are affected.

```
c8
\override NoteHead #'font-size = #-4
c f
\override NoteHead #'font-size = #3
g
```



Font size changes are achieved by scaling the design size that is closest to the desired size. The standard font size (for `font-size` equals 0), depends on the standard staff height. For a 20pt staff, a 10pt font is selected.

The `font-size` property can only be set on layout objects that use fonts. These are the ones supporting the `font-interface` layout interface.

Predefined commands

The following commands set `fontSize` for the current voice:

`\tiny`, `\small`, `\normalsize`.

See also

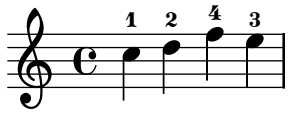
Snippets: Editorial-annotations .

Internals Reference: `font-interface`.

1.7.1.2 Fingering instructions

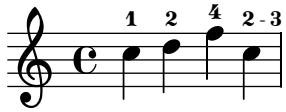
Fingering instructions can be entered using *note-digit*:

```
c4-1 d-2 f-4 e-3
```



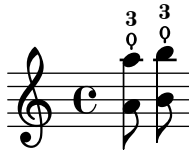
Use markup texts for finger changes.

```
c4-1 d-2 f-4 c^\markup { \finger "2 - 3" }
```



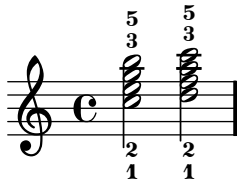
You can use the thumb-script (e.g., in cello music)to indicate that a note should be played with the thumb.

```
<a_\thumb a'-3>8 <b_\thumb b'-3>
```



Fingerings for chords can also be added to individual notes of the chord by adding them after the pitches.

```
< c-1 e-2 g-3 b-5 >2 < d-1 f-2 a-3 c-5 >
```

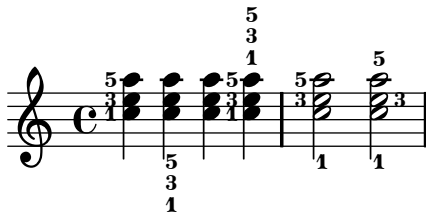


Selected Snippets

`fingeringOrientations` are used to control where the fingering numbers are placed.

```
\set fingeringOrientations = #'(left)
<c-1 e-3 a-5 > 4
\set fingeringOrientations = #'(down)
<c-1 e-3 a-5 >
\set fingeringOrientations = #'(right)
<c-1 e-3 a-5 >
\set fingeringOrientations = #'(up)
<c-1 e-3 a-5 >
\set fingeringOrientations = #'(left down)
```

```
<c-1 e-3 a-5 > 2
\set fingeringOrientations = #'(up right down)
<c-1 e-3 a-5 >
```



See also

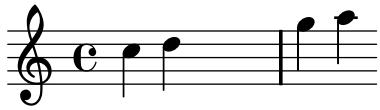
Snippets: Editorial-annotations .

Internals Reference: `Fingering`.

1.7.1.3 Hidden notes

Hidden (or invisible or transparent) notes can be useful in preparing theory or composition exercises.

```
c4 d4
\hideNotes
e4 f4
\unHideNotes
g4 a
```



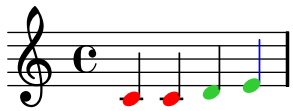
See also

Snippets: Editorial-annotations .

1.7.1.4 Coloring objects

Individual objects may be assigned colors. You may use the color names listed in the Section B.3 [List of colors], page 288.

```
\override NoteHead #'color = #red
c4 c
\override NoteHead #'color = #(x11-color 'LimeGreen)
d
\override Stem #'color = #blue
e
```



The full range of colors defined for X11 can be accessed by using the Scheme function x11-color. The function takes one argument that can be a symbol

```
\override Beam #'color = #(x11-color 'MediumTurquoise)
```

or a string

```
\override Beam #'color = #(x11-color "MediumTurquoise")
```

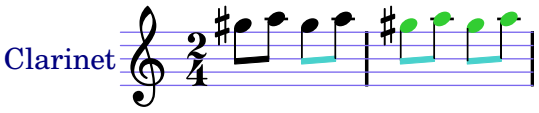
The first form is quicker to write and is more efficient. However, using the second form it is possible to access X11 colors by the multi-word form of its name

```
\override Beam #'color = #(x11-color "medium turquoise")
```

If x11-color cannot make sense of the parameter then the color returned defaults to black. It should be obvious from the final score that something is wrong.

This example illustrates the use of x11-color. Notice that the stem color remains black after being set to (x11-color 'Boggle), which is deliberate nonsense.

```
{
  \override Staff.StaffSymbol #'color = #(x11-color 'SlateBlue2)
  \set Staff.instrumentName = \markup {
    \with-color #(x11-color 'navy) "Clarinet"
  }
  \time 2/4
  gis''8 a''
  \override Beam #'color = #(x11-color "medium turquoise")
  gis'' a''
  \override NoteHead #'color = #(x11-color "LimeGreen")
  gis'' a''
  \override Stem #'color = #(x11-color 'Boggle)
  gis'' a''
}
```



TODO you can get exact RGB colors by specifying

% black \override Stem #'color = #(rgb-color 0 0 0) % white \override Stem #'color = #(rgb-color 1 1 1) % dark blue \override Stem #'color = #(rgb-color 0 0 0.5)

See also

Notation Reference: Section B.3 [List of colors], page 288, Section 6.2.5 [Objects connected to the input], page 262.

Snippets: Editorial-annotations .

Known issues and warnings

Not all x11 colors are distinguishable in a web browser. For web use normal colors are recommended.

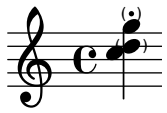
An x11 color is not necessarily exactly the same shade as a similarly named normal color.

Notes in a chord cannot be colored with `\override`; use `\tweak` instead. See Section 6.2.5 [Objects connected to the input], page 262, for details.

1.7.1.5 Parentheses

Objects may be parenthesized by prefixing `\parenthesize` to the music event,

```
<
  c
  \parenthesize d
  g
>4-\parenthesize -.
```



This only functions inside chords, to parenthesize a single note it must be enclosed with `<>` as if it is a chord.

```
< \parenthesize NOTE>
```

See also

Snippets: Editorial-annotations .

1.7.1.6 Stems

Whenever a note is found, a `Stem` object is created automatically. For whole notes and rests, they are also created but made invisible.

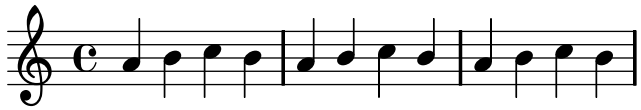
Predefined commands

`\stemUp`, `\stemDown`, `\stemNeutral`.

Selected Snippets

To change the direction of stems on the center line of the staff, use

```
a4 b c b
\override Stem #'neutral-direction = #up
a4 b c b
\override Stem #'neutral-direction = #down
a4 b c b
```



See also

Snippets: Editorial-annotations .

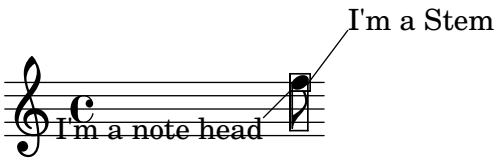
1.7.2 Outside the staff

1.7.2.1 Balloon help

Elements of notation can be marked and named with the help of a square balloon. The primary purpose of this feature is to explain notation.

The following example demonstrates its use.

```
\new Voice \with { \consists "Balloon_engraver" }
{
  \balloonGrobText #'Stem #'(3 . 4) \markup { "I'm a Stem" }
  <f-\balloonText #'(-2 . -2) \markup { "I'm a note head" }  >8
}
```



There are two music functions, `balloonGrobText` and `balloonText`; the former takes the name of the grob to adorn, while the latter may be used as an articulation on a note. The other arguments are the offset and the text of the label.

See also

Snippets: Editorial-annotations .

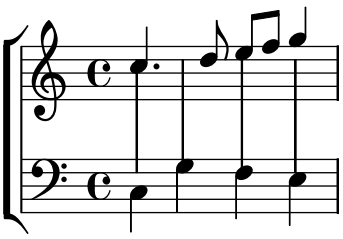
Internals Reference: `text-balloon-interface`.

1.7.2.2 Grid lines

Vertical lines can be drawn between staves synchronized with the notes.

```
\layout {
  \context {
    \Staff
    \consists "Grid_point_engraver" %% sets of grid
    gridInterval = #(ly:make-moment 1 4)
  }
}

\new Score \with {
  \consists "Grid_line_span_engraver"
  %% centers grid lines  horizontally below note heads
  \override NoteColumn #'X-offset = #-0.5
}

\new ChoirStaff <<
  \new Staff {
    \stemUp
    \relative {
      c'4. d8 e8 f g4
    }
```

```
  }
  \new Staff {
    %% centers grid lines  vertically
    \override Score.GridLine #'extra-offset = #'( 0.0 . 1.0 )
    \stemDown
    \clef bass
    \relative c {
      c4  g'  f  e
    }
  }
>>
```



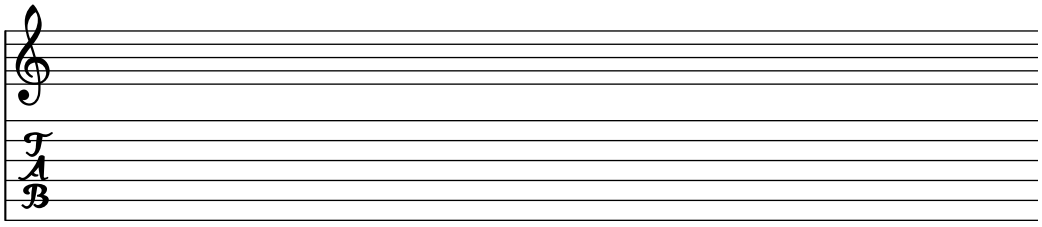
See also

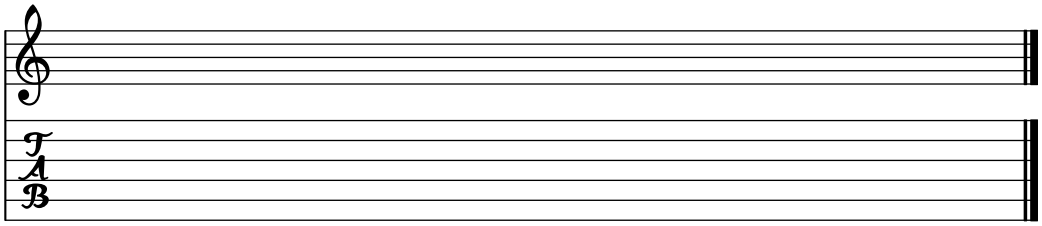
Snippets: Editorial-annotations .

1.7.2.3 Blank music sheet

A blank music sheet can be produced also by using invisible notes, and removing `Bar_number_engraver`.

```
\layout{ indent = #0 }
emptymusic = {
  \repeat unfold 2 % Change this for more lines.
  { s1\break }
  \bar "|."
}
\new Score \with {
  \override TimeSignature #'transparent = ##t
% un-comment this line if desired
%  \override Clef #'transparent = ##t
  defaultBarType = #""
  \remove Bar_number_engraver
} <<

% modify these to get the staves you want
  \new Staff \emptymusic
  \new TabStaff \emptymusic
>>
```





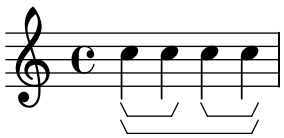
See also

Snippets: Editorial-annotations .

1.7.2.4 Analysis brackets

Brackets are used in musical analysis to indicate structure in musical pieces. LilyPond supports a simple form of nested horizontal brackets. To use this, add the `Horizontal_bracket_engraver` to the `Staff` context. A bracket is started with `\startGroup` and closed with `\stopGroup`.

```
\score {
  \relative c'' {
    c4\startGroup\startGroup
    c4\stopGroup
    c4\startGroup
    c4\stopGroup\stopGroup
  }
  \layout {
    \context {
      \Staff \consists "Horizontal_bracket_engraver"
}}}
```

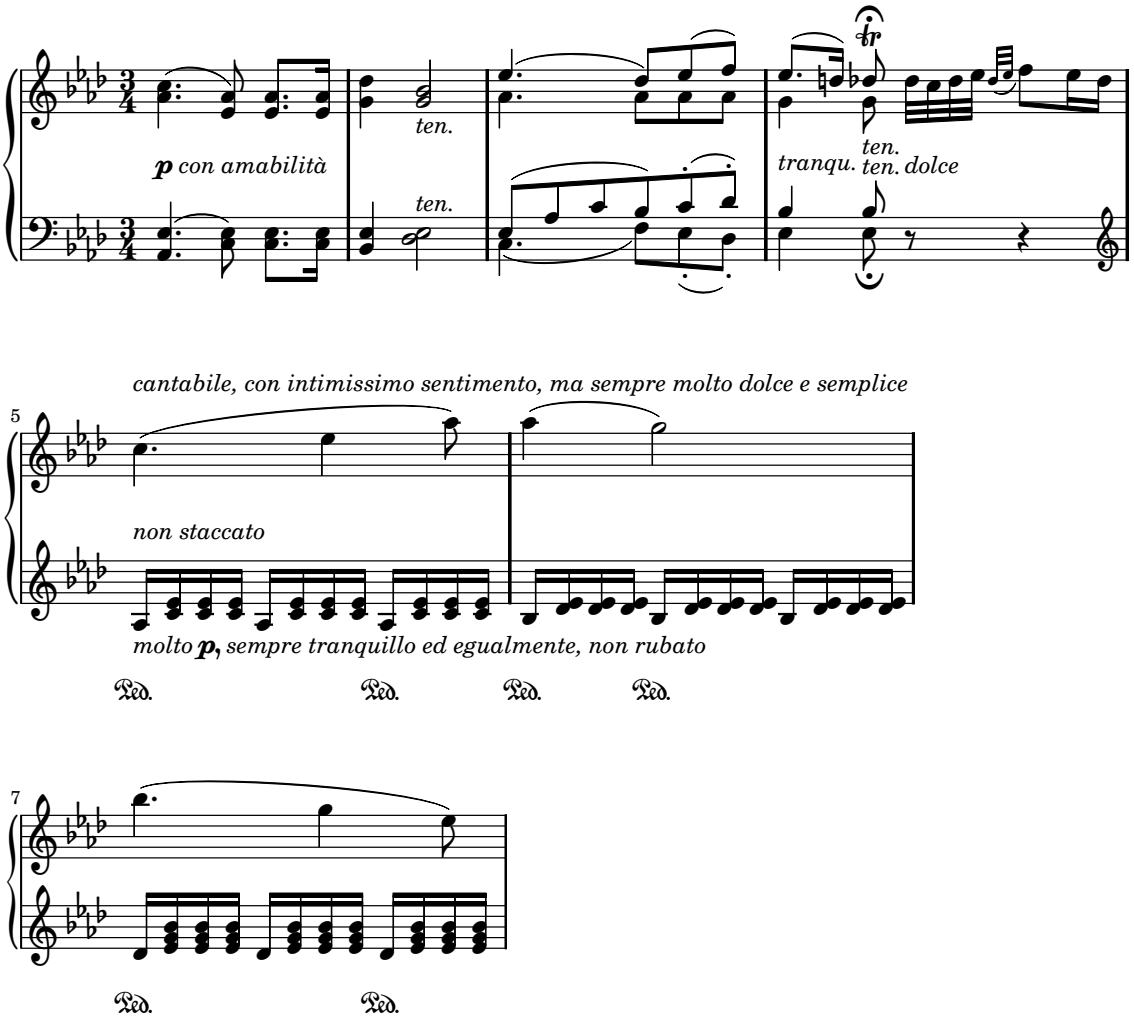


See also

Snippets: Editorial-annotations .

Internals Reference: `HorizontalBracket`, `Horizontal_bracket_engraver`, `Staff`.

1.8 Text



This section explains how to include text (with various formatting) in your scores.

> **Note:** To write accented and special text (such as characters from other languages), simply insert the characters directly into the lilypond file. The file must be saved as UTF-8. For more information, see Section 3.1.6 [Text encoding], page 193.

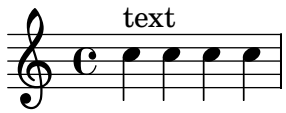
1.8.1 Writing text

1.8.1.1 Overview of text entry

There are four ways to add text to scores:

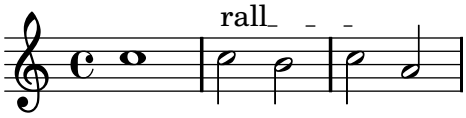
- Section 1.8.1.2 [Text scripts], page 114: blah blah

```
c4^"text" c c c
```



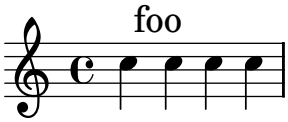
- Section 1.8.1.3 [Text spanners], page 115: blah blah

```
c1
\override TextSpanner #'bound-details #'left #'text =
  \markup { \upright "rall" }
c2\startTextSpan b c\stopTextSpan a
```



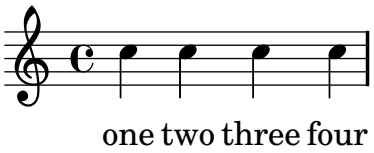
- Section 1.8.1.4 [Text marks], page 116: blah blah

```
c4\mark "foo" c c c
```



- Section 2.1 [Vocal music], page 129: blah blah, not in this section.

```
<<
  \relative c'' { c4 c c c }
  \addlyrics { one two three four }
>>
```



See also

Snippets: Text

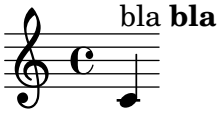
1.8.1.2 Text scripts

It is possible to place arbitrary strings of text or Section 1.8.2 [Text markup], page 119 with *note*`-"`*text*`"`.

INSERT EXAMPLE

More complex formatting may also be added to a note by using the `\markup` command, as described in Section 1.8.2 [Text markup], page 119.

```
c'4^\markup { bla \bold bla }
```



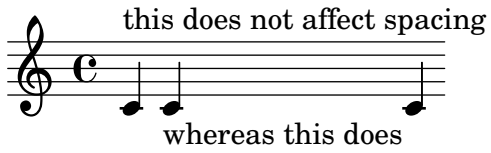
Predefined commands

`\textLengthOn`, `\textLengthOff`.

Selected Snippets

By default, these indications do not influence the note spacing. However, by using the command `\textLengthOn`, the widths will be taken into account:

```
c4^"this does not affect spacing" \textLengthOn c4_"whereas this does" c4
```



After having used such a command, to go back to the default behavior and prevent text from influencing spacing, use `\textLengthOff`.

Checking to make sure that text scripts and lyrics are within the margins is a relatively large computational task. To speed up processing, lilypond does not perform such calculations by default; to enable it, use

```
\override Score.PaperColumn #'keep-inside-line = ##t
```

See also

Notation Reference: Section 1.8.2 [Text markup], page 119.

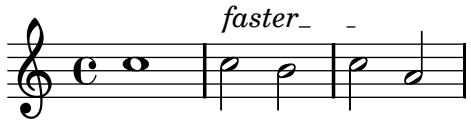
Snippets: Text

Internals Reference: `TextScript`.

1.8.1.3 Text spanners

Some performance indications, e.g., *rallentando* or *accelerando*, are written as text and are extended over many measures with dotted lines. Such texts are created using text spanners; attach `\startTextSpan` and `\stopTextSpan` to the first and last notes of the spanner.

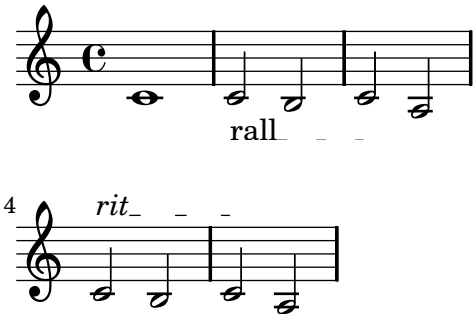
```
c1
\override TextSpanner #'bound-details #'left #'text = "faster"
c2\startTextSpan b c\stopTextSpan a
```



The string to be printed, as well as the style, is set through object properties. It can accept `\markup` blocks as well:

```
c1
\textSpannerDown
\override TextSpanner #'bound-details #'left #'text =
  \markup { \upright "rall" }
c2\startTextSpan b c\stopTextSpan a
```

```
\break
\textSpannerUp
\override TextSpanner #'bound-details #'left #'text =
  \markup { \italic "rit" }
c2\startTextSpan b c\stopTextSpan a
```



Predefined commands

`\textSpannerUp`, `\textSpannerDown`, `\textSpannerNeutral`.

Selected Snippets

To print a solid line, use

```
\override TextSpanner #'style = #'line
```

See also

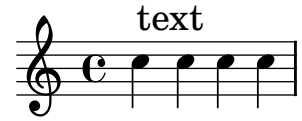
Snippets: Text

Internals Reference: `TextSpanner`.

1.8.1.4 Text marks

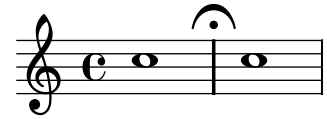
The `\mark` command is primarily used for Section 1.2.5.4 [Rehearsal marks], page 62, but it can also be used to add text elements in a score:

```
c4\mark "text" c c c
```



As it can contain a `\markup` object, the `\mark` command makes possible to put any text, but also signs like coda, segno, or fermata on a bar line. The appropriate symbol has to be specified in the `\markup` block; these symbols are listed in Section B.4 [The Feta font], page 290.

```
c1 \mark \markup { \musicglyph #"scripts.ufermata" }
c1
```



`\mark` is only typeset above the top staff of the score. If you specify the `\mark` command at a bar line, the resulting mark is placed above the bar line. If you specify it in the middle of a bar, the resulting mark is positioned between notes. If it is specified before the beginning of a score line, it is placed before the first note of the line. Finally, if the mark occurs at a line break, the mark will be printed at the beginning of the next line.

If there is no next line, then the mark will not be printed at all.

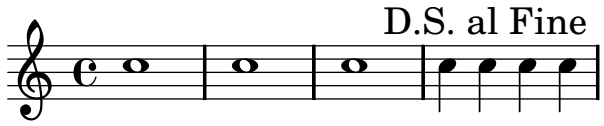
Selected Snippets

To print the mark at the end of the current line, use

```
\override Score.RehearsalMark
  #'break-visibility = #begin-of-line-invisible
```

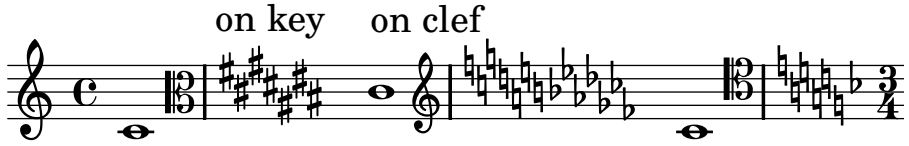
`\mark` is often useful for adding text to the end of bar. In such cases, changing the `#'self-alignment` is very useful

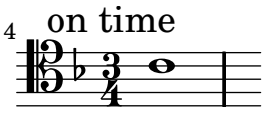
```
\override Score.RehearsalMark
  #'break-visibility = #begin-of-line-invisible
c1 c c c4 c c c
\once \override Score.RehearsalMark #'self-alignment-X = #right
\mark "D.S. al Fine "
```



Text marks may be aligned with notation objects other than bar lines,

```
\relative {
  c1
  \key cis \major
  \clef alto
  \override Score.RehearsalMark #'break-align-symbols = #'(key-signature)
  \mark "on key"
  cis
  \key ces \major
  \override Score.RehearsalMark #'break-align-symbols = #'(clef)
  \clef treble
  \mark "on clef"
  ces
  \override Score.RehearsalMark #'break-align-symbols = #'(time-signature)
  \key d \minor
  \clef tenor
  \time 3/4
  \mark "on time"
  c
}
```

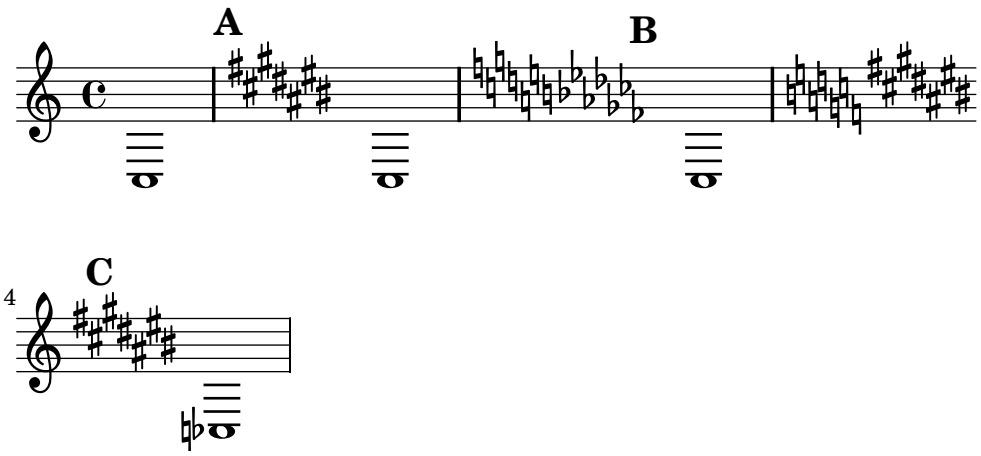




Possible symbols for the `break-align-symbols` list are `ambitus`, `breathing-sign`, `clef`, `custos`, `staff-bar`, `left-edge`, `key-cancellation`, `key-signature`, and `time-signature`.

The text marks will, by default, be aligned with the middle of the notation object, but this can be changed by overriding the `break-align-anchor-alignment` and `break-align-anchor` properties for the appropriate grob. For scores with multiple staves, this setting should be done for all the staves.

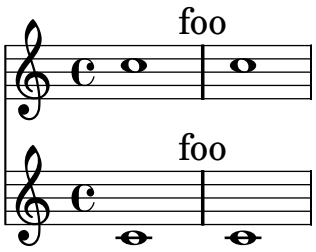
```
{
  \override Score.RehearsalMark #'break-align-symbols = #'(key-signature)
  c1
  \key cis \major

  % the RehearsalMark will be aligned with the left edge of the KeySignature
  \once \override Score.KeySignature #'break-align-anchor-alignment = #LEFT
  \mark \default
  cis1
  \key ces \major

  % the RehearsalMark will be aligned with the right edge of the KeySignature
  \once \override Score.KeySignature #'break-align-anchor-alignment = #RIGHT
  \mark \default
  ces1
  \key cis \major

  % the RehearsalMark will be aligned with the left edge of the KeySignature
  % and then shifted right by 2 units.
  \once \override Score.KeySignature #'break-align-anchor = #2
  \mark \default
  ces1
}
```



Although text marks are normally only printed above the topmost staff, you may alter this to print them on every staff,

```
{
  \new Score \with {
    \remove "Mark_engraver"
```

```
  }
  <<
    \new Staff \with {
      \consists "Mark_engraver"
    }
    { c''1 \mark "foo" c'' }
    \new Staff \with {
      \consists "Mark_engraver"
    }
    { c'1 \mark "foo" c' }
  >>
}
```



See also

Snippets: Text

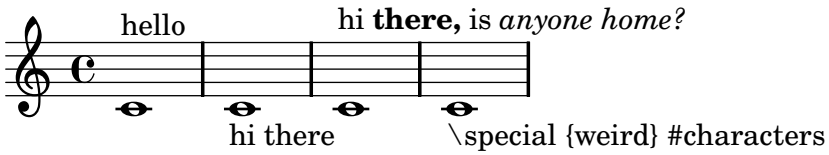
Internals Reference: `RehearsalMark`.

1.8.2 Text markup

1.8.2.1 Text markup introduction

Use `\markup` to typeset text. Commands are entered with the backslash `\`. To enter `\` and `#`, use double quotation marks.

```
c1^\markup { hello }
c1_\markup { hi there }
c1^\markup { hi \bold there, is \italic {anyone home?} }
c1_\markup { "\special {weird} #characters" }
```

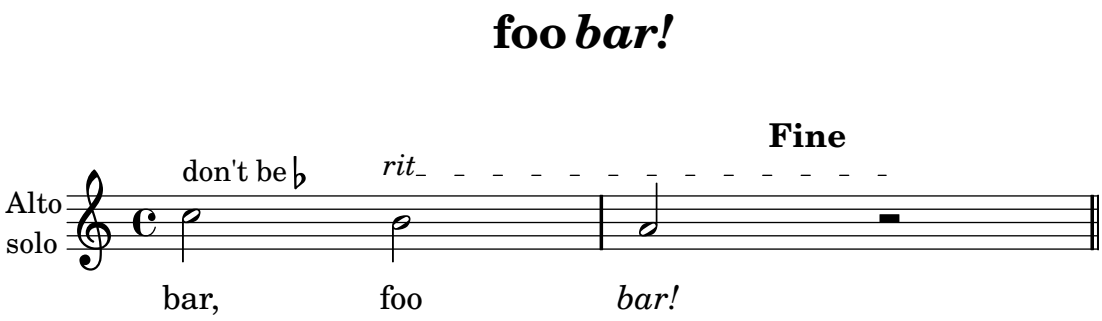


See Section B.6 [Overview of text markup commands], page 306, for a list of all commands.

`\markup` is primarily used for `TextScript`s, but it can also be used anywhere text is called in lilypond

```
\header{ title = \markup{ \bold { foo \italic { bar! } } } }
\score{
  \relative c'' {
```

```
    \override Score.RehearsalMark
      #'break-visibility = #begin-of-line-invisible
    \override Score.RehearsalMark #'self-alignment-X = #right

    \set Staff.instrumentName = \markup{ \column{ Alto solo } }
    c2^\markup{ don't be \flat }
    \override TextSpanner #'bound-details #'left #'text = \markup{\italic rit }
    b2\startTextSpan
    a2\mark \markup{ \large \bold Fine }
    r2\stopTextSpan
    \bar "||"
  }
  \addlyrics { bar, foo \markup{ \italic bar! } }
}
```



A `\markup` command can also be placed on its own, away from any `\score` block, see Section 3.1.3 [Multiple scores in a book], page 191.

```
\markup{ Here is some text. }
```

Here is some text.

The markup in the example demonstrates font switching commands. The command `\bold` and `\italic` apply to the first following word only; to apply a command to more than one word, enclose the words with braces,

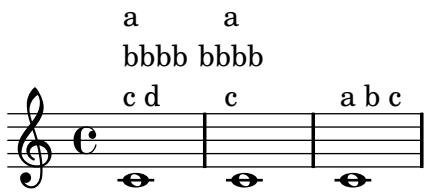
```
\markup { \bold { hi there } }
```

For clarity, you can also do this for single arguments, e.g.,

```
\markup { is \italic { anyone } home }
```

In markup mode you can compose expressions, similar to mathematical expressions, XML documents, and music expressions. You can stack expressions grouped vertically with the command `\column`. Similarly, `\center-align` aligns texts by their center lines:

```
c1^\markup { \column { a bbbb \line { c d } } }
c1^\markup { \center-align { a bbbb c } }
c1^\markup { \line { a b c } }
```



Lists with no previous command are not kept distinct. The expression

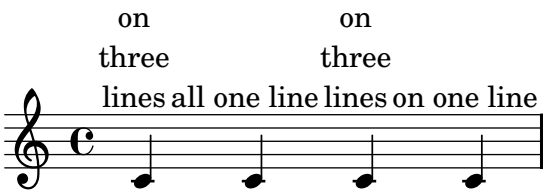
```
\center-align { { a b } { c d } }
```

is equivalent to

```
\center-align { a b c d }
```

To keep lists of words distinct, please use quotes " or the `\line` command

```
\textLengthOn
c4^\markup{ \center-align { on three lines } }
c4^\markup{ \center-align { "all one line" } }
c4^\markup{ \center-align { { on three lines } } }
c4^\markup{ \center-align { \line { on one line } } }
```



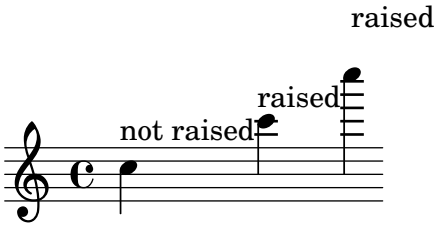
Markups can be stored in variables and these variables may be attached to notes, like

```
allegro = \markup { \bold \large { Allegro } }
 { a^\allegro b c d }
```

Some objects have alignment procedures of their own, which cancel out any effects of alignments applied to their markup arguments as a whole. For example, the `RehearsalMark` is horizontally centered, so using `\mark \markup { \left-align .. }` has no effect.

In addition, vertical placement is performed after creating the text markup object. If you wish to move an entire piece of markup, you need to use the #'padding property or create an 'anchor' point inside the markup (generally with `\hspace #0`).

```
\textLengthOn
c'4^\markup{ \raise #5 "not raised" }
\once \override TextScript #'padding = #3
c'4^\markup{ raised }
c'4^\markup{ \hspace #0 \raise #1.5 raised }
```



Some situations (such as dynamic marks) have preset font-related properties. If you are creating text in such situations, it is advisable to cancel those properties with `normal-text`. See Section B.6 [Overview of text markup commands], page 306, for more details.

See also

This manual: Section B.6 [Overview of text markup commands], page 306.

Snippets: Text

Internals Reference: `TextScript`.

Init files: '`scm/new-markup.scm`'.

Known issues and warnings

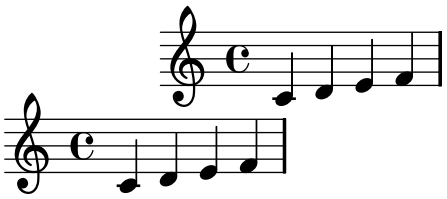
Kerning or generation of ligatures is only done when the TEX backend is used. In this case, LilyPond does not account for them so texts will be spaced slightly too wide.

Syntax errors for markup mode are confusing.

1.8.2.2 Nested scores

It is possible to nest music inside markups, by adding a `\score` block to a markup expression. Such a score must contain a `\layout` block.

```
\relative {
  c4 d^\markup {
    \score {
      \relative { c4 d e f }
      \layout { }
    }
  }
  e f
}
```



See also

Snippets: Text

1.8.2.3 Page wrapping text

Whereas `\markup` is used to enter a non-breakable block of text, `\markuplines` can be used at top-level to enter lines of text that can spread over multiple pages:

```
\markuplines {
  \justified-lines {
    A very long text of justified lines.
    ...
  }
  \justified-lines {
```

```
    An other very long paragraph.
    ...
  }
  ...
}
```

`\markuplines` accepts a list of markup, that is either the result of a markup list command, or a list of markups or of markup lists. The built-in markup list commands are described in Section B.7 [Overview of text markup list commands], page 314.

See also

This manual: Section B.7 [Overview of text markup list commands], page 314, Section 7.4.4 [New markup list command definition], page 282.

Snippets: Text

Predefined commands

`\markuplines`

1.8.2.4 Font selection

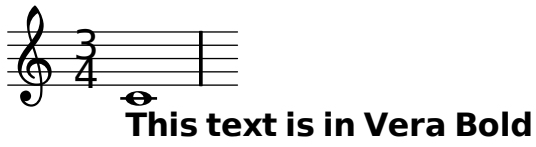
By setting the object properties described below, you can select a font from the preconfigured font families. LilyPond has default support for the feta music fonts. Text fonts are selected through Pango/FontConfig. The serif font defaults to New Century Schoolbook, the sans and typewriter to whatever the Pango installation defaults to.

- `font-encoding` is a symbol that sets layout of the glyphs. This should only be set to select different types of non-text fonts, e.g.

  `fetaBraces` for piano staff braces, `fetaMusic` the standard music font, including ancient glyphs, `fetaDynamic` for dynamic signs and `fetaNumber` for the number font.
- `font-family` is a symbol indicating the general class of the typeface. Supported are `roman` (Computer Modern), `sans`, and `typewriter`.
- `font-shape` is a symbol indicating the shape of the font. There are typically several font shapes available for each font family. Choices are `italic`, `caps`, and `upright`.
- `font-series` is a symbol indicating the series of the font. There are typically several font series for each font family and shape. Choices are `medium` and `bold`.

Fonts selected in the way sketched above come from a predefined style sheet. If you want to use a font from outside the style sheet, then set the `font-name` property,

```
{
  \override Staff.TimeSignature #'font-name = #"Charter"
  \override Staff.TimeSignature #'font-size = #2
  \time 3/4
  c'1_\markup {
    \override #'(font-name . "Vera Bold")
      { This text is in Vera Bold }
  }
}
```



Any font can be used, as long as it is available to Pango/FontConfig. To get a full list of all available fonts, run the command

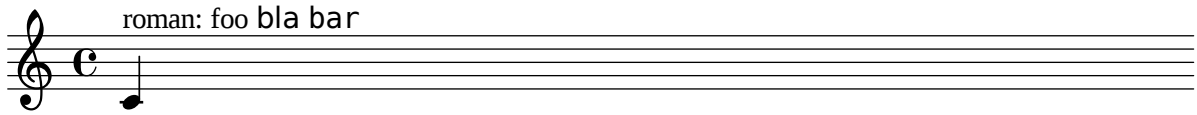
```
lilypond -dshow-available-fonts blabla
```

(the last argument of the command can be anything, but has to be present).

The size of the font may be set with the `font-size` property. The resulting size is taken relative to the `text-font-size` as defined in the `\paper` block.

It is also possible to change the default font family for the entire document. This is done by calling the `make-pango-font-tree` from within the `\paper` block. The function takes names for the font families to use for roman, sans serif and monospaced text. For example,

```
\paper  {
  myStaffSize = #20

  #(define fonts
    (make-pango-font-tree "Times New Roman"
                          "Nimbus Sans"
                          "Luxi Mono"
                           (/ myStaffSize 20)))
}

{
  c'^\markup { roman: foo \sans bla \typewriter bar }
}
```



See also

Snippets: Text

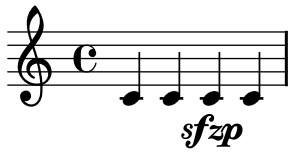
1.8.3 Special text concerns

1.8.3.1 New dynamic marks

It is possible to print new dynamic marks or text that should be aligned with dynamics. Use `make-dynamic-script` to create these marks. Note that the dynamic font only contains the characters `f`,`m`,`p`,`r`,`s` and `z`.

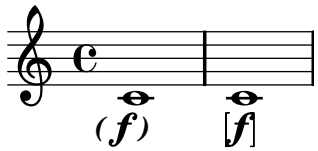
Some situations (such as dynamic marks) have preset font-related properties. If you are creating text in such situations, it is advisable to cancel those properties with `normal-text`. See Section B.6 [Overview of text markup commands], page 306, for more details.

```
sfzp = #(make-dynamic-script "sfzp")
\relative c' {
  c4 c c\sfzp c
}
```



It is also possible to print dynamics in round parenthesis or square brackets. These are often used for adding editorial dynamics.

```
rndf = \markup{ \center-align {\line { \bold{\italic (}
  \dynamic f \bold{\italic )} }} }
boxf = \markup{ \bracket { \dynamic f } }
{ c'1_\rndf c'1_\boxf }
```



See also

Snippets: Text

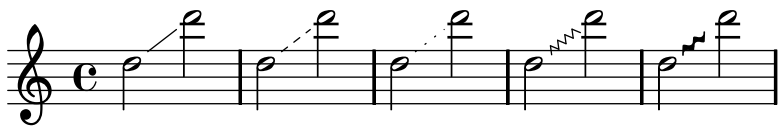
1.8.3.2 Text and line spanners

Some performance indications, e.g., *rallentando* and *accelerando* and *trills* are written as text and are extended over many measures with lines, sometimes dotted or wavy.

These all use the same routines as the glissando for drawing the texts and the lines, and tuning their behavior is therefore also done in the same way. It is done with a spanner, and the routine responsible for drawing the spanners is `ly:line-interface::print`. This routine determines the exact location of the two *span points* and draws a line in between, in the style requested.

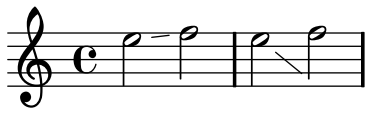
Here is an example of the different line styles available, and how to tune them.

```
d2 \glissando d'2
\once \override Glissando #'style = #'dashed-line
d,2 \glissando d'2
\override Glissando #'style = #'dotted-line
d,2 \glissando d'2
\override Glissando #'style = #'zigzag
d,2 \glissando d'2
\override Glissando #'style = #'trill
d,2 \glissando d'2
```



The information that determines the end-points is computed on-the-fly for every graphic object, but it is possible to override these.

```
e2 \glissando f
\once \override Glissando #'bound-details #'right #'Y = #-2
e2 \glissando f
```

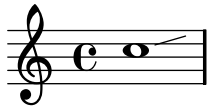


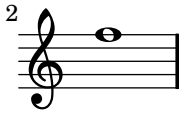
The `Glissando` object, like any other using the `ly:line-interface::print` routine, carries a nested association list. In the above statement, the value for `Y` is set to `-2` for the association list corresponding to the right end point. Of course, it is also possible to adjust the left side with `left` instead of `right`.

If `Y` is not set, the value is computed from the vertical position of right attachment point of the spanner.

In case of a line break, the values for the span-points are extended with contents of the `left-broken` and `right-broken` sublists, for example

```
\override Glissando #'breakable = ##T
\override Glissando #'bound-details #'right-broken #'Y = #-3
c1 \glissando \break
f1
```





The following properties can be used for the

`Y` This sets the Y-coordinate of the end point, in staff space. By default, it is the center of the bound object, so for a glissando it points to the vertical center of the note head.

For horizontal spanners, such as text spanner and trill spanners, it is hardcoded to 0.

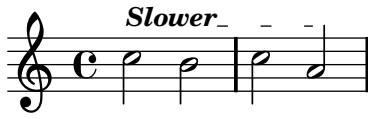
`attach-dir`
This determines where the line starts and ends in X-direction, relative to the bound object. So, a value of `-1` (or `LEFT`) makes the line start/end at the left side of the note head it is attached to.

`X` This is the absolute coordinate of the end point. It is usually computed on the fly, and there is little use in overriding it.

`stencil` Line spanners may have symbols at the beginning or end, which is contained in this sub-property. This is for internal use, it is recommended to use `text`.

`text` This is a markup that is evaluated to yield stencil. It is used to put *cresc.* and *tr* on horizontal spanners.

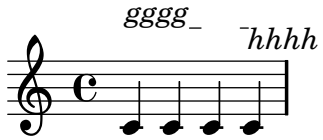
```
\override TextSpanner #'bound-details #'left #'text
   = \markup { \small \bold Slower }
c2\startTextSpan b c a\stopTextSpan
```



`stencil-align-dir-y`
`stencil-offset`

Without setting this, the stencil is simply put there at the end-point, as defined by the `X` and `Y` sub properties. Setting either `stencil-align-dir-y` or `stencil-offset` will move the symbol at the edge relative to the end point of the line

```
\override TextSpanner #'bound-details
  #'left #'stencil-align-dir-y = #DOWN
\override TextSpanner #'bound-details
  #'right #'stencil-align-dir-y = #UP

\override TextSpanner #'bound-details
  #'left #'text = #"gggg"
\override TextSpanner #'bound-details
  #'right #'text = #"hhhh"
c4^\startTextSpan c c c \stopTextSpan
```



`arrow` Setting this sub property to `#t` produce an arrowhead at the end of the line.

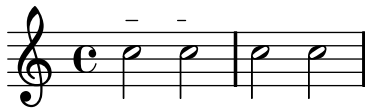
`padding` This sub property controls the space between the specified end-point of the line and the actual end. Without padding, a glissando would start and end in the center of each note head.

TODO: add this somewhere

```
\new Staff {
  \override TextSpanner #'bound-details #'left-broken #'text = ##f
   \override TextSpanner #'bound-details #'left #'text = \markup {
"start" }
   c'1 \startTextSpan \break
   c'1
   c'1 \stopTextSpan
}
```

The music function \endSpanners terminates spanners and hairpins after exactly one note.

```
\endSpanners
c2 \startTextSpan c2
c2 \< c2
```



When using \endSpanners it is not necessary to close \startTextSpan with \stopTextSpan, nor is it necessary to close hairpins with \!.

See also

Snippets: Text

Internals Reference: `TextSpanner`, `Glissando`, `VoiceFollower`, `TrillSpanner`, `line-spanner-interface`.

2 Specialist notation

This chapter explains how to create musical notation.

2.1 Vocal music

Since LilyPond input files are text, there are two issues to consider when working with vocal music:

- Song texts must be entered as text, not notes. For example, the input `d` should be interpreted as a one letter syllable, not the note D.
- Song texts must be aligned with the notes of their melody.

There are a few different ways to define lyrics; we shall begin by examining the simplest method, and gradually increase complexity.

Selected Snippets

Checking to make sure that text scripts and lyrics are within the margins is a relatively large computational task. To speed up processing, lilypond does not perform such calculations by default; to enable it, use

```
\override Score.PaperColumn #'keep-inside-line = ##t
```

To make lyrics avoid bar lines as well, use

```
\layout {
  \context {
    \Lyrics
      \consists "Bar_engraver"
      \consists "Separating_line_group_engraver"
      \override BarLine #'transparent = ##t
  }
}
```

2.1.1 Simple lyrics

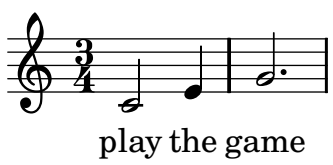
2.1.1.1 Setting simple songs

The easiest way to add lyrics to a melody is to append

```
\addlyrics { the lyrics }
```

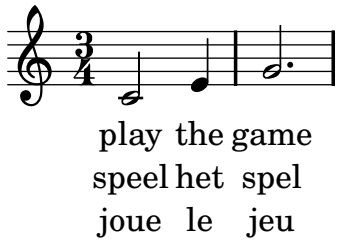
to a melody. Here is an example,

```
\time 3/4
\relative { c2 e4 g2. }
\addlyrics { play the game }
```



More stanzas can be added by adding more `\addlyrics` sections

```
\time 3/4
\relative { c2 e4 g2. }
\addlyrics { play the game }
\addlyrics { speel het spel }
\addlyrics { joue le jeu }
```



The command `\addlyrics` cannot handle polyphony settings. For these cases you should use `\lyricsto` and `\lyricmode`, as will be introduced in Section 2.1.1.2 [Entering lyrics], page 130.

2.1.1.2 Entering lyrics

Lyrics are entered in a special input mode, which can be introduced by the keyword `\lyricmode`, or by using `\addlyrics` or `\lyricsto`. In this mode you can enter lyrics, with punctuation and accents, and the input `d` is not parsed as a pitch, but rather as a one letter syllable. Syllables are entered like notes, but with pitches replaced by text. For example,

```
\lyricmode { Twin-4 kle4 twin- kle litt- le star2 }
```

There are two main methods to specify the horizontal placement of the syllables, either by specifying the duration of each syllable explicitly, like in the example above, or by automatically aligning the lyrics to a melody or other voice of music, using `\addlyrics` or `\lyricsto`.

A word or syllable of lyrics begins with an alphabetic character, and ends with any space or digit. The following characters can be any character that is not a digit or white space.

Any character that is not a digit or white space will be regarded as part of the syllable; one important consequence of this is that a word can end with `}`, which often leads to the following mistake:

```
\lyricmode { lah- lah}
```

In this example, the `}` is included in the final syllable, so the opening brace is not balanced and the input file will probably not compile.

Similarly, a period which follows an alphabetic sequence is included in the resulting string. As a consequence, spaces must be inserted around property commands: do *not* write

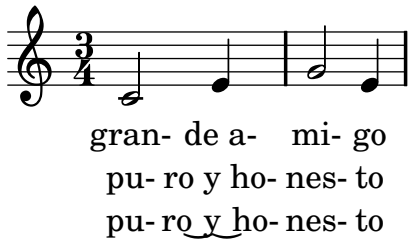
```
\override Score.LyricText #'font-shape = #'italic
```

but instead use

```
\override Score . LyricText #'font-shape = #'italic
```

In order to assign more than one syllable to a single note, you can surround them with quotes or use a `_` character, to get spaces between syllables, or use tilde symbol (`~`) to get a lyric tie.

```
\time 3/4
\relative { c2 e4 g2 e4 }
\addlyrics { gran- de_a- mi- go }
\addlyrics { pu- "ro y ho-" nes- to }
\addlyrics { pu- ro~y~ho- nes- to }
```

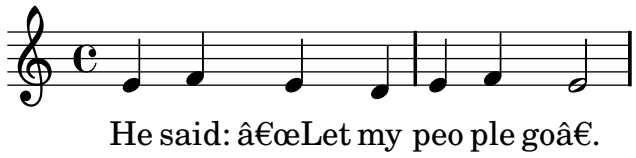


The lyric tie is implemented with the Unicode character U+203F, so be sure to have a font (Like DejaVuLGC) installed that includes this glyph.

To enter lyrics with characters from non-English languages, or with accented and special characters (such as the heart symbol or slanted quotes), simply insert the characters directly into the input file and save it with utf-8 encoding. See Section 3.1.6 [Text encoding], page 193, for more info.

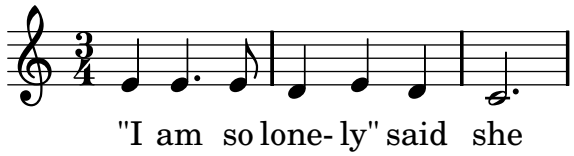
FIXME: quotes.

```
\relative { e4 f e d e f e2 }
\addlyrics { He said: â€œLet my peo ple goâ€. }
```



To use normal quotes in lyrics, add a backslash before the quotes. For example,

```
\relative c' { \time 3/4 e4 e4. e8 d4 e d c2. }
\addlyrics { "\"I" am so lone- "ly\"" said she }
```



The full definition of a word start in Lyrics mode is somewhat more complex.

A word in Lyrics mode begins with: an alphabetic character, `_`, `?`, `!`, `:`, `'`, the control characters `^A` through `^F`, `^Q` through `^W`, `^Y`, `^^`, any 8-bit character with ASCII code over 127, or a two-character combination of a backslash followed by one of `` ` ``, `'`, `"`, or `^`.

To define variables containing lyrics, the function `lyricmode` must be used.

```
verseOne = \lyricmode { Joy to the world the Lord is come }
\score {
  <<
    \new Voice = "one" \relative c'' {
      \autoBeamOff
      \time 2/4
      c4 b8. a16 g4. f8 e4 d c2
    }
    \addlyrics { \verseOne }
  >>
}
```

See also

Internals Reference: `LyricText`, `LyricSpace`.

2.1.2 Aligning lyrics to a melody

Lyrics are printed by interpreting them in the context called `Lyrics`.

```
\new Lyrics \lyricmode ...
```

There are two main methods to specify the horizontal placement of the syllables:

- by automatically aligning the lyrics to a melody or other voice of music, using `\addlyrics` or `\lyricsto`.
- or by specifying the duration of each syllable explicitly, using `\lyricmode`

2.1.2.1 Automatic syllable durations

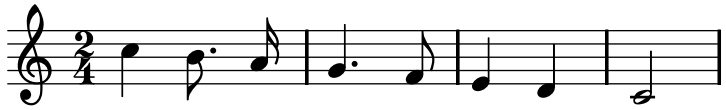
The lyrics can be aligned under a given melody automatically. This is achieved by combining the melody and the lyrics with the `\lyricsto` expression

```
\new Lyrics \lyricsto name ...
```

This aligns the lyrics to the notes of the `Voice` context called *name*, which must already exist. Therefore normally the `Voice` is specified first, and then the lyrics are specified with `\lyricsto`. The command `\lyricsto` switches to `\lyricmode` mode automatically, so the `\lyricmode` keyword may be omitted.

The following example uses different commands for entering lyrics.

```
<<
  \new Voice = "one" \relative c'' {
    \autoBeamOff
    \time 2/4
    c4 b8. a16 g4. f8 e4 d c2
  }
  \new Lyrics \lyricmode { Joy4 to8. the16 world!4. the8 Lord4 is come.2 }
  \new Lyrics \lyricmode { Joy to the earth! the Sa -- viour reigns. }
  \new Lyrics \lyricsto "one" { No more let sins and sor -- rows grow. }
>>
```



Joy to the world! the Lord is come.
Joy to the earth! the Sa - viour
No more let sins and sor-rows grow.

8

reigns.

The second stanza is not properly aligned because the durations were not specified. A solution for that would be to use `\lyricsto`.

The `\addlyrics` command is actually just a convenient way to write a more complicated LilyPond structure that sets up the lyrics.

```
{ MUSIC }
\addlyrics { LYRICS }
```

is the same as

```
\new Voice = "blah" { music }
\new Lyrics \lyricsto "blah" { LYRICS }
```

2.1.2.2 Another way of entering lyrics

Lyrics can also be entered without `\addlyrics` or `\lyricsto`. In this case, syllables are entered like notes – but with pitches replaced by text – and the duration of each syllable must be entered explicitly. For example:

```
play2 the4 game2.
sink2 or4 swim2.
```

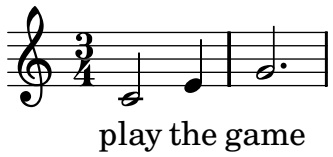
The alignment to a melody can be specified with the `associatedVoice` property,

```
\set associatedVoice = #"lala"
```

The value of the property (here: `"lala"`) should be the name of a `Voice` context. Without this setting, extender lines will not be formatted properly.

Here is an example demonstrating manual lyric durations,

```
<< \new Voice = "melody" {
    \time 3/4
    c2 e4 g2.
 }
 \new Lyrics \lyricmode {
   \set associatedVoice = #"melody"
   play2 the4 game2.
 } >>
```



See also

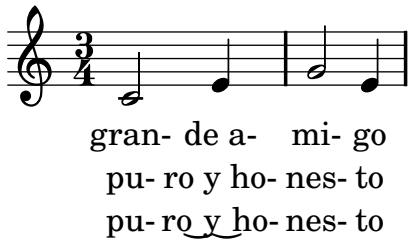
Internals Reference: `Lyrics`.

2.1.2.3 Assigning more than one syllable to a single note

In order to assign more than one syllable to a single note, you can surround them with quotes or use a `_` character, to get spaces between syllables, or use tilde symbol (`~`) to get a lyric tie[1].

```
\time 3/4
\relative { c2 e4 g2 e4 }
\addlyrics { gran- de_a- mi- go }
\addlyrics { pu- "ro y ho-" nes- to }
\addlyrics { pu- ro~y~ho- nes- to }
```

[1] The lyric ties is implemented with the Unicode character U+203F, so be sure to have a font (Like DejaVuLGC) installed that includes this glyph.



See also

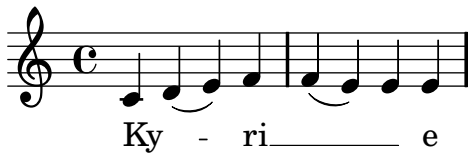
Internals Reference: `LyricCombineMusic`.

2.1.2.4 More than one note on a single syllable

Sometimes, particularly in Medieval music, several notes are to be sung on one single syllable; such vocalises are called melismas, or melismata.

You can define melismata entirely in the lyrics, by entering `_` for every note that is part of the melisma.

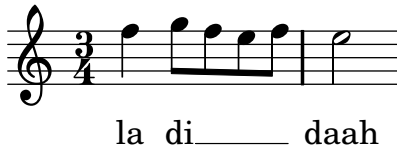
```
{ \set melismaBusyProperties = #'()
 c d( e) f f( e) e e  }
\addlyrics
 { Ky -- _ _ ri __ _ _ _  e }
```



In this case, you can also have ties and slurs in the melody if you set `melismaBusyProperties`, as is done in the example above.

However, the `\lyricsto` command can also detect melismata automatically: it only puts one syllable under a tied or slurred group of notes. If you want to force an unslurred group of notes to be a melisma, insert `\melisma` after the first note of the group, and `\melismaEnd` after the last one, e.g.,

```
<<
  \new Voice = "lala" {
    \time 3/4
    f4 g8
    \melisma
    f e f
    \melismaEnd
    e2
  }
  \new Lyrics \lyricsto "lala" {
    la di __ daah
  }
>>
```



In addition, notes are considered a melisma if they are manually beamed, and automatic beaming (see Section 1.2.4.2 [Setting automatic beam behavior], page 51) is switched off.

A complete example of a SATB score setup is in section learning manual, ⟨undefined⟩ [Vocal ensembles], page ⟨undefined⟩ .

Predefined commands

`\melisma`, `\melismaEnd`

See also

Internals Reference: `Melisma_translator`.

Known issues and warnings

Melismata are not detected automatically, and extender lines must be inserted by hand.

2.1.2.5 Extenders and hyphens

Melismata are indicated with a horizontal line centered between a syllable and the next one. Such a line is called an extender line, and it is entered as ' `__` ' (note the spaces before and after the two underscore characters).

Centered hyphens are entered as ' `--` ' between syllables of a same word (note the spaces before and after the two hyphen characters). The hyphen will be centered between the syllables, and its length will be adjusted depending on the space between the syllables.

In tighly engraved music, hyphens can be removed. Whether this happens can be controlled with the `minimum-distance` (minimum distance between two syllables) and the `minimum-length` (threshold below which hyphens are removed).

See also

Internals Reference: `LyricExtender`, `LyricHyphen`

2.1.3 Vocals and variables

2.1.3.1 Working with lyrics and variables

To define variables containing lyrics, the function `\lyricmode` must be used. You do not have to enter durations though, if you add `\addlyrics` or `\lyricsto` when invoking your variable.

```
verseOne = \lyricmode { Joy to the world the Lord is come }
\score {
 <<
   \new Voice = "one" \relative c'' {
     \autoBeamOff
     \time 2/4
     c4 b8. a16 g4. f8 e4 d c2
   }
   \addlyrics { \verseOne }
```

```
 >>
}
```

For different or more complex orderings, the best way is to setup the hierarchy of staves and lyrics first, e.g.,

```
\new ChoirStaff <<
  \new Voice = "soprano" { music }
  \new Lyrics = "sopranoLyrics" { s1 }
  \new Lyrics = "tenorLyrics" { s1 }
  \new Voice = "tenor" { music }
>>
```

and then combine the appropriate melodies and lyric lines

```
\context Lyrics = sopranoLyrics \lyricsto "soprano"
the lyrics
```

The final input would resemble

```
<<\new ChoirStaff << setup the music >>
 \lyricsto "soprano" etc
 \lyricsto "alto" etc
etc
>>
```

See also

Internals Reference: `LyricCombineMusic`, `Lyrics`.

2.1.4 Flexibility in placement

Often, different stanzas of one song are put to one melody in slightly differing ways. Such variations can still be captured with `\lyricsto`.

2.1.4.1 Lyrics to multiple notes of a melisma

One possibility is that the text has a melisma in one stanza, but multiple syllables in another one. One solution is to make the faster voice ignore the melisma. This is done by setting `ignoreMelismata` in the Lyrics context.

There is one tricky aspect: the setting for `ignoreMelismata` must be set one syllable *before* the non-melismatic syllable in the text, as shown here,

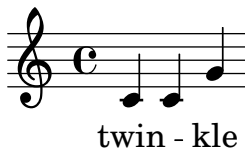
```
%{
<<
  \relative \new Voice = "lahlah" {
    \set Staff.autoBeaming = ##f
    c4
    \slurDotted
    f8.[( g16])
    a4
  }
  \new Lyrics \lyricsto "lahlah" {
    more slow -- ly
  }
  \new Lyrics \lyricsto "lahlah" {
    \set ignoreMelismata = ##t % applies to "fas"
```

```
    go fas -- ter
    \unset ignoreMelismata
    still
  }
>>
%}
```

The `ignoreMelismata` applies to the syllable ‘fas’, so it should be entered before ‘go’.

The reverse is also possible: making a lyric line slower than the standard. This can be achieved by insert `\skip`s into the lyrics. For every `\skip`, the text will be delayed another note. For example,

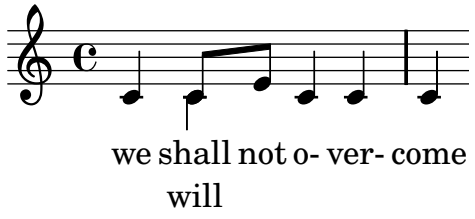
```
\relative { c c g' }
\addlyrics {
  twin -- \skip 4
  kle
}
```



2.1.4.2 Divisi lyrics

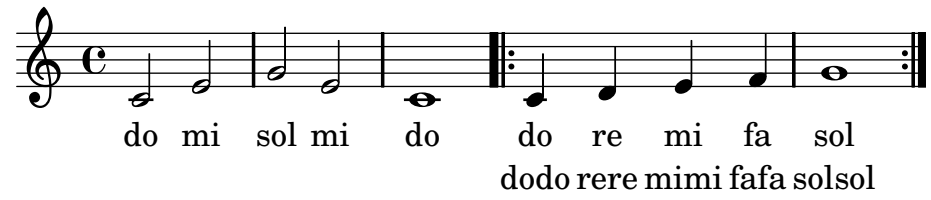
You can display alternate (or divisi) lyrics by naming voice contexts and attaching lyrics to those specific contexts.

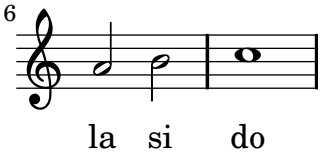
```
\score{ <<
  \new Voice = "melody" {
    \relative c' {
      c4
      <<
        { \voiceOne c8 e }
        \new Voice = "splitpart" { \voiceTwo c4 }
      >>
      \oneVoice c4 c | c
    }
  }
  \new Lyrics \lyricsto "melody" { we shall not o- ver- come }
  \new Lyrics \lyricsto "splitpart" { will }
>> }
```



You can use this trick to display different lyrics for a repeated section.

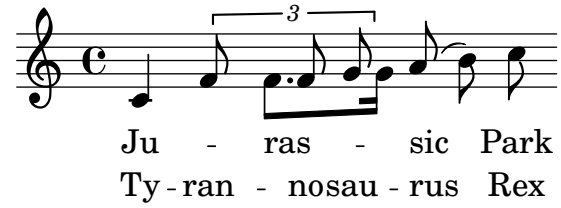
```
\score{ <<
  \new Voice = "melody" \relative c' {
    c2 e | g e | c1 |
    \new Voice = "verse" \repeat volta 2 {c4 d e f | g1 | }
    a2 b | c1}
  \new Lyrics = "mainlyrics" \lyricsto melody \lyricmode {
    do mi sol mi do
    la si do }
  \context Lyrics = "mainlyrics" \lyricsto verse \lyricmode {
   do re mi fa sol }
  \new Lyrics = "repeatlyrics" \lyricsto verse \lyricmode {
   dodo rere mimi fafa solsol }
>>
}
```





2.1.4.3 Switching the melody associated with a lyrics line

More complex variations in text underlay are possible. It is possible to switch the melody for a line of lyrics during the text. This is done by setting the `associatedVoice` property. In the example



the text for the first stanza is set to a melody called ‘lahlah’,

```
\new Lyrics \lyricsto "lahlah" {
  Ju -- ras -- sic Park
}
```

The second stanza initially is set to the `lahlah` context, but for the syllable ‘ran’, it switches to a different melody. This is achieved with

```
\set associatedVoice = alternative
```

Here, `alternative` is the name of the `Voice` context containing the triplet.

Again, the command must be one syllable too early, before ‘Ty’ in this case.

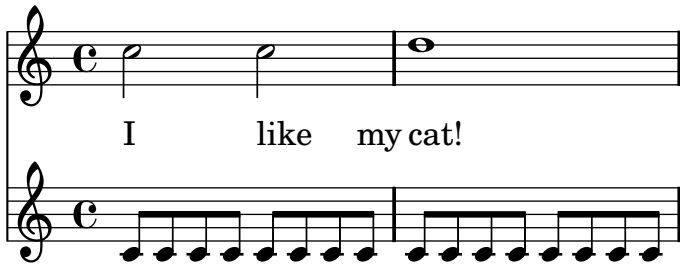
```
\new Lyrics \lyricsto "lahlah" {
  \set associatedVoice = alternative % applies to "ran"
  Ty --
  ran --
  no --
  \set associatedVoice = lahlah % applies to "rus"
  sau -- rus Rex
}
```

The underlay is switched back to the starting situation by assigning `lahlah` to `associatedVoice`.

2.1.4.4 Lyrics independent of notes

In some complex vocal music, it may be desirable to place lyrics completely independently of notes. Music defined inside `lyricrhythm` disappears into the `Devnull` context, but the rhythms can still be used to place the lyrics.

```
voice = {
  c''2
  \tag #'music { c''2 }
  \tag #'lyricrhythm { c''4. c''8 }
  d''1
}

lyr = \lyricmode { I like my cat! }

<<
  \new Staff \keepWithTag #'music \voice
  \new Devnull="nowhere" \keepWithTag #'lyricrhythm \voice
  \new Lyrics \lyricsto "nowhere" \lyr
  \new Staff { c'8 c' c' c' c' c' c' c'
  c' c' c' c' c' c' c' c' }
>>
```



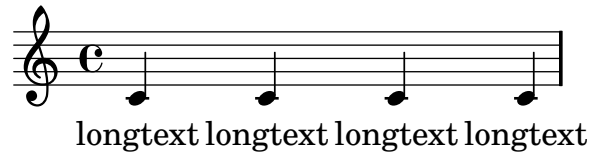
2.1.5 Spacing vocals

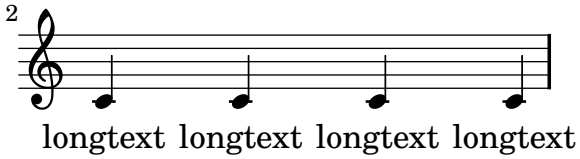
2.1.5.1 Spacing lyrics

To increase the spacing between lyrics, set the minimum-distance property of LyricSpace.

```
{
  c c c c
  \override Lyrics.LyricSpace #'minimum-distance = #1.0
  c c c c
}
\addlyrics {
  longtext longtext longtext longtext
```

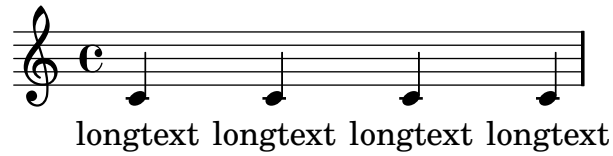
```
  longtext longtext longtext longtext
}
```

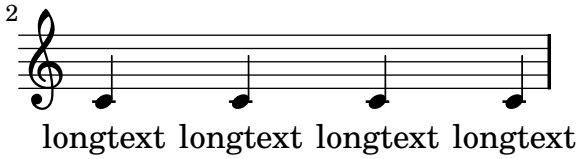




To make this change for all lyrics in the score, set the property in the layout.

```
\score {
  \relative c' {
  c c c c
  c c c c
  }
  \addlyrics {
  longtext longtext longtext longtext
  longtext longtext longtext longtext
  }
  \layout {
    \context {
      \Lyrics
      \override LyricSpace #'minimum-distance = #1.0
    }
  }
}
```



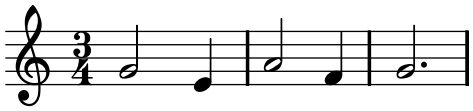


2.1.6 More about stanzas

2.1.6.1 Adding stanza numbers

Stanza numbers can be added by setting `stanza`, e.g.,

```
\new Voice {
  \time 3/4 g2 e4 a2 f4 g2.
} \addlyrics {
  \set stanza = "1. "
  Hi, my name is Bert.
} \addlyrics {
  \set stanza = "2. "
  Oh, chÃ© -- ri, je t'aime
}
```



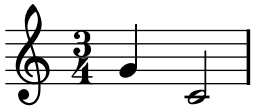
1. Hi, my name is Bert.
2. Oh, chÃ©-ri, je t'aime

These numbers are put just before the start of the first syllable.

2.1.6.2 Adding dynamics marks

Stanzas differing in loudness may be indicated by putting a dynamics mark before each stanza. In Lilypond, everthing coming in front of a stanza goes into the `StanzaNumber` object; dynamics marks are no different. For technical reasons, you have to set the stanza outside `\lyricmode`:

```
text = {
  \set stanza = \markup { \dynamic "ff" "1. " }
  \lyricmode {
    Big bang
  }
}

<<
  \new Voice = "tune" {
    \time 3/4
    g'4 c'2
  }
\new Lyrics \lyricsto "tune" \text
>>
```

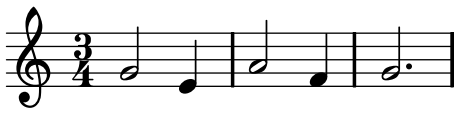


ff **1.** Big bang

2.1.6.3 Adding singer names

Names of singers can also be added. They are printed at the start of the line, just like instrument names. They are created by setting `vocalName`. A short version may be entered as `shortVocalName`.

```
\new Voice {
  \time 3/4 g2 e4 a2 f4 g2.
} \addlyrics {
  \set vocalName = "Bert "
  Hi, my name is Bert.
} \addlyrics {
  \set vocalName = "Ernie "
  Oh, che -- ri, je t'aime
}
```

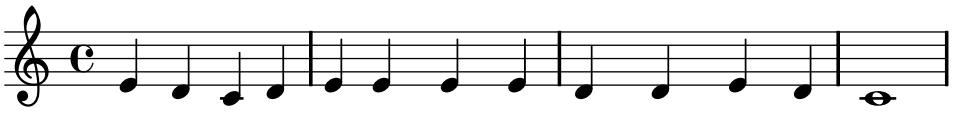


2.1.6.4 Printing stanzas at the end

Sometimes it is appropriate to have one stanza set to the music, and the rest added in verse form at the end of the piece. This can be accomplished by adding the extra verses into a `\markup` section outside of the main score block. Notice that there are two different ways to force linebreaks when using `\markup`.

```
melody = \relative c' {
e d c d | e e e e |
d d e d | c1 |
}

text = \lyricmode {
\set stanza = "1." Ma- ry had a lit- tle lamb,
its fleece was white as snow.
}

\score{ <<
  \new Voice = "one" { \melody }
  \new Lyrics \lyricsto "one" \text
>>
  \layout { }
}
\markup { \column{
  \line{ Verse 2. }
  \line{ All the children laughed and played }
  \line{ To see a lamb at school. }
  }
}
\markup{
  \wordwrap-string #"
  Verse 3.

  Mary took it home again,

  It was against the rule."
```

```
}
```



1. Ma- ry had a lit- tle lamb, its fleece was white as snow.

Verse 2.
All the children laughed and played
To see a lamb at school.

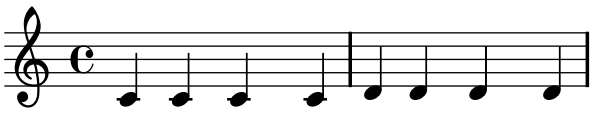
Verse 3.
Mary took it home again,
It was against the rule.

2.1.6.5 Printing stanzas at the end in multiple columns

When a piece of music has many verses, they are often printed in multiple columns across the page. An outdented verse number often introduces each verse. The following example shows how to produce such output in Lilypond.

```
melody = \relative c' {
  c c c c | d d d d
}

text = \lyricmode {
  \set stanza = "1." This is verse one.
  It has two lines.
}

\score{ <<
    \new Voice = "one" { \melody }
    \new Lyrics \lyricsto "one" \text
   >>
  \layout { }
}

\markup {
  \fill-line {
    \hspace #0.1 % moves the column off the left margin; can be removed if
        % space on the page is tight
     \column {
      \line { \bold "2."
        \column {
          "This is verse two."
          "It has two lines."
        }
      }
      \hspace #0.1 % adds vertical spacing between verses
      \line { \bold "3."
```

```
        \column {
          "This is verse three."
          "It has two lines."
        }
      }
    }
    \hspace #0.1  % adds horizontal spacing between columns; if they are
        % still too close, add more " " pairs until the result
        % looks good
     \column {
      \line { \bold "4."
        \column {
          "This is verse four."
          "It has two lines."
        }
      }
      \hspace #0.1 % adds vertical spacing between verses
      \line { \bold "5."
        \column {
          "This is verse five."
          "It has two lines."
        }
      }
    }
  \hspace #0.1 % gives some extra space on the right margin; can
      % be removed if page space is tight
  }
}
```



1. This is verse one. It has two lines.

2. This is verse two.
It has two lines.

3. This is verse three.
It has two lines.

4. This is verse four.
It has two lines.

5. This is verse five.
It has two lines.

See also

Internals Reference: `LyricText`, `StanzaNumber`, `VocalName`.

2.2 Chords

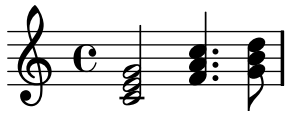
Intro text.

2.2.1 Chords sections

2.2.1.1 A lead sheet

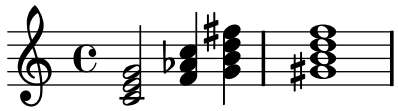
In popular music it is common to denote accompaniment with chord names. Such chords can be entered like notes,

```
\chordmode { c2 f4. g8 }
```



Now each pitch is read as the root of a chord instead of a note. This mode is switched on with `\chordmode`. Other chords can be created by adding modifiers after a colon. The following example shows a few common modifiers:

```
\chordmode { c2 f4:m g4:maj7 gis1:dim7 }
```

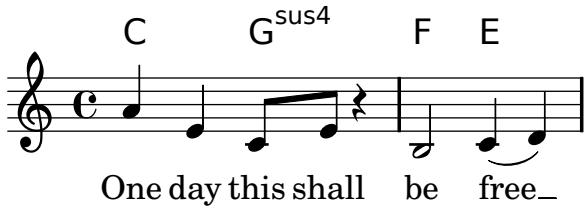


For lead sheets, chords are not printed on staves, but as names on a line for themselves. This is achieved by using `\chords` instead of `\chordmode`. This uses the same syntax as `\chordmode`, but renders the notes in a `ChordNames` context, with the following result:

```
\chords { c2 f4.:m g4.:maj7 gis8:dim7 }
```

When put together, chord names, lyrics and a melody form a lead sheet,

```
<<
  \chords { c2 g:sus4 f e }
  \relative c'' {
    a4 e c8 e r4
    b2 c4( d)
  }
  \addlyrics { One day this shall be free __ }
>>
```



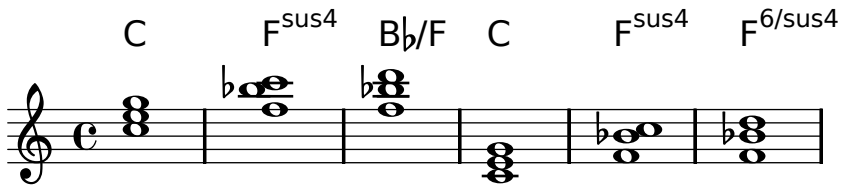
See also

A complete list of modifiers and other options for layout can be found in notation reference, Section 2.2 [Chords], page 144 .

2.2.1.2 Introducing chord names

LilyPond has support for printing chord names. Chords may be entered in musical chord notation, i.e., `< .. >`, but they can also be entered by name. Internally, the chords are represented as a set of pitches, so they can be transposed

```
twoWays = \transpose c c' {
  \chordmode {
    c1 f:sus4 bes/f
  }
  <c e g>
  <f bes c'>
  <f bes d'>
}

<< \new ChordNames \twoWays
   \new Voice \twoWays >>
```



This example also shows that the chord printing routines do not try to be intelligent. The last chord (`f bes d`) is not interpreted as an inversion.

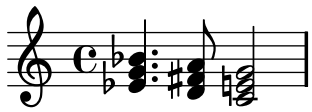
Note that the duration of chords must be specified outside the `<>`.

```
<c e g>2
```

2.2.1.3 Chords mode

In chord mode sets of pitches (chords) are entered with normal note names. A chord is entered by the root, which is entered like a normal pitch

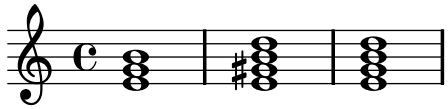
```
\chordmode { es4. d8 c2 }
```



The mode is introduced by the keyword `\chordmode`.

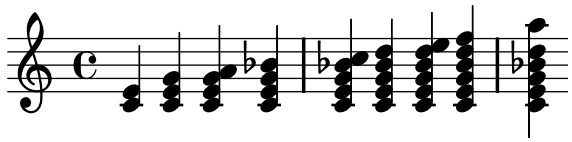
Other chords may be entered by suffixing a colon and introducing a modifier (which may include a number if desired)

```
\chordmode { e1:m e1:7 e1:m7 }
```



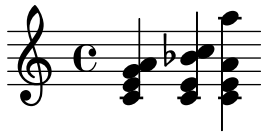
The first number following the root is taken to be the 'type' of the chord, thirds are added to the root until it reaches the specified number. The exception is `c:13`, for which the 11 is omitted.

```
\chordmode { c:3 c:5 c:6 c:7 c:8 c:9 c:10 c:11 c:13 }
```



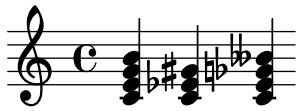
More complex chords may also be constructed adding separate steps to a chord. Additions are added after the number following the colon and are separated by dots

```
\chordmode { c:5.6 c:3.7.8 c:3.6.13 }
```



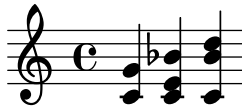
Chord steps can be altered by suffixing a `-` or `+` sign to the number

```
\chordmode { c:7+ c:5+.3- c:3-.5-.7- }
```



Removals are specified similarly and are introduced by a caret. They must come after the additions

```
\chordmode { c^3 c:7^5 c:9^3.5 }
```



Modifiers can be used to change pitches. The following modifiers are supported

`m` The minor chord. This modifier lowers the 3rd and (if present) the 7th step.

`dim` The diminished chord. This modifier lowers the 3rd, 5th and (if present) the 7th step.

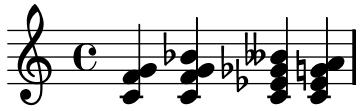
`aug` The augmented chord. This modifier raises the 5th step.

`maj` The major 7th chord. This modifier raises the 7th step if present.

`sus` The suspended 4th or 2nd. This modifier removes the 3rd step. Append either `2` or `4` to add the 2nd or 4th step to the chord.

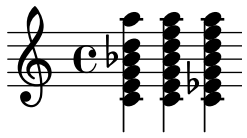
Modifiers can be mixed with additions

```
\chordmode { c:sus4 c:7sus4 c:dim7 c:m6 }
```



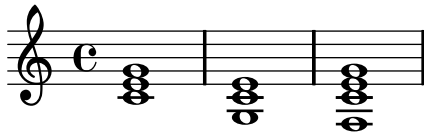
Since an unaltered 11 does not sound good when combined with an unaltered 13, the 11 is removed in this case (unless it is added explicitly)

```
\chordmode { c:13 c:13.11 c:m13 }
```



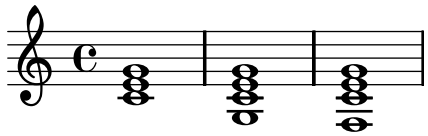
An inversion (putting one pitch of the chord on the bottom), as well as bass notes, can be specified by appending */pitch* to the chord

```
\chordmode { c1 c/g c/f }
```



A bass note can be added instead of transposed out of the chord, by using */+pitch*.

```
\chordmode { c1 c/+g c/+f }
```

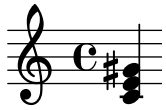


Chords is a mode similar to `\lyricmode`, etc. Most of the commands continue to work, for example, `r` and `\skip` can be used to insert rests and spaces, and property commands may be used to change various settings.

Known issues and warnings

Each step can only be present in a chord once. The following simply produces the augmented chord, since `5+` is interpreted last

```
\chordmode { c:5.5-.5+ }
```

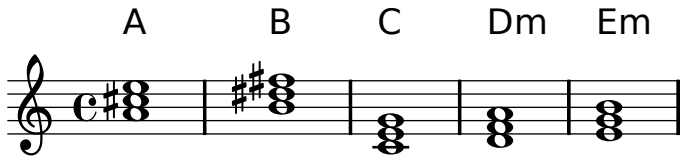


2.2.1.4 Printing chord names

For displaying printed chord names, use the `ChordNames` context. The chords may be entered either using the notation described above, or directly using `<` and `>`

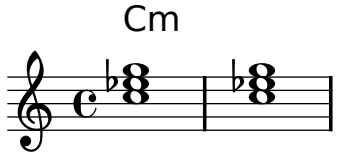
```
harmonies = {
  \chordmode {a1 b c} <d' f' a'> <e' g' b'>
}
<<
  \new ChordNames \harmonies
```

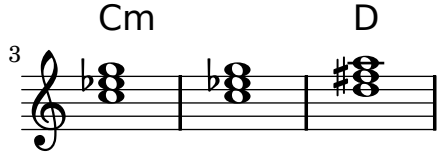
```
  \new Staff \harmonies
>>
```



You can make the chord changes stand out by setting `ChordNames.chordChanges` to true. This will only display chord names when there is a change in the chords scheme and at the start of a new line

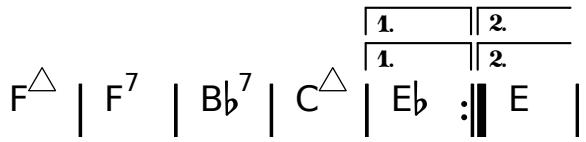
```
harmonies = \chordmode {
  c1:m c:m \break c:m c:m d
}
<<
  \new ChordNames {
    \set chordChanges = ##t
    \harmonies }
  \new Staff \transpose c c' \harmonies
>>
```





The previous examples all show chords over a staff. This is not necessary. Chords may also be printed separately. It may be necessary to add `Volta_engraver` and `Bar_engraver` for showing repeats.

```
\new ChordNames \with {
  \override BarLine #'bar-size = #4
  \consists Bar_engraver
  \consists "Volta_engraver"
}
\chordmode { \repeat volta 2 {
  f1:maj7 f:7 bes:7
  c:maj7
} \alternative {
  es e
}
}
```



The default chord name layout is a system for Jazz music, proposed by Klaus Ignatzek (see Appendix A [Literature list], page 285). It can be tuned through the following properties

`chordNameExceptions`
: This is a list that contains the chords that have special formatting.

  The exceptions list should be encoded as

  ```
  { <c f g bes>1 \markup { \super "7" "wahh" } }
  ```

  To get this information into `chordNameExceptions` takes a little manoeuvring. The following code transforms `chExceptionMusic` (which is a sequential music) into a list of exceptions.

  ```
  (sequential-music-to-chord-exceptions chExceptionMusic #t)
  ```

  Then,

  ```
  (append
   (sequential-music-to-chord-exceptions chExceptionMusic #t)
   ignatzekExceptions)
  ```

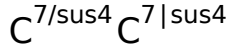
  adds the new exceptions to the default ones, which are defined in '`ly/chord-modifier-init.ly`'.

  For an example of tuning this property, see also

`majorSevenSymbol`
: This property contains the markup object used for the 7th step, when it is major. Predefined options are `whiteTriangleMarkup` and `blackTriangleMarkup`. See

`chordNameSeparator`
: Different parts of a chord name are normally separated by a slash. By setting `chordNameSeparator`, you can specify other separators, e.g.,

  ```
  \new ChordNames \chordmode {
    c:7sus4
    \set chordNameSeparator
      = \markup { \typewriter "|" }
    c:7sus4
  }
  ```

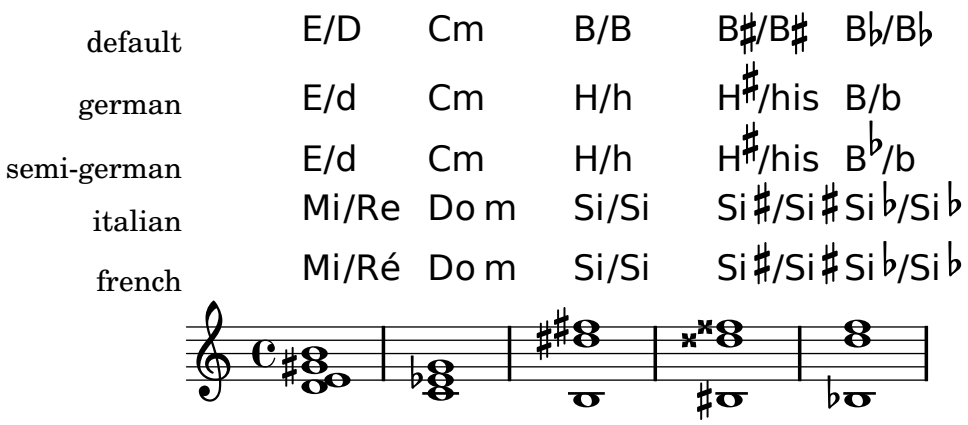


`chordRootNamer`
: The root of a chord is usually printed as a letter with an optional alteration. The transformation from pitch to letter is done by this function. Special note names (for example, the German 'H' for a B-chord) can be produced by storing a new function in this property.

`chordNoteNamer`
: The default is to print single pitch, e.g., the bass note, using the `chordRootNamer`. The `chordNoteNamer` property can be set to a specialized function to change this behavior. For example, the base can be printed in lower case.

`chordPrefixSpacer`
: The 'm' for minor chords is usually printed right after the root of the chord. By setting `chordPrefixSpacer`, you can fix a spacer between the root and 'm'. The spacer is not used when the root is altered.

The predefined variables `\germanChords`, `\semiGermanChords`, `\italianChords` and `\frenchChords` set these variables. The effect is demonstrated here,



There are also two other chord name schemes implemented: an alternate Jazz chord notation, and a systematic scheme called Banter chords. The alternate Jazz notation is also shown on the chart in Section B.1 [Chord name chart], page 286. Turning on these styles is demonstrated in

Predefined commands

`\germanChords`, `\semiGermanChords`. `\italianChords`. `\frenchChords`.

See also

Examples:

Init files: '`scm/chords-ignatzek.scm`', and '`scm/chord-entry.scm`'.

Known issues and warnings

Chord names are determined solely from the list of pitches. Chord inversions are not identified, and neither are added bass notes. This may result in strange chord names when chords are entered with the `< . . >` syntax.

2.3 Piano music

2.3.1 Piano sections

Piano staves are two normal staves coupled with a brace. The staves are largely independent, but sometimes voices can cross between the two staves. The same notation is also used for harps and other key instruments. The `PianoStaff` is especially built to handle this cross-staffing behavior. In this section we discuss the `PianoStaff` and some other pianistic peculiarities.

Known issues and warnings

Dynamics are not centered, but workarounds do exist. See the 'piano centered dynamics' template in learning manual, ⟨undefined⟩ [Piano templates], page ⟨undefined⟩ .

2.3.1.1 Automatic staff changes

Voices can be made to switch automatically between the top and the bottom staff. The syntax for this is

```
\autochange ...music...
```

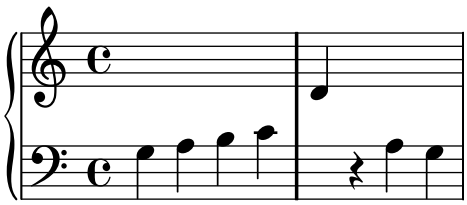
This will create two staves inside the current PianoStaff, called `up` and `down`. The lower staff will be in bass clef by default.

A `\relative` section that is outside of `\autochange` has no effect on the pitches of *music*, so, if necessary, put `\relative` inside `\autochange` like

```
\autochange \relative ... ...
```

The autochanger switches on basis of the pitch (middle C is the turning point), and it looks ahead skipping over rests to switch in advance. Here is a practical example

```
\new PianoStaff
  \autochange \relative c'
  {
    g4 a b c d r4 a g
  }
```



See also

Notation Reference: Section 2.3.1.2 [Manual staff switches], page 152.

Internals Reference: `AutoChangeMusic`.

Known issues and warnings

The staff switches may not end up in optimal places. For high quality output, staff switches should be specified manually.

`\autochange` cannot be inside `\times`.

2.3.1.2 Manual staff switches

Voices can be switched between staves manually, using the command

```
\change Staff = staffname music
```

The string *staffname* is the name of the staff. It switches the current voice from its current staff to the Staff called *staffname*. Typically *staffname* is `"up"` or `"down"`. The `Staff` referred to must already exist, so usually the setup for a score will start with a setup of the staves,

```
<<
  \new Staff = "up" {
    \skip 1 * 10  % keep staff alive
    }
```

```
    \new Staff = "down" {
      \skip 1 * 10  % idem
      }
  >>
```

and the `Voice` is inserted afterwards

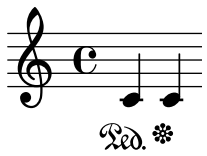
```
\context Staff = down
  \new Voice { ... \change Staff = up ... }
```

2.3.1.3 Pedals

Pianos have pedals that alter the way sound is produced. Generally, a piano has three pedals, sustain, una corda, and sostenuto.

Piano pedal instruction can be expressed by attaching `\sustainDown`, `\sustainUp`, `\unaCorda`, `\treCorde`, `\sostenutoDown` and `\sostenutoUp` to a note or chord

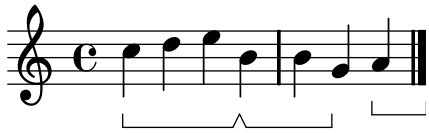
```
c'4\sustainDown c'4\sustainUp
```



What is printed can be modified by setting `pedal`*X*`Strings`, where *X* is one of the pedal types: `Sustain`, `Sostenuto` or `UnaCorda`. Refer to `SustainPedal` in the program reference for more information.

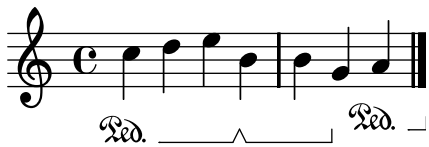
Pedals can also be indicated by a sequence of brackets, by setting the `pedalSustainStyle` property to bracket objects

```
\set Staff.pedalSustainStyle = #'bracket
c\sustainDown d e
b\sustainUp\sustainDown
b g \sustainUp a \sustainDown \bar "|."
```



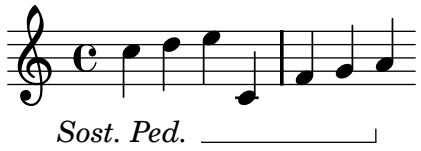
A third style of pedal notation is a mixture of text and brackets, obtained by setting the `pedalSustainStyle` property to `mixed`

```
\set Staff.pedalSustainStyle = #'mixed
c\sustainDown d e
b\sustainUp\sustainDown
b g \sustainUp a \sustainDown \bar "|."
```



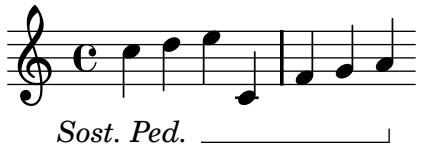
The default '*Ped.' style for sustain and damper pedals corresponds to style `#'text`. The sostenuto pedal uses `mixed` style by default.

```
c\sostenutoDown d e c, f g a\sostenutoUp
```



For fine-tuning the appearance of a pedal bracket, the properties `edge-width`, `edge-height`, and `shorten-pair` of `PianoPedalBracket` objects (see `PianoPedalBracket` in the Internals Reference) can be modified. For example, the bracket may be extended to the right edge of the note head

```
\override Staff.PianoPedalBracket #'shorten-pair = #'(0 . -1.0)
c\sostenutoDown d e c, f g a\sostenutoUp
```



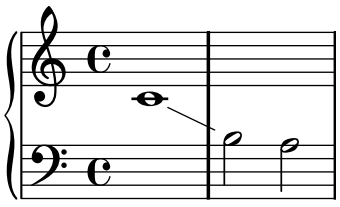
See also

Notation Reference: Section 1.2.1.4 [Ties], page 35 (laissez vibrer).

2.3.1.4 Staff switch lines

Whenever a voice switches to another staff, a line connecting the notes can be printed automatically. This is switched on by setting `followVoice` to true

```
\new PianoStaff <<
  \new Staff="one" {
    \set followVoice = ##t
    c1
    \change Staff=two
    b2 a
  }
  \new Staff="two" { \clef bass \skip 1*2 }
>>
```



See also

Internals Reference: `VoiceFollower`.

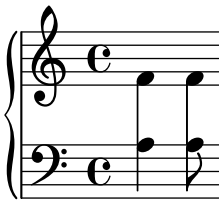
Predefined commands

`\showStaffSwitch`, `\hideStaffSwitch`.

2.3.1.5 Cross staff stems

Chords that cross staves may be produced by increasing the length of the stem in the lower staff, so it reaches the stem in the upper staff, or vice versa.

```
stemExtend = {
  \once \override Stem #'length = #10
  \once \override Stem #'cross-staff = ##t
}
noFlag = \once \override Stem #'flag-style = #'no-flag
\new PianoStaff <<
  \new Staff {
    \stemDown \stemExtend
    f'4
    \stemExtend \noFlag
    f'8
  }
  \new Staff {
    \clef bass
    a4 a8
  }
>>
```



2.4 Percussion

2.4.1 Percussion sections

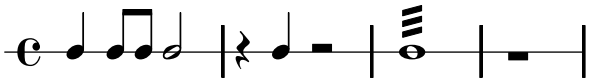
Rhythmic music is primarily used for percussion and drum notation, but it can also be used to show the rhythms of melodies.

2.4.1.1 Showing melody rhythms

Sometimes you might want to show only the rhythm of a melody. This can be done with the rhythmic staff. All pitches of notes on such a staff are squashed, and the staff itself has a single line

```
\new RhythmicStaff {
  \time 4/4
  c4 e8 f g2 | r4 g r2 | g1:32 | r1 |
```

```
}
```



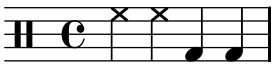
See also

Internals Reference: `RhythmicStaff`.

2.4.1.2 Entering percussion

Percussion notes may be entered in `\drummode` mode, which is similar to the standard mode for entering notes. Each piece of percussion has a full name and an abbreviated name, and both can be used in input files

```
\drums {
  hihat hh bassdrum bd
}
```



The complete list of drum names is in the init file '`ly/drumpitch-init.ly`'.

See also

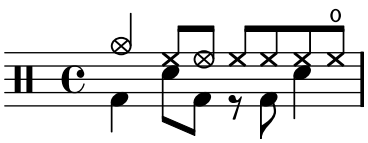
Internals Reference: `note-event`.

2.4.1.3 Percussion staves

A percussion part for more than one instrument typically uses a multiline staff where each position in the staff refers to one piece of percussion.

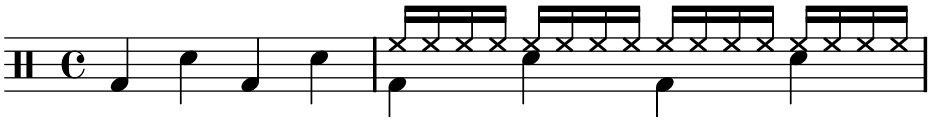
To typeset the music, the notes must be interpreted in a `DrumStaff` and `DrumVoice` contexts

```
up = \drummode { crashcymbal4 hihat8 halfopenhihat hh hh hh openhihat }
down = \drummode { bassdrum4 snare8 bd r bd sn4 }
  \new DrumStaff <<
    \new DrumVoice { \voiceOne \up }
    \new DrumVoice { \voiceTwo \down }
  >>
```



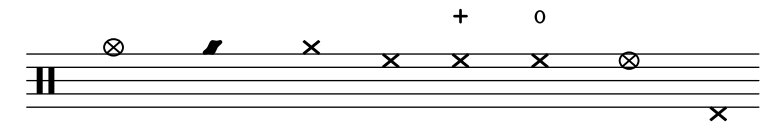
The above example shows verbose polyphonic notation. The short polyphonic notation, described in learning manual, ⟨undefined⟩ [I'm seeing Voices], page ⟨undefined⟩ , can also be used if the `DrumVoices` are instantiated by hand first. For example,

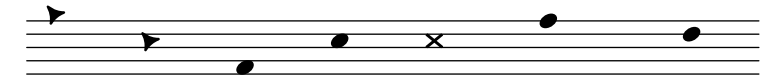
```
\new DrumStaff <<
  \new DrumVoice = "1" { s1 *2 }
  \new DrumVoice = "2" { s1 *2 }
  \drummode {
    bd4 sn4 bd4 sn4
    <<
      { \repeat unfold 16 hh16 }
      \\
      { bd4 sn4 bd4 sn4 }
    >>
  }
>>
```

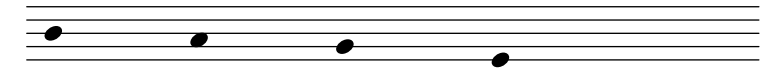


There are also other layout possibilities. To use these, set the property `drumStyleTable` in context `DrumVoice`. The following variables have been predefined

`drums-style`
: This is the default. It typesets a typical drum kit on a five-line staff

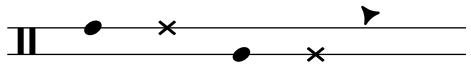






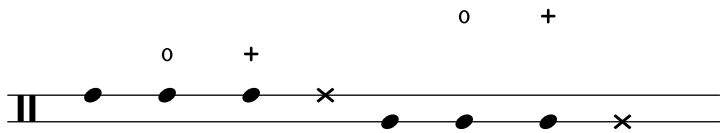
The drum scheme supports six different toms. When there are fewer toms, simply select the toms that produce the desired result, i.e., to get toms on the three middle lines you use `tommh`, `tomml`, and `tomfh`.

`timbales-style`
: This typesets timbales on a two line staff



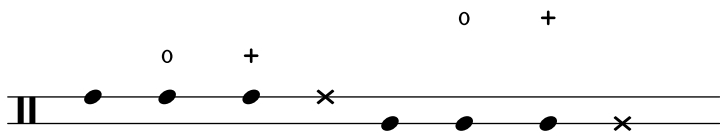
```
timh ssh timl ssl cb
```

`congas-style`
: This typesets congas on a two line staff



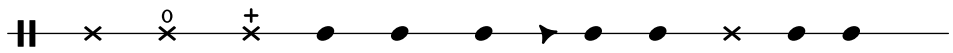
```
cgh cgho cghm ssh cgl cglo cglm ssl
```

`bongos-style`
: This typesets bongos on a two line staff



```
boh boho bohm ssh bol bolo bolm ssl
```

`percussion-style`
: To typeset all kinds of simple percussion on one line staves.

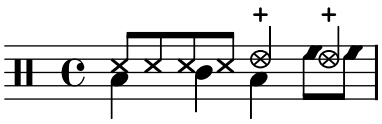


```
tri trio trim gui guis guil cb cl tamb cab mar hc
```

If you do not like any of the predefined lists you can define your own list at the top of your file

```
#(define mydrums '(
         (bassdrum     default   #f         -1)
         (snare        default   #f         0)
         (hihat        cross     #f         1)
         (pedalhihat   xcircle   "stopped"  2)
         (lowtom       diamond   #f         3)))
up = \drummode { hh8 hh hh hh hhp4 hhp }
down = \drummode { bd4 sn bd toml8 toml }
```

```
\new DrumStaff <<
  \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)
  \new DrumVoice { \voiceOne \up }
  \new DrumVoice { \voiceTwo \down }
>>
```



See also

Init files: ‘`ly/drumpitch-init.ly`’.

Internals Reference: `DrumStaff`, `DrumVoice`.

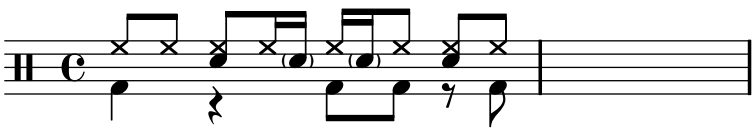
Known issues and warnings

Because general MIDI does not contain rim shots, the sidestick is used for this purpose instead.

2.4.1.4 Ghost notes

Ghost notes for drums and percussion may be created using the `\parenthesize` command detailed in Section 1.7.1.5 [Parentheses], page 109. However, the default `\drummode` does not include the `Parenthesis_engraver` plugin which allows this. You must add the plugin explicitly in the context definition as detailed in Section 6.1.3 [Changing context properties on the fly], page 251.

```
\new DrumStaff \with {
  \consists "Parenthesis_engraver"
} <<
  \context DrumVoice  = "1"  { s1 *2 }
  \context DrumVoice  = "2" { s1 *2 }
  \drummode {
    <<
      {
        hh8[ hh] <hh sn> hh16
        < \parenthesize sn > hh < \parenthesize
        sn > hh8 <hh sn> hh
      } \\ {
        bd4 r4 bd8 bd r8 bd
      }
    >>
  }
>>
```



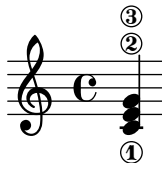
Also note that you must add chords (< > brackets) around each `\parenthesize` statement.

2.5 Guitar

2.5.1 Guitar sections

2.5.1.1 String number indications

String numbers can be added to chords, by indicating the string number with *number*,



See also

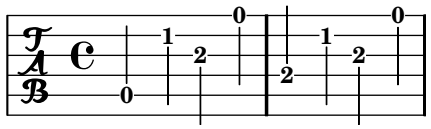
Internals Reference: `StringNumber`,

2.5.1.2 Tablatures basic

Tablature notation is used for notating music for plucked string instruments. Pitches are not denoted with note heads, but by numbers indicating on which string and fret a note must be played. LilyPond offers limited support for tablature.

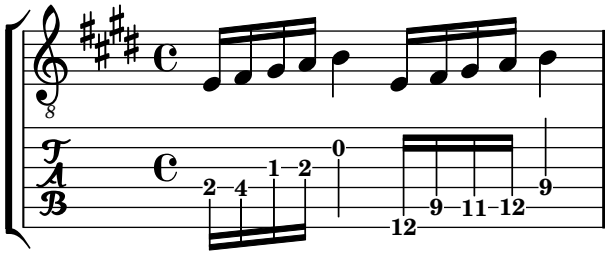
The string number associated to a note is given as a backslash followed by a number, e.g., `c4\3` for a C quarter on the third string. By default, string 1 is the highest one, and the tuning defaults to the standard guitar tuning (with 6 strings). The notes are printed as tablature, by using `TabStaff` and `TabVoice` contexts

```
\new TabStaff {
  a,4\5 c'\2 a\3 e'\1
  e\4 c'\2 a\3 e'\1
}
```



When no string is specified, the first string that does not give a fret number less than `minimumFret` is selected. The default value for `minimumFret` is 0

```
e16 fis gis a b4
\set TabStaff.minimumFret = #8
e16 fis gis a b4
```



Selected Snippets

To print tablatures with stems down and horizontal beams, initialize the `TabStaff` with this code:

```
\stemDown
\override Beam #'damping = #100000
```

See also

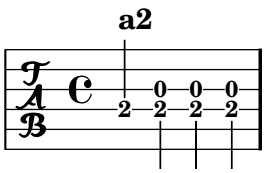
Internals Reference: `TabNoteHead`, `TabStaff`, `TabVoice`.

Known issues and warnings

Chords are not handled in a special way, and hence the automatic string selector may easily select the same string to two notes in a chord.

In order to handle `\partcombine`, a `TabStaff` must use specially-created voices:

```
melodia = \partcombine { e4 g g g }{ e4 e e e }
<<
  \new TabStaff <<
    \new TabVoice = "one" s1
    \new TabVoice = "two" s1
    \new TabVoice = "shared" s1
    \new TabVoice = "solo" s1
    { \melodia }
  >>
>>
```

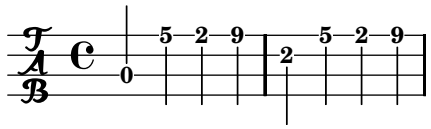


2.5.1.3 Non-guitar tablatures

You can change the tuning of the strings. A string tuning is given as a Scheme list with one integer number for each string, the number being the pitch (measured in semitones relative to middle C) of an open string. The numbers specified for `stringTunings` are the numbers of semitones to subtract or add, starting the specified pitch by default middle C, in string order. LilyPond automatically calculates the number of strings by looking at `stringTunings`.

In the next example, `stringTunings` is set for the pitches e, a, d, and g

```
\new TabStaff <<
  \set TabStaff.stringTunings = #'(-5 -10 -15 -20)
  {
    a,4 c' a e' e c' a e'
  }
>>
```



LilyPond comes with predefined string tunings for banjo, mandolin, guitar and bass guitar.

```
\set TabStaff.stringTunings = #bass-tuning
```

The default string tuning is `guitar-tuning` (the standard EADGBE tuning). Some other predefined tunings are `guitar-open-g-tuning`, `mandolin-tuning` and `banjo-open-g-tuning`.

See also

The file '`scm/output-lib.scm`' contains the predefined string tunings. Internals Reference: `Tab_note_heads_engraver`.

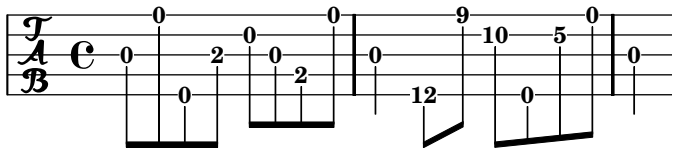
Known issues and warnings

No guitar special effects have been implemented.

2.5.1.4 Banjo tablatures

LilyPond has basic support for five stringed banjo. When making tablatures for five stringed banjo, use the banjo tablature format function to get correct fret numbers for the fifth string:

```
\new TabStaff <<
  \set TabStaff.tablatureFormat = #fret-number-tablature-format-banjo
  \set TabStaff.stringTunings = #banjo-open-g-tuning
  {
    \stemDown
    g8 d' g'\5 a b g e d' |
    g4 d''8\5 b' a'\2 g'\5 e'\2 d' |
    g4
  }
>>
```



A number of common tunings for banjo are predefined in LilyPond: `banjo-c-tuning` (gCGBD), `banjo-modal-tuning` (gDGCD), `banjo-open-d-tuning` (aDF#AD) and `banjo-open-dm-tuning` (aDFAD).

These tunings may be converted to four string banjo tunings using the `four-string-banjo` function:

```
\set TabStaff.stringTunings = #(four-string-banjo banjo-c-tuning)
```

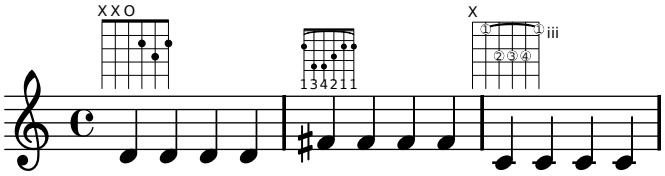
See also

The file '`scm/output-lib.scm`' contains predefined banjo tunings.

2.5.1.5 Fret diagrams

Fret diagrams can be added to music as a markup to the desired note. The markup contains information about the desired fret diagram, as shown in the following example

```
\new Voice {
  d'^\markup \fret-diagram #"6-x;5-x;4-o;3-2;2-3;1-2;"
  d' d' d'
  fis'^\markup \override #'(size . 0.75) {
    \override #'(finger-code . below-string) {
      \fret-diagram-verbose #'((place-fret 6 2 1) (barre 6 1 2)
                               (place-fret 5 4 3) (place-fret 4 4 4)
                               (place-fret 3 3 2) (place-fret 2 2 1)
                               (place-fret 1 2 1))
    }
  }
  fis' fis' fis'
  c'^\markup \override #'(dot-radius . 0.35) {
    \override #'(finger-code . in-dot) {
      \override #'(dot-color . white) {
        \fret-diagram-terse #"x;3-1-(;5-2;5-3;5-4;3-1-);"
      }
    }
  }
  c' c' c'
}
```



There are three different fret-diagram markup interfaces: standard, terse, and verbose. The three interfaces produce equivalent markups, but have varying amounts of information in the markup string. Details about the markup interfaces are found at Section B.6 [Overview of text markup commands], page 306.

You can set a number of graphical properties according to your preference. Details about the property interface to fret diagrams are found at `fret-diagram-interface`.

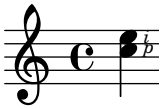
See also

Examples:

2.5.1.6 Right hand fingerings

Right hand fingerings in chords can be entered using *note*`-\rightHandFinger` *finger*

```
<c-\rightHandFinger #1 e-\rightHandFinger #2 >
```



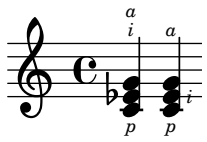
for brevity, you can abbreviate `\rightHandFinger` to something short, for example `RH`,

```
#(define RH rightHandFinger)
```

Selected Snippets

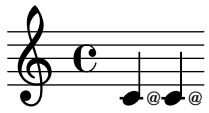
You may exercise greater control over right handing fingerings by setting `strokeFingerOrientations`,

```
#(define RH rightHandFinger)
{
  \set strokeFingerOrientations = #'(up down)
  <c-\RH #1 es-\RH #2 g-\RH #4 > 4
  \set strokeFingerOrientations = #'(up right down)
  <c-\RH #1 es-\RH #2 g-\RH #4 > 4
}
```



The letters used for the fingerings are contained in the property `digit-names`, but they can also be set individually by supplying `\rightHandFinger` with a string argument, as in the following example

```
#(define RH rightHandFinger)
{
  \set strokeFingerOrientations = #'(right)
  \override StrokeFinger #'digit-names = ##("x" "y" "z" "!" "@")
  <c-\RH #5 >4
  <c-\RH "@">4
}
```



See also

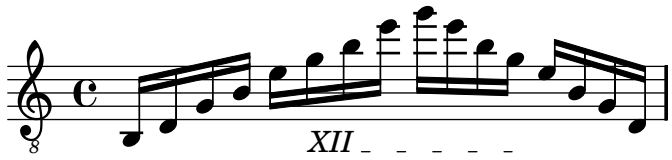
Internals Reference: `StrokeFinger`

2.5.1.7 Other guitar issues

This example demonstrates how to include guitar position and barring indications.

```
\clef "G_8"
b16 d16 g16 b16 e16
\textSpannerDown
\override TextSpanner #'bound-details #'left #'text = #"XII "
  g16\startTextSpan
  b16 e16 g16 e16 b16 g16\stopTextSpan
```

```
e16 b16 g16 d16
```



Stopped (X) note heads are used in guitar music to signal a place where the guitarist must play a certain note or chord, with its fingers just touching the strings instead of fully pressing them. This gives the sound a percussive noise-like sound that still maintains part of the original pitch. It is notated with cross note heads; this is demonstrated in Section 1.1.4.1 [Special note heads], page 26.

2.6 Orchestral strings

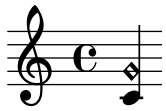
This section includes extra information for writing for orchestral strings.

2.6.1 Orchestral strings sections

2.6.1.1 Artificial harmonics (strings)

Artificial harmonics are notated with a different note head style. They are entered by marking the harmonic pitch with `\harmonic`.

```
<c g'\harmonic>4
```



2.7 Bagpipes

This section includes extra information for writing for bagpipes.

2.7.1 Bagpipe sections

2.7.1.1 Bagpipe definitions

LilyPond contains special definitions for music for the Scottish highland bagpipe; to use them, add

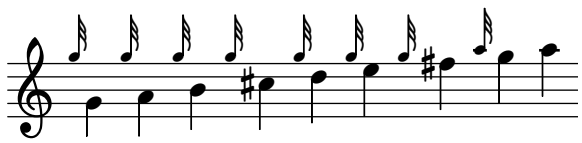
```
\include "bagpipe.ly"
```

at the top of your input file. This lets you add the special gracenotes common to bagpipe music with short commands. For example, you could write `\taor` instead of

```
\grace { \small G32[ d G e] }
```

`bagpipe.ly` also contains pitch definitions for the bagpipe notes in the appropriate octaves, so you do not need to worry about `\relative` or `\transpose`.

```
\include "bagpipe.ly"
{ \grg G4 \grg a \grg b \grg c \grg d \grg e \grg f \grA g A }
```



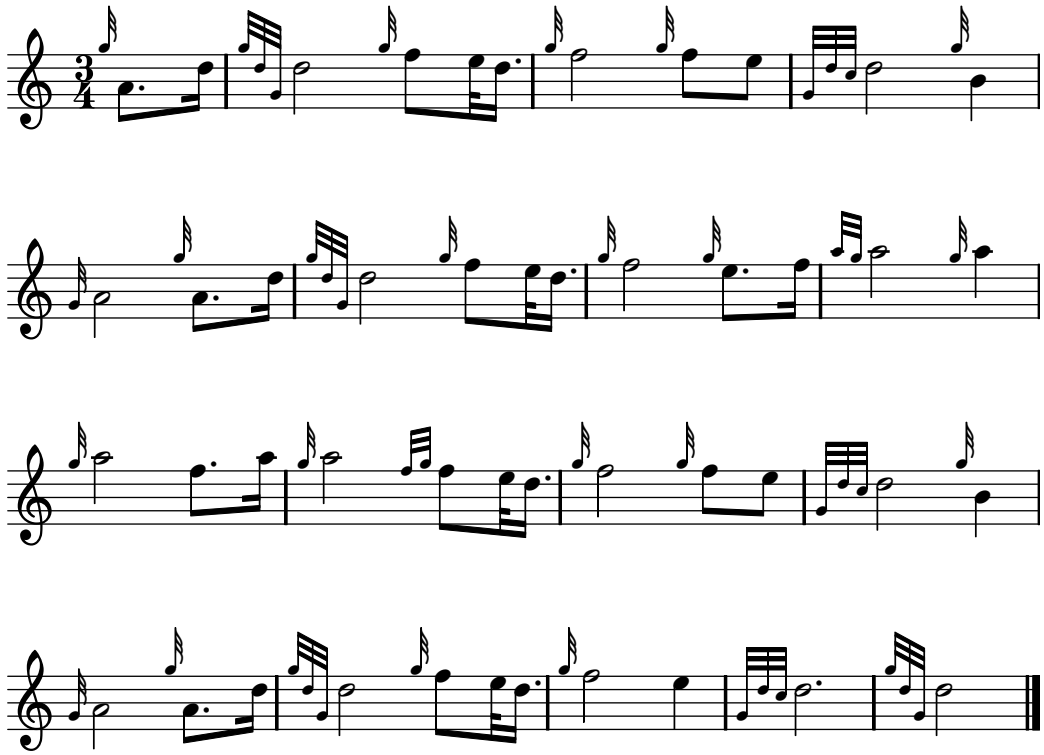
Bagpipe music nominally uses the key of D Major (even though that isn't really true). However, since that is the only key that can be used, the key signature is normally not written out. To set this up correctly, always start your music with `\hideKeySignature`. If you for some reason want to show the key signature, you can use `\showKeySignature` instead.

Some modern music use cross fingering on c and f to flatten those notes. This can be indicated by `cflat` or `fflat`. Similarly, the piobaireachd high g can be written `gflat` when it occurs in light music.

2.7.1.2 Bagpipe example

This is what the well known tune Amazing Grace looks like in bagpipe notation.

```
\include "bagpipe.ly"
\layout {
  indent = 0.0\cm
  \context { \Score \remove "Bar_number_engraver" }
}

\header {
  title = "Amazing Grace"
  meter = "Hymn"
  arranger = "Trad. arr."
}

{
  \hideKeySignature
  \time 3/4
  \grg \partial 4 a8. d16
  \slurd d2 \grg f8[ e32 d16.]
  \grg f2 \grg f8 e
  \thrwd d2 \grg b4
  \grG a2 \grg a8. d16
  \slurd d2 \grg f8[ e32 d16.]
  \grg f2 \grg e8. f16
  \dblA A2 \grg A4
  \grg A2 f8. A16
  \grg A2 \hdblf f8[ e32 d16.]
  \grg f2 \grg f8 e
  \thrwd d2 \grg b4
  \grG a2 \grg a8. d16
  \slurd d2 \grg f8[ e32 d16.]
  \grg f2 e4
  \thrwd d2.
  \slurd d2
  \bar "|."
}
```



2.8 Ancient notation

Support for ancient notation includes features for mensural notation and Gregorian Chant notation. There is also limited support for figured bass notation.

Many graphical objects provide a `style` property, see

- Section 2.8.1.1 [Ancient note heads], page 168,
- Section 2.8.1.2 [Ancient accidentals], page 168,
- Section 2.8.1.3 [Ancient rests], page 169,
- Section 2.8.1.4 [Ancient clefs], page 169,
- Section 2.8.1.5 [Ancient flags], page 171,
- Section 2.8.1.6 [Ancient time signatures], page 172.

By manipulating such a grob property, the typographical appearance of the affected graphical objects can be accommodated for a specific notation flavor without the need for introducing any new notational concept.

In addition to the standard articulation signs described in section Section 1.3.1.1 [Articulations and ornamentations], page 68, specific articulation signs for ancient notation are provided.

- Section 2.8.2.1 [Ancient articulations], page 173

Other aspects of ancient notation can not that easily be expressed in terms of just changing a style property of a graphical object or adding articulation signs. Some notational concepts are introduced specifically for ancient notation,

- Section 2.8.2.2 [Custodes], page 174,
- Section 2.8.2.3 [Divisiones], page 175,
- Section 2.8.2.4 [Ligatures], page 176.

If this all is too much of documentation for you, and you just want to dive into typesetting without worrying too much about the details on how to customize a context, you may have a look at the predefined contexts. Use them to set up predefined style-specific voice and staff contexts, and directly go ahead with the note entry,

- Section 2.8.3.1 [Gregorian Chant contexts], page 183,

- Section 2.8.3.2 [Mensural contexts], page 184.

There is limited support for figured bass notation which came up during the baroque period.

- Section 2.8.5 [Figured bass], page 185

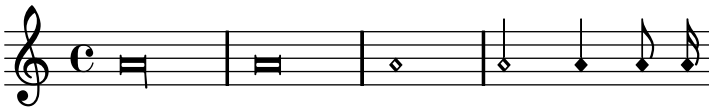
2.8.1 Alternative note signs for ancient music

2.8.1.1 Ancient note heads

For ancient notation, a note head style other than the `default` style may be chosen. This is accomplished by setting the `style` property of the `NoteHead` object to `baroque`, `neomensural`, `mensural` or `petrucci`. The `baroque` style differs from the `default` style only in using a square shape for `\breve` note heads. The `neomensural` style differs from the `baroque` style in that it uses rhomboidal heads for whole notes and all smaller durations. Stems are centered on the note heads. This style is particularly useful when transcribing mensural music, e.g., for the incipit. The `mensural` style produces note heads that mimic the look of note heads in historic printings of the 16th century. Finally, the `petrucci` style also mimicks historic printings, but uses bigger note heads.

The following example demonstrates the `neomensural` style

```
\set Score.skipBars = ##t
\override NoteHead #'style = #'neomensural
a'\longa a'\breve a'1 a'2 a'4 a'8 a'16
```



When typesetting a piece in Gregorian Chant notation, the `Gregorian_ligature_engraver` will automatically select the proper note heads, so there is no need to explicitly set the note head style. Still, the note head style can be set, e.g., to `vaticana_punctum` to produce punctum neumes. Similarly, a `Mensural_ligature_engraver` is used to automatically assemble mensural ligatures. See Section 2.8.2.4 [Ligatures], page 176, for how ligature engravers work.

See also

Section B.5 [Note head styles], page 305, gives an overview over all available note head styles.

2.8.1.2 Ancient accidentals

Use the `glyph-name-alist` property of grob `Accidental` and `KeySignature` to select ancient accidentals.

vaticana medicaea hufnagel mensural

As shown, not all accidentals are supported by each style. When trying to access an unsupported accidental, LilyPond will switch to a different style, as demonstrated in

Similarly to local accidentals, the style of the key signature can be controlled by the `glyph-name-alist` property of the `KeySignature` grob.

See also

Notation Reference: Section 1.1 [Pitches], page 1, Section 1.1.1.3 [Accidentals], page 4, and Section 1.1.3.5 [Automatic accidentals], page 19, give a general introduction of the use of accidentals. Section 1.1.3.2 [Key signature], page 14, gives a general introduction of the use of key signatures.

Internals Reference: `KeySignature`.

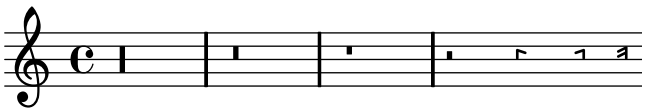
Examples:

2.8.1.3 Ancient rests

Use the `style` property of grob `Rest` to select ancient rests. Supported styles are `classical`, `neomensural`, and `mensural`. `classical` differs from the `default` style only in that the quarter rest looks like a horizontally mirrored 8th rest. The `neomensural` style suits well for, e.g., the incipit of a transcribed mensural piece of music. The `mensural` style finally mimics the appearance of rests as in historic prints of the 16th century.

The following example demonstrates the `neomensural` style

```
\set Score.skipBars = ##t
\override Rest #'style = #'neomensural
r\longa r\breve r1 r2 r4 r8 r16
```



There are no 32th and 64th rests specifically for the mensural or neo-mensural style. Instead, the rests from the default style will be taken. See

There are no rests in Gregorian Chant notation; instead, it uses Section 2.8.2.3 [Divisiones], page 175.

See also

Notation Reference: Section 1.2.2.1 [Rests], page 37, gives a general introduction into the use of rests.

2.8.1.4 Ancient clefs

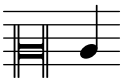
LilyPond supports a variety of clefs, many of them ancient.

The following table shows all ancient clefs that are supported via the `\clef` command. Some of the clefs use the same glyph, but differ only with respect to the line they are printed on. In such cases, a trailing number in the name is used to enumerate these clefs. Still, you can manually force a clef glyph to be typeset on an arbitrary line, as described in Section 1.1.3.1 [Clef], page 11. The note printed to the right side of each clef in the example column denotes the `c'` with respect to that clef.

Description	Supported Clefs	Example

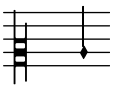
modern style mensural C clef

`neomensural-c1`, `neomensural-c2`, `neomensural-c3`, `neomensural-c4`



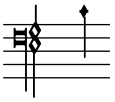
petrucci style mensural C clefs, for use on different staff lines (the examples show the 2nd staff line C clef)

`petrucci-c1`, `petrucci-c2`, `petrucci-c3`, `petrucci-c4`, `petrucci-c5`



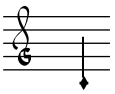
petrucci style mensural F clef

`petrucci-f`



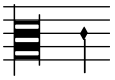
petrucci style mensural G clef

`petrucci-g`



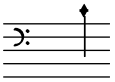
historic style mensural C clef

`mensural-c1`, `mensural-c2`, `mensural-c3`, `mensural-c4`



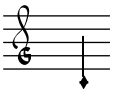
historic style mensural F clef

`mensural-f`



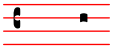
historic style mensural G clef

`mensural-g`



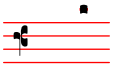
Editio Vaticana style do clef

`vaticana-do1`, `vaticana-do2`, `vaticana-do3`

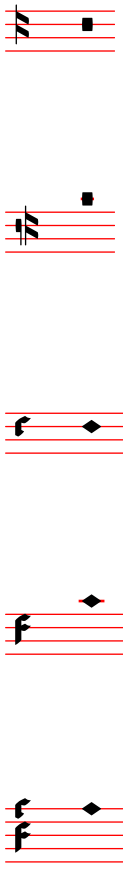


Editio Vaticana style fa clef

`vaticana-fa1`, `vaticana-fa2`



Editio Medicaea style do clef	`medicaea-do1`, `medicaea-do2`, `medicaea-do3`
Editio Medicaea style fa clef	`medicaea-fa1`, `medicaea-fa2`
historic style hufnagel do clef	`hufnagel-do1`, `hufnagel-do2`, `hufnagel-do3`
historic style hufnagel fa clef	`hufnagel-fa1`, `hufnagel-fa2`
historic style hufnagel combined do/fa clef	`hufnagel-do-fa`



Modern style means "as is typeset in contemporary editions of transcribed mensural music."

Petrucci style means "inspired by printings published by the famous engraver Petrucci (1466-1539)."

Historic style means "as was typeset or written in historic editions (other than those of Petrucci)."

Editio XXX style means "as is/was printed in Editio XXX."

Petrucci used C clefs with differently balanced left-side vertical beams, depending on which staff line it is printed.

See also

Notation Reference: see Section 1.1.3.1 [Clef], page 11.

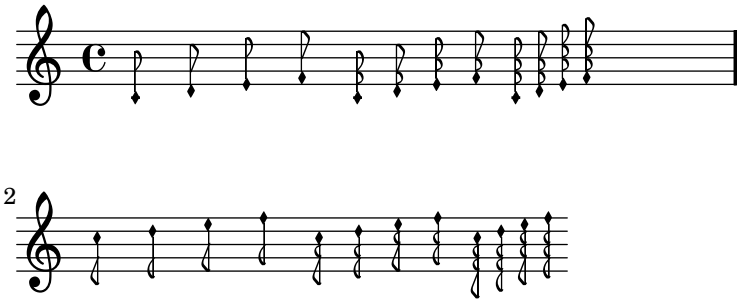
Known issues and warnings

The mensural g clef is mapped to the Petrucci g clef.

2.8.1.5 Ancient flags

Use the `flag-style` property of grob `Stem` to select ancient flags. Besides the `default` flag style, only the `mensural` style is supported

```
\override Stem #'flag-style = #'mensural
\override Stem #'thickness = #1.0
\override NoteHead #'style = #'mensural
\autoBeamOff
c'8 d'8 e'8 f'8 c'16 d'16 e'16 f'16 c'32 d'32 e'32 f'32 s8
c''8 d''8 e''8 f''8 c''16 d''16 e''16 f''16 c''32 d''32 e''32 f''32
```



Note that the innermost flare of each mensural flag always is vertically aligned with a staff line.

There is no particular flag style for neo-mensural notation. Hence, when typesetting the incipit of a transcribed piece of mensural music, the default flag style should be used. There are no flags in Gregorian Chant notation.

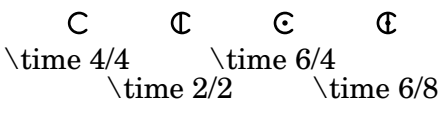
Known issues and warnings

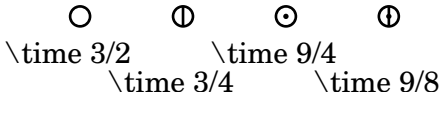
The attachment of ancient flags to stems is slightly off due to a change in early 2.3.x.

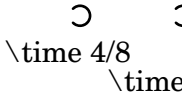
Vertically aligning each flag with a staff line assumes that stems always end either exactly on or exactly in the middle between two staff lines. This may not always be true when using advanced layout features of classical notation (which however are typically out of scope for mensural notation).

2.8.1.6 Ancient time signatures

There is limited support for mensural time signatures. The glyphs are hard-wired to particular time fractions. In other words, to get a particular mensural signature glyph with the `\time n/m` command, `n` and `m` have to be chosen according to the following table

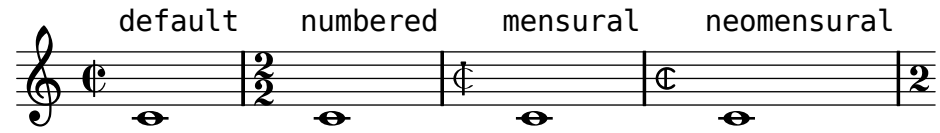


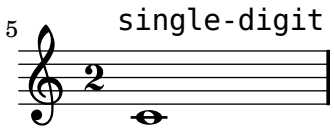




Use the `style` property of grob `TimeSignature` to select ancient time signatures. Supported styles are `neomensural` and `mensural`. The above table uses the `neomensural` style. This style is appropriate for the incipit of transcriptions of mensural pieces. The `mensural` style mimics the look of historical printings of the 16th century.

The following examples show the differences in style,





See also

This manual: Section 1.2.3.1 [Time signature], page 42, gives a general introduction to the use of time signatures.

Known issues and warnings

Ratios of note durations do not change with the time signature. For example, the ratio of 1 brevis = 3 semibrevis (tempus perfectum) must be made by hand, by setting

```
breveTP = #(ly:make-duration -1 0 3 2)
...
{ c\breveTP f1 }
```

This sets `breveTP` to 3/2 times 2 = 3 times a whole note.

The `old6/8alt` symbol (an alternate symbol for 6/8) is not addressable with `\time`. Use a `\markup` instead

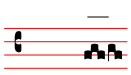
2.8.2 Additional note signs for ancient music

2.8.2.1 Ancient articulations

In addition to the standard articulation signs described in section Section 1.3.1.1 [Articulations and ornamentations], page 68, articulation signs for ancient notation are provided. These are specifically designed for use with notation in Editio Vaticana style.

```
\include "gregorian-init.ly"
\score {
  \new VaticanaVoice {
    \override TextScript #'font-family = #'typewriter
    \override TextScript #'font-shape = #'upright
    \override Script #'padding = #-0.1
    a\ictus_"ictus" \break
    a\circulus_"circulus" \break
    a\semicirculus_"semicirculus" \break
```

```
      a\accentus_"accentus" \break
      \[ a_"episem" \episemInitium \pes b \flexa a b \episemFinis \flexa a \]
    }
  }
```



Known issues and warnings

Some articulations are vertically placed too closely to the corresponding note heads.

The episem line is not displayed in many cases. If it is displayed, the right end of the episem line is often too far to the right.

2.8.2.2 Custodes

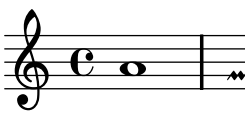
A *custos* (plural: *custodes*; Latin word for 'guard') is a symbol that appears at the end of a staff. It anticipates the pitch of the first note(s) of the following line thus helping the performer to manage line breaks during performance.

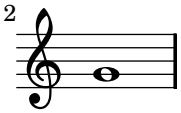
Custodes were frequently used in music notation until the 17th century. Nowadays, they have survived only in a few particular forms of musical notation such as contemporary editions of Gregorian chant like the *editio vaticana*. There are different custos glyphs used in different flavors of notational style.

For typesetting custodes, just put a `Custos_engraver` into the `Staff` context when declaring the `\layout` block, as shown in the following example

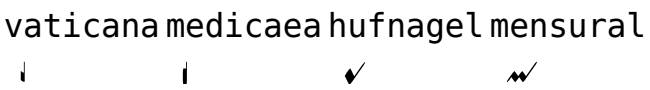
```
\layout {
  \context {
    \Staff
    \consists Custos_engraver
    Custos \override #'style = #'mensural
  }
}
```

The result looks like this





The custos glyph is selected by the `style` property. The styles supported are `vaticana`, `medicaea`, `hufnagel`, and `mensural`. They are demonstrated in the following fragment



See also

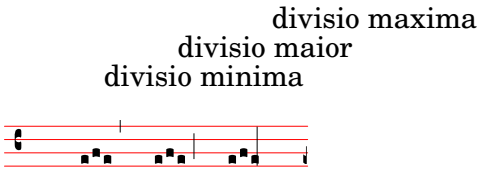
Internals Reference: `Custos`.

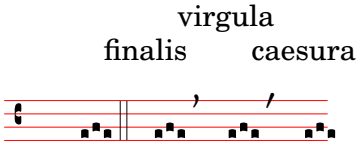
Examples:

2.8.2.3 Divisiones

A *divisio* (plural: *divisiones*; Latin word for 'division') is a staff context symbol that is used to structure Gregorian music into phrases and sections. The musical meaning of *divisio minima*, *divisio maior*, and *divisio maxima* can be characterized as short, medium, and long pause, somewhat like the breathmarks from Section 1.3.2.3 [Breath marks], page 74. The *finalis* sign not only marks the end of a chant, but is also frequently used within a single antiphonal/responsorial chant to mark the end of each section.

To use divisiones, include the file '`gregorian-init.ly`'. It contains definitions that you can apply by just inserting `\divisioMinima`, `\divisioMaior`, `\divisioMaxima`, and `\finalis` at proper places in the input. Some editions use *virgula* or *caesura* instead of divisio minima. Therefore, '`gregorian-init.ly`' also defines `\virgula` and `\caesura`





Predefined commands

`\virgula`, `\caesura`, `\divisioMinima`, `\divisioMaior`, `\divisioMaxima`, `\finalis`.

See also

Notation Reference: Section 1.3.2.3 [Breath marks], page 74.

Internals Reference: `BreathingSign`.

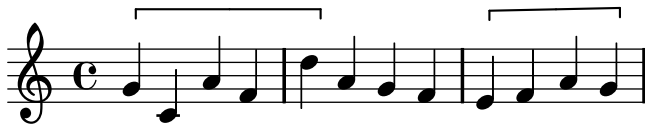
Examples:

2.8.2.4 Ligatures

A ligature is a graphical symbol that represents at least two distinct notes. Ligatures originally appeared in the manuscripts of Gregorian chant notation to denote ascending or descending sequences of notes.

Ligatures are entered by enclosing them in `\[` and `\]`. Some ligature styles may need additional input syntax specific for this particular type of ligature. By default, the `LigatureBracket` engraver just puts a square bracket above the ligature

```
\transpose c c' {
  \[ g c a f d' \]
  a g f
  \[ e f a g \]
}
```



To select a specific style of ligatures, a proper ligature engraver has to be added to the `Voice` context, as explained in the following subsections. Only white mensural ligatures are supported with certain limitations.

Known issues and warnings

Ligatures need special spacing that has not yet been implemented. As a result, there is too much space between ligatures most of the time, and line breaking often is unsatisfactory. Also, lyrics do not correctly align with ligatures.

Accidentals must not be printed within a ligature, but instead need to be collected and printed in front of it.

The syntax still uses the deprecated infix style `\[ music expr \]`. For consistency reasons, it will eventually be changed to postfix style `note\[ ... note\]`. Alternatively, the file '`gregorian-init.ly`' can be included; it provides a scheme function

`\ligature` *`music expr`*

with the same effect and is believed to be stable.

2.8.2.5 White mensural ligatures

There is limited support for white mensural ligatures.

To engrave white mensural ligatures, in the layout block put the `Mensural_ligature_engraver` into the `Voice` context, and remove the `Ligature_bracket_engraver`, like this

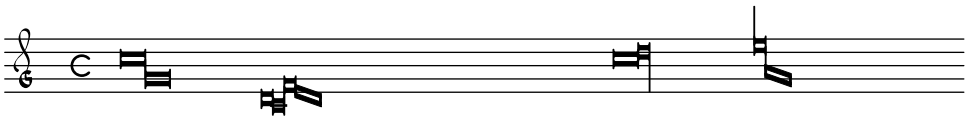
```
\layout {
  \context {
    \Voice
    \remove Ligature_bracket_engraver
    \consists Mensural_ligature_engraver
  }
}
```

There is no additional input language to describe the shape of a white mensural ligature. The shape is rather determined solely from the pitch and duration of the enclosed notes. While this approach may take a new user a while to get accustomed to, it has the great advantage

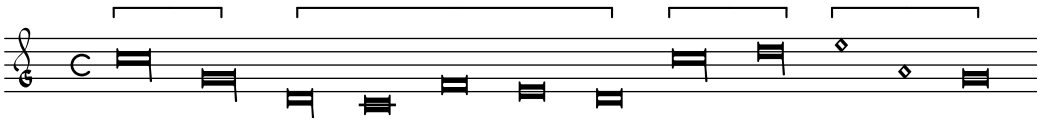
that the full musical information of the ligature is known internally. This is not only required for correct MIDI output, but also allows for automatic transcription of the ligatures.

For example,

```
\set Score.timing = ##f
\set Score.defaultBarType = "empty"
\override NoteHead #'style = #'neomensural
\override Staff.TimeSignature #'style = #'neomensural
\clef "petrucci-g"
\[ c'\maxima g \]
\[ d\longa c\breve f e d \]
\[ c'\maxima d'\longa \]
\[ e'1 a g\breve \]
```



Without replacing `Ligature_bracket_engraver` with `Mensural_ligature_engraver`, the same music transcribes to the following



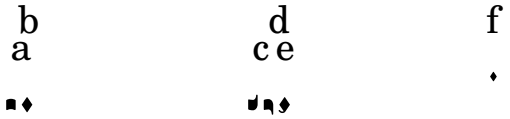
Known issues and warnings

Horizontal spacing is poor.

2.8.2.6 Gregorian square neumes ligatures

There is limited support for Gregorian square neumes notation (following the style of the Editio Vaticana). Core ligatures can already be typeset, but essential issues for serious typesetting are still lacking, such as (among others) horizontal alignment of multiple ligatures, lyrics alignment and proper handling of accidentals.

The following table contains the extended neumes table of the 2nd volume of the Antiphonale Romanum (*Liber Hymnarius*), published 1983 by the monks of Solesmes.

Neuma aut Neumarum Elementa	**Figurae Rectae**	**Figurae Liquescentes Auctae**	**Figurae Liquescentes Deminutae**
`1. Punctum`			
			

2. Virga

g

3. Apostropha vel Stropha

h i

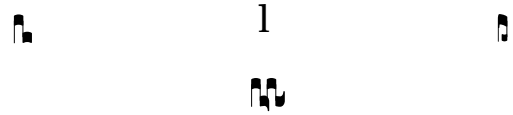
4. Oriscus

j

5. Clivis vel Flexa

m

l

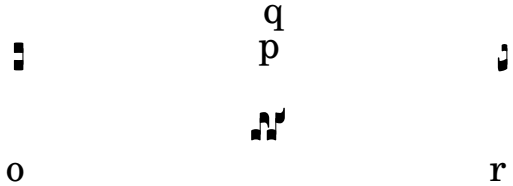


k n

6. Podatus vel Pes

q

p



o r

7. Pes Quassus

s t

8. Quilisma Pes

u v

9. Podatus Initio Debilis

w x

10. Torculus

y z A

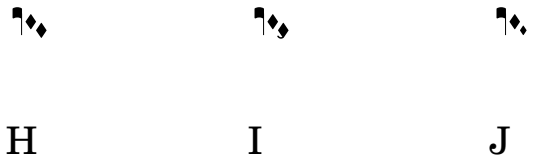
11. Torculus Initio Debilis

B C D

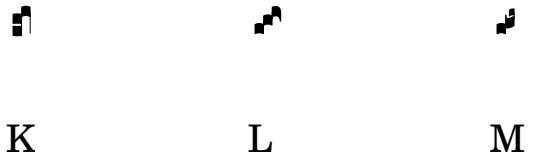
12. Porrectus

E F G

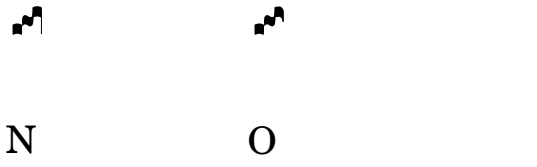
13. Climacus



14. Scandicus



15. Salicus



16. Trigonus

Unlike most other neumes notation systems, the input language for neumes does not reflect the typographical appearance, but is designed to focus on musical meaning. For example, `\[ a \pes b \flexa g \]` produces a Torculus consisting of three Punctum heads, while `\[ a \flexa g \pes b \]` produces a Porrectus with a curved flexa shape and only a single Punctum head. There is no command to explicitly typeset the curved flexa shape; the decision of when to typeset a curved flexa shape is based on the musical input. The idea of this approach is to separate the musical aspects of the input from the notation style of the output. This way, the same input can be reused to typeset the same music in a different style of Gregorian chant notation.

The following table shows the code fragments that produce the ligatures in the above neumes table. The letter in the first column in each line of the below table indicates to which ligature in the above table it refers. The second column gives the name of the ligature. The third column shows the code fragment that produces this ligature, using `g`, `a`, and `b` as example pitches.

#	**Name**	**Input Language**
a	Punctum	`\[ b \]`
b	Punctum Inclinatum	`\[ \inclinatum b \]`
c	Punctum Auctum Ascendens	`\[ \auctum \ascendens b \]`
d	Punctum Auctum Descendens	`\[ \auctum \descendens b \]`
e	Punctum Inclinatum Auctum	`\[ \inclinatum \auctum b \]`
f	Punctum Inclinatum Parvum	`\[ \inclinatum \deminutum b \]`
g	Virga	`\[ \virga b \]`
h	Stropha	`\[ \stropha b \]`
i	Stropha Aucta	`\[ \stropha \auctum b \]`
j	Oriscus	`\[ \oriscus b \]`
k	Clivis vel Flexa	`\[ b \flexa g \]`
l	Clivis Aucta Descendens	`\[ b \flexa \auctum \descendens g \]`
m	Clivis Aucta Ascendens	`\[ b \flexa \auctum \ascendens g \]`
n	Cephalicus	`\[ b \flexa \deminutum g \]`
o	Podatus vel Pes	`\[ g \pes b \]`
p	Pes Auctus Descendens	`\[ g \pes \auctum \descendens b \]`
q	Pes Auctus Ascendens	`\[ g \pes \auctum \ascendens b \]`
r	Epiphonus	`\[ g \pes \deminutum b \]`
s	Pes Quassus	`\[ \oriscus g \pes \virga b \]`
t	Pes Quassus Auctus Descendens	`\[ \oriscus g \pes \auctum \descendens b \]`
u	Quilisma Pes	`\[ \quilisma g \pes b \]`
v	Quilisma Pes Auctus Descendens	`\[ \quilisma g \pes \auctum \descendens b \]`
w	Pes Initio Debilis	`\[ \deminutum g \pes b \]`

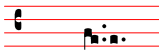
x	Pes Auctus Descendens Initio Debilis	`\[ \deminutum g \pes \auctum \descendens b \]`
y	Torculus	`\[ a \pes b \flexa g \]`
z	Torculus Auctus Descendens	`\[ a \pes b \flexa \auctum \descendens g \]`
A	Torculus Deminutus	`\[ a \pes b \flexa \deminutum g \]`
B	Torculus Initio Debilis	`\[ \deminutum a \pes b \flexa g \]`
C	Torculus Auctus Descendens Initio Debilis	`\[ \deminutum a \pes b \flexa \auctum \descendens g \]`
D	Torculus Deminutus Initio Debilis	`\[ \deminutum a \pes b \flexa \deminutum g \]`
E	Porrectus	`\[ a \flexa g \pes b \]`
F	Porrectus Auctus Descendens	`\[ a \flexa g \pes \auctum \descendens b \]`
G	Porrectus Deminutus	`\[ a \flexa g \pes \deminutum b \]`
H	Climacus	`\[ \virga b \inclinatum a \inclinatum g \]`
I	Climacus Auctus	`\[ \virga b \inclinatum a \inclinatum \auctum g \]`
J	Climacus Deminutus	`\[ \virga b \inclinatum a \inclinatum \deminutum g \]`
K	Scandicus	`\[ g \pes a \virga b \]`
L	Scandicus Auctus Descendens	`\[ g \pes a \pes \auctum \descendens b \]`
M	Scandicus Deminutus	`\[ g \pes a \pes \deminutum b \]`
N	Salicus	`\[ g \oriscus a \pes \virga b \]`
O	Salicus Auctus Descendens	`\[ g \oriscus a \pes \auctum \descendens b \]`
P	Trigonus	`\[ \stropha b \stropha b \stropha a \]`

The ligatures listed above mainly serve as a limited, but still representative pool of Gregorian ligature examples. Virtually, within the ligature delimiters `\[` and `\]`, any number of heads may be accumulated to form a single ligature, and head prefixes like `\pes`, `\flexa`, `\virga`, `\inclinatum`, etc. may be mixed in as desired. The use of the set of rules that underlies the construction of the ligatures in the above table is accordingly extrapolated. This way, infinitely many different ligatures can be created.

Augmentum dots, also called *morae*, are added with the music function `\augmentum`. Note that `\augmentum` is implemented as a unary music function rather than as head prefix. It applies to the immediately following music expression only. That is, `\augmentum \virga c` will have no visible effect. Instead, say `\virga \augmentum c` or `\augmentum {\virga c}`. Also note that you can say `\augmentum {a g}` as a shortcut for `\augmentum a \augmentum g`.

```
\include "gregorian-init.ly"
\score {
  \new VaticanaVoice {
```

```
    \[ \augmentum a \flexa \augmentum g \]
    \augmentum g
  }
}
```



Predefined commands

The following head prefixes are supported

`\virga`, `\stropha`, `\inclinatum`, `\auctum`, `\descendens`, `\ascendens`, `\oriscus`, `\quilisma`, `\deminutum`, `\cavum`, `\linea`.

Head prefixes can be accumulated, though restrictions apply. For example, either `\descendens` or `\ascendens` can be applied to a head, but not both to the same head.

Two adjacent heads can be tied together with the `\pes` and `\flexa` infix commands for a rising and falling line of melody, respectively.

Use the unary music function `\augmentum` to add augmentum dots.

Known issues and warnings

When an `\augmentum` dot appears at the end of the last staff within a ligature, it is sometimes vertically placed wrong. As a workaround, add an additional skip note (e.g. `s8`) as last note of the staff.

`\augmentum` should be implemented as a head prefix rather than a unary music function, such that `\augmentum` can be intermixed with head prefixes in arbitrary order.

2.8.3 Pre-defined contexts

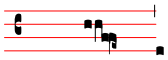
2.8.3.1 Gregorian Chant contexts

The predefined `VaticanaVoiceContext` and `VaticanaStaffContext` can be used to engrave a piece of Gregorian Chant in the style of the Editio Vaticana. These contexts initialize all relevant context properties and grob properties to proper values, so you can immediately go ahead entering the chant, as the following excerpt demonstrates

```
\include "gregorian-init.ly"
\score {
  <<
    \new VaticanaVoice = "cantus" {
      \[ c'\melisma c' \flexa a \]
      \[ a \flexa \deminutum g\melismaEnd \]
      f \divisioMinima
      \[ f\melisma \pes a c' c' \pes d'\melismaEnd \]
      c' \divisioMinima \break
      \[ c'\melisma c' \flexa a \]
      \[ a \flexa \deminutum g\melismaEnd \] f \divisioMinima
    }
    \new Lyrics \lyricsto "cantus" {
```

```
        San- ctus, San- ctus, San- ctus
      }
    >>
}
```

San- ctus, San- ctus,

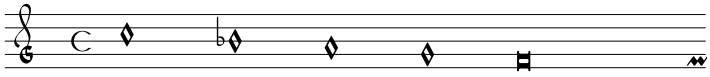


San- ctus

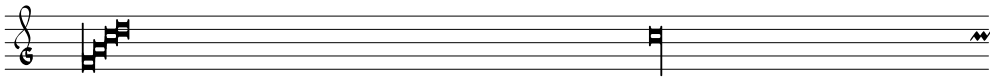
2.8.3.2 Mensural contexts

The predefined `MensuralVoiceContext` and `MensuralStaffContext` can be used to engrave a piece in mensural style. These contexts initialize all relevant context properties and grob properties to proper values, so you can immediately go ahead entering the chant, as the following excerpt demonstrates

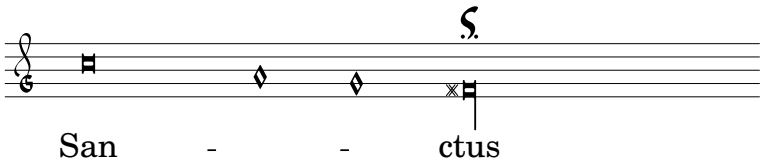
```
\score {
  <<
    \new MensuralVoice = "discantus" \transpose c c' {
      \override Score.BarNumber #'transparent = ##t {
        c'1\melisma bes a g\melismaEnd
        f\breve
        \[ f1\melisma a c'\breve d'\melismaEnd \]
        c'\longa
        c'\breve\melisma a1 g1\melismaEnd
        fis\longa^\signumcongruentiae
      }
    }
    \new Lyrics \lyricsto "discantus" {
      San -- ctus, San -- ctus, San -- ctus
    }
  >>
}
```



San - - ctus,



San - - - - ctus,

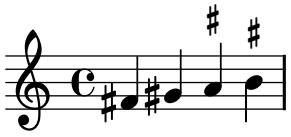


2.8.4 Musica ficta accidentals

In European music from before about 1600, singers were often expected to chromatically alter notes at their own initiative. This is called 'Musica Ficta'. In modern transcriptions, these accidentals are usually printed over the note.

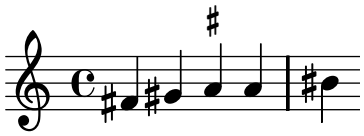
Support for such suggested accidentals is included, and can be switched on by setting `suggestAccidentals` to true.

```
fis gis
\set suggestAccidentals = ##t
ais bis
```



This will treat *every* subsequent accidentals as *musica ficta* until it is unset with `\set suggestAccidentals = ##f`. A more convenient way is to use `\once`:

```
fis gis
\once \set suggestAccidentals = ##t
ais ais bis
```



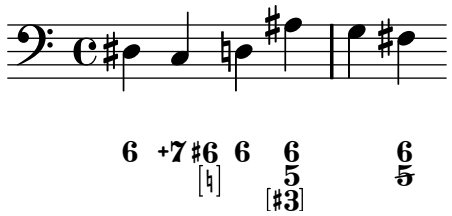
See also

Internals Reference: `Accidental_engraver` engraver and the `AccidentalSuggestion` object.

2.8.5 Figured bass

LilyPond has support for figured bass

```
<<
  \new Voice { \clef bass dis4 c d ais g fis}
  \new FiguredBass \figuremode {
    < 6 >4 < 7\+ >8 < 6+ [_!] >
    < 6 >4 <6 5 [3+] >
    < _ >4 < 6 5/>4
  }
>>
```



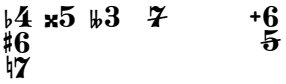
The support for figured bass consists of two parts: there is an input mode, introduced by `\figuremode`, where you can enter bass figures as numbers, and there is a context called `FiguredBass` that takes care of making `BassFigure` objects.

In figures input mode, a group of bass figures is delimited by `<` and `>`. The duration is entered after the `>`

```
<4 6>
```

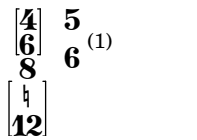
Accidentals are added when you append `-`, `!`, and `+` to the numbers. A plus sign is added when you append `\+`, and diminished fifths and sevenths can be obtained with `5/` and `7/`.

```
<4- 6+ 7!> <5++> <3--> <7/> r <6\+ 5/>
```



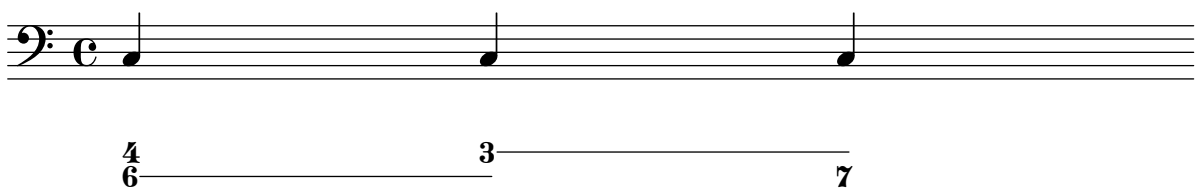
Spaces may be inserted by using `_`. Brackets are introduced with `[` and `]`. You can also include text strings and text markups, see Section B.6 [Overview of text markup commands], page 306.

```
< [4 6] 8 [_! 12] > < 5 \markup { \number 6 \super (1) } >
```



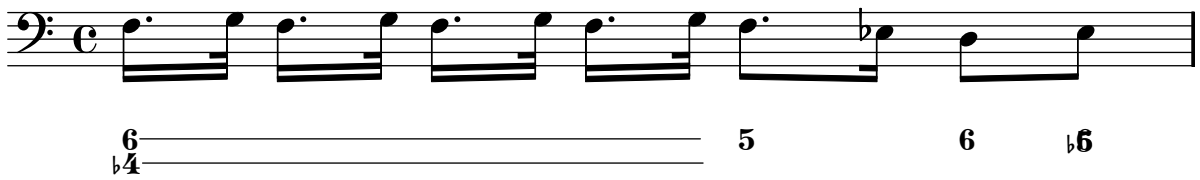
It is also possible to use continuation lines for repeated figures,

```
<<
  \new Staff {
    \clef bass
    c4 c c
  }
  \figures {
    \set useBassFigureExtenders = ##t
    <4 6> <3 6> <3 7>
  }
>>
```



In this case, the extender lines always replace existing figures.

The `FiguredBass` context doesn't pay attention to the actual bass line. As a consequence, you may have to insert extra figures to get extender lines below all notes, and you may have to add `\!` to avoid getting an extender line, e.g.



When using continuation lines, common figures are always put in the same vertical position. When this is unwanted, you can insert a rest with `r`. The rest will clear any previous alignment. For example, you can write

```
<4 6>8 r8
```

instead of

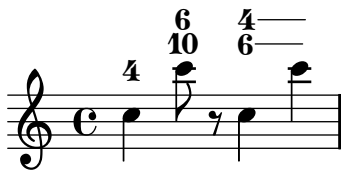
```
<4 6>4
```

Accidentals and plus signs can appear before or after the numbers, depending on the `figuredBassAlterationDirection` and `figuredBassPlusDirection` properties

Although the support for figured bass may superficially resemble chord support, it is much simpler. The `\figuremode` mode simply stores the numbers and `FiguredBass` context prints them as entered. There is no conversion to pitches and no realizations of the bass are played in the MIDI file.

Internally, the code produces markup texts. You can use any of the markup text properties to override formatting. For example, the vertical spacing of the figures may be set with `baseline-skip`.

Figured bass can also be added to `Staff` contexts directly. In this case, their vertical position is adjusted automatically.



Selected Snippets

By default, this method produces figures above the notes. To get figures below the notes, use

```
\override Staff.BassFigureAlignmentPositioning #'direction = #DOWN
```

Known issues and warnings

When using figured bass above the staff with extender lines and `implicitBassFigures` the lines may become swapped around. Maintaining order consistently will be impossible when multiple figures have overlapping extender lines. To avoid this problem, please use `stacking-dir` on `BassFigureAlignment`.

See also

Internals Reference: `NewBassFigure`, `BassFigureAlignment`, `BassFigureLine`, `BassFigureBracket`, and `BassFigureContinuation` objects and `FiguredBass` context.

3 Input syntax

This section deals with general lilypond input syntax issues, rather than specific notation.

FIXME: don't complain about anything in this chapter. It's still under heavy development.

3.1 Input files

The main format of input for LilyPond are text files. By convention, these files end with `.ly`.

3.1.1 File structure

A `.ly` file contains any number of toplevel expressions, where a toplevel expression is one of the following

- An output definition, such as `\paper`, `\midi`, and `\layout`. Such a definition at the toplevel changes the default settings for the block entered.
- A direct scheme expression, such as `#(set-default-paper-size "a7" 'landscape)` or `#(ly:set-option 'point-and-click #f)`.
- A `\header` block. This sets the global header block. This is the block containing the definitions for book-wide settings, like composer, title, etc.
- A `\score` block. This score will be collected with other toplevel scores, and combined as a single `\book`.

  This behavior can be changed by setting the variable `toplevel-score-handler` at toplevel. The default handler is defined in the init file '`scm/lily.scm`'.

  The `\score` must begin with a music expression, and may contain only one music expression.
- A `\book` block logically combines multiple movements (i.e., multiple `\score` blocks) in one document. If there are a number of `\scores`, one output file will be created for each `\book` block, in which all corresponding movements are concatenated. The only reason to explicitly specify `\book` blocks in a `.ly` file is if you wish multiple output files from a single input file. One exception is within lilypond-book documents, where you explicitly have to add a `\book` block if you want more than a single `\score` or `\markup` in the same example.

  This behavior can be changed by setting the variable `toplevel-book-handler` at toplevel. The default handler is defined in the init file '`scm/lily.scm`'.
- A compound music expression, such as

  ```
  { c'4 d' e'2 }
  ```

  This will add the piece in a `\score` and format it in a single book together with all other toplevel `\score`s and music expressions. In other words, a file containing only the above music expression will be translated into

  ```
  \book {
    \score {
      \new Staff {
        \new Voice {
          { c'4 d' e'2 }
        }
      }
    }
  \layout { }
  \header { }
  }
  ```

  This behavior can be changed by setting the variable `toplevel-music-handler` at toplevel. The default handler is defined in the init file '`scm/lily.scm`'.

- A markup text, a verse for example

```
\markup {
   2.  The first line verse two.
}
```

  Markup texts are rendered above, between or below the scores or music expressions, wherever they appear.

- An variable, such as

```
foo = { c4 d e d }
```

  This can be used later on in the file by entering `\foo`. The name of an variable should have alphabetic characters only; no numbers, underscores or dashes.

The following example shows three things that may be entered at toplevel

```
\layout {
  % movements are non-justified by default
  ragged-right = ##t
}

\header {
   title = "Do-re-mi"
}

{ c'4 d' e2 }
```

At any point in a file, any of the following lexical instructions can be entered:

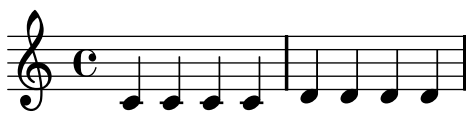
- `\version`
- `\include`
- `\sourcefilename`
- `\sourcefileline`

3.1.2 A single music expression

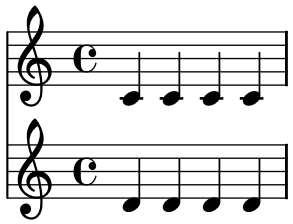
A `\score` must contain a single music expression. However, this music expression may be of any size. Recall that music expressions may be included inside other expressions to form larger expressions. All of these examples are single music expressions; note the curly braces `{ }` or angle brackets `<< >>` at the beginning and ending of the music.

```
{ c'4 c' c' c' }
```

```
{
  { c'4 c' c' c'}
  { d'4 d' d' d'}
}
```



```
<<
  \new Staff { c'4 c' c' c' }
  \new Staff { d'4 d' d' d' }
>>
```



```
{
  \new GrandStaff <<
    \new StaffGroup <<
      \new Staff { \flute }
      \new Staff { \oboe }
    >>
    \new StaffGroup <<
      \new Staff { \violinI }
      \new Staff { \violinII }
    >>
  >>
}
```

3.1.3 Multiple scores in a book

A document may contain multiple pieces of music and texts. Examples of these are an etude book, or an orchestral part with multiple movements. Each movement is entered with a `\score` block,

```
\score {
  ..music..
}
```

and texts are entered with a `\markup` block,

```
\markup {
  ..text..
}
```

All the movements and texts which appear in the same `.ly` file will normally be typeset in the form of a single output file.

```
\score {
  ..
}
\markup {
  ..
}
\score {
  ..
}
```

However, if you want multiple output files from the same `.ly` file, then you can add multiple `\book` blocks, where each such `\book` block will result in a separate output. If you do not specify any `\book` block in the file, LilyPond will implicitly treat the full file as a single `\book` block, see Section 3.1.1 [File structure], page 189. One important exception is within lilypond-book documents, where you explicitly have to add a `\book` block, otherwise only the first `\score` or `\markup` will appear in the output.

The header for each piece of music can be put inside the `\score` block. The `piece` name from the header will be printed before each movement. The title for the entire book can be put inside the `\book`, but if it is not present, the `\header` which is at the top of the file is inserted.

```
\header {
```

```
    title = "Eight miniatures"
    composer = "Igor Stravinsky"
  }
  \score {
    ...
    \header { piece = "Romanze" }
  }
  \markup {
     ..text of second verse..
  }
  \markup {
     ..text of third verse..
  }
  \score {
    ...
    \header { piece = "Menuetto" }
  }
```

3.1.4 Extracting fragments of notation

It is possible to quote small fragments of a large score directly from the output. This can be compared to clipping a piece of a paper score with scissors.

This is done by defininining the measures that need to be cut out separately. For example, including the following definition

```
\layout {
  clip-regions
  = #(list
      (cons
       (make-rhythmic-location 5 1 2)
       (make-rhythmic-location 7 3 4)))
}
```

will extract a fragment starting halfway the fifth measure, ending in the seventh measure. The meaning of `5 1 2` is: after a 1/2 note in measure 5, and `7 3 4` after 3 quarter notes in measure 7.

More clip regions can be defined by adding more pairs of rhythmic-locations to the list.

In order to use this feature, LilyPond must be invoked with `-dclip-systems`. The clips are output as EPS files, and are converted to PDF and PNG if these formats are switched on as well.

For more information on output formats, see program usage manual, ⟨undefined⟩ [Invoking lilypond], page ⟨undefined⟩ .

See also

Examples:

3.1.5 Including LilyPond files

A large project may be split up into separate files. To refer to another file, use

```
\include "otherfile.ly"
```

The line `\include "file.ly"` is equivalent to pasting the contents of file.ly into the current file at the place where you have the \include. For example, for a large project you might write

separate files for each instrument part and create a 'full score' file which brings together the individual instrument files.

The initialization of LilyPond is done in a number of files that are included by default when you start the program, normally transparent to the user. Run lilypond –verbose to see a list of paths and files that Lily finds.

Files placed in directory '`PATH/TO/share/lilypond/VERSION/ly/`' (where VERSION is in the form '2.6.1') are on the path and available to `\include`. Files in the current working directory are available to \include, but a file of the same name in LilyPond's installation takes precedence. Files are available to \include from directories in the search path specified as an option when invoking `lilypond --include=DIR` which adds DIR to the search path.

The `\include` statement can use full path information, but with the Unix convention `/` rather than the DOS/Windows \. For example, if '`stuff.ly`' is located one directory higher than the current working directory, use

```
\include "../stuff.ly"
```

3.1.6 Text encoding

LilyPond uses the Pango library to format multi-lingual texts, and does not perform any input-encoding conversions. This means that any text, be it title, lyric text, or musical instruction containing non-ASCII characters, must be utf-8. The easiest way to enter such text is by using a Unicode-aware editor and saving the file with utf-8 encoding. Most popular modern editors have utf-8 support, for example, vim, Emacs, jEdit, and GEdit do.

To use a Unicode escape sequence, use

```
#(ly:export (ly:wide-char->utf-8 #x2014))
```

See also

3.1.7 Different editions from one source

The `\tag` command marks music expressions with a name. These tagged expressions can be filtered out later. With this mechanism it is possible to make different versions of the same music source.

In the following example, we see two versions of a piece of music, one for the full score, and one with cue notes for the instrumental part

```
c1
<<
  \tag #'part <<
    R1 \\
    {
      \set fontSize = #-1
      c4_"cue" f2 g4 }
  >>
  \tag #'score R1
>>
c1
```

The same can be applied to articulations, texts, etc.: they are made by prepending

```
-\tag #your-tag
```

to an articulation, for example,

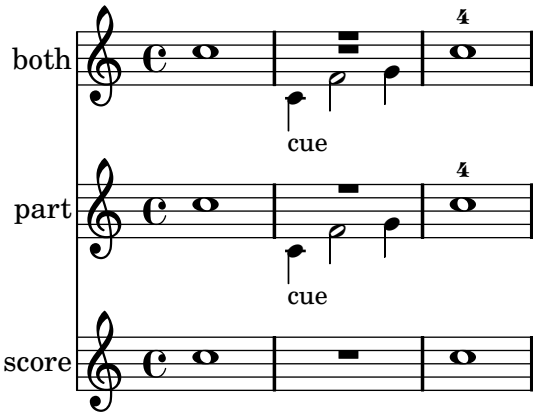
```
c1-\tag #'part ^4
```

This defines a note with a conditional fingering indication.

By applying the `\keepWithTag` and `\removeWithTag` commands, tagged expressions can be filtered. For example,

```
<<
  the music
  \keepWithTag #'score the music
  \keepWithTag #'part the music
>>
```

would yield



The arguments of the `\tag` command should be a symbol (such as `#'score` or `#'part`), followed by a music expression. It is possible to put multiple tags on a piece of music with multiple `\tag` entries,

```
\tag #'original-part \tag #'transposed-part ...
```

See also

Examples:

Known issues and warnings

Multiple rests are not merged if you create the score with both tagged sections.

3.2 Common syntax issues TODO name?

3.2.1 Controlling direction

TODO: everything

By default, lilypnod does a pretty jazz'n job of picking directions. But in some cases, it may be desirable to force a direction.

\-

^ _

Also cover #UP #DOWN #LEFT #RIGHT.

Maybe rename section to "directions".

Also mention \override Foo #'direction = #'DOWN.

also mention the typical \fooDown, \fooNeutral predefined commands.

3.2.2 Distances and measurements MAYBE MOVE

DISCUSS after working on other sections.

TODO: staff spaces, #UP #DOWN #LEFT #RIGHT. Maybe move into tweaks?

3.3 Other stuffs TODO move?

3.3.1 Displaying LilyPond notation

Displaying a music expression in LilyPond notation can be done using the music function `\displayLilyMusic`. For example,

```
{
  \displayLilyMusic \transpose c a, { c e g a bes }
}
```

will display

```
{ a, cis e fis g }
```

By default, LilyPond will print these messages to the console along with all the other messages. To split up these messages and save the results of `\display{STUFF}`, redirect the output to a file.

```
lilypond file.ly >display.txt
```

3.3.2 Skipping corrected music

When entering or copying music, usually only the music near the end (where you are adding notes) is interesting to view and correct. To speed up this correction process, it is possible to skip typesetting of all but the last few measures. This is achieved by putting

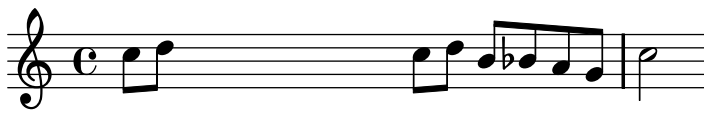
```
showLastLength = R1*5
\score { ... }
```

in your source file. This will render only the last 5 measures (assuming 4/4 time signature) of every `\score` in the input file. For longer pieces, rendering only a small part is often an order of magnitude quicker than rendering it completely

Skipping parts of a score can be controlled in a more fine-grained fashion with the property `Score.skipTypesetting`. When it is set, no typesetting is performed at all.

This property is also used to control output to the MIDI file. Note that it skips all events, including tempo and instrument changes. You have been warned.

```
\relative c'' {
  c8 d
  \set Score.skipTypesetting = ##t
  e e e e e e e
  \set Score.skipTypesetting = ##f
  c d b bes a g c2 }
```



In polyphonic music, `Score.skipTypesetting` will affect all voices and staves, saving even more time.

3.3.3 context list FIXME

>> > > - list of contexts: my *danger unmaintainable* >> > > alarm just went off. I'm

I knew it would... And leaving out some of them is perfectly fine with me. I do think that a list like this, with the main contexts and a brief description of what they do (perhaps also with a note about what default behaviour is associated with each of them, but this may be unmanageable), should be there, and then we could simply list the remaining ones without further explanation and with links to the IR.

The Master Of All Contexts ==========

* Score This is the top level notation context. No other context can contain a Score context. This context handles the administration of time signatures. It also makes sure that items such as clefs, time signatures, and key-signatures are aligned across staves. You cannot explicitly instantiate a Score context (since it is not contained in any other context). It is instantiated automatically when an output definition (a \score or \layout block) is processed. (it should also be made clear somewhere what the difference is between \score and \Score).

Top-level contexts: Staff containers ========== * StaffGroup Groups staves while adding a bracket on the left side, grouping the staves together. The bar lines of the contained staves are connected vertically. StaffGroup only consists of a collection of staves, with a bracket in front and spanning bar lines. * ChoirStaff Identical to StaffGroup except that the contained staves are not connected vertically. * GrandStaff A group of staves, with a brace on the left side, grouping the staves together. The bar lines of the contained staves are connected vertically. * PianoStaff Just like GrandStaff but with a forced distance between the staves, so cross staff beaming and slurring can be used. * DrumStaff Handles typesetting for percussion. Can contain DrumVoice * InnerStaffGroup * InnerChoirStaff

Staff-level contexts ========== * Staff Handles clefs, bar lines, keys, accidentals. It can contain Voice contexts. * RhythmicStaff Like Staff but for printing rhythms. Pitches are ignored; the notes are printed on one line. * TabStaff Context for generating tablature. By default lays the music expression out as a guitar tablature, printed on six lines. * VaticanaStaff Same as Staff, except that it is accommodated for typesetting a piece in gregorian style. * MensuralStaff Same as Staff, except that it is accommodated for typesetting a piece in mensural style.

Voice-level (bottom) contexts ========== What is generated by default here? The voice-level contexts initiate certain properties and start engravers.

* Voice Corresponds to a voice on a staff. This context handles the conversion of dynamic signs, stems, beams, super- and subscripts, slurs, ties, and rests. You have to instantiate this explicitly if you want to have multiple voices on the same staff. Bottom context. * VaticanaVoice Same as Voice, except that it is accommodated for typesetting a piece in gregorian style. * MensuralVoice Same as Voice, except that it is accommodated for typesetting a piece in mensural style. * Lyrics Corresponds to a voice with lyrics. Handles the printing of a single line of lyrics. Bottom context. * DrumVoice A voice on a percussion staff. * FiguredBass

* ChordNames Typesets chord names. This context is a 'bottom' context; it cannot contain other contexts.

———— Then the following, which I don't know what to do with:

* TabVoice * GregorianTranscriptionVoice * GregorianTranscriptionStaff

* FretBoards Engraves fretboards from chords. Not easy... Not documented. * NoteNames

* CueVoice Not documented * Global Hard coded entry point for LilyPond. Cannot be tuned. * Devnull Silently discards all musical information given to this context.

3.3.4 another thing FIXME

Another thing that is needed, is an overview of the various naming conventions:

scheme functions: lowercase-with-hyphens (incl. one-word names) scheme functions: ly:plus-scheme-style music events, music classes and music properties: as-scheme-functions Grob interfaces: scheme-style backend properties: scheme-style (but X and Y!) contexts (and MusicExpressions and grobs): Capitalized or CamelCase context properties: lowercaseFollowedByCamelCase engravers: Capitalized_followed_by_lowercase_and_with_underscores

Which of these are conventions and which are rules? Which are rules of the underlying language, and which are LP-specific?

3.3.5 Input modes FIXME

\notemode

\notemode turns the front end of LilyPond into note mode (which is the default parsing mode). It's certainly useful in certain situations, for example if you are in \lyricmode or \chordmode or ... and want to insert something that only can be done with \notemode syntax.

See for example http://lists.gnu.org/archive/html/lilypond-user/2007-03/msg00418.html http://lists.gnu.org/archive/html/lilypond-user/2007-03/msg00218.html http://lists.gnu.org/archive/html/lilypond-user/2006-12/msg00236.html http://lists.gnu.org/archive/html/lilypond-user/2006-11/msg00061.html

\chords \drums \fretmode ?

4 Non-musical notation

This section deals with general lilypond issues, rather than specific notation.

4.1 Titles and headers

Almost all printed music includes a title and the composer's name; some pieces include a lot more information.

4.1.1 Creating titles

Titles are created for each `\score` block, as well as for the full input file (or `\book` block).

The contents of the titles are taken from the `\header` blocks. The header block for a book supports the following

`dedication`
: The dedicatee of the music, centered at the top of the first page.

`title`
: The title of the music, centered just below the dedication.

`subtitle`
: Subtitle, centered below the title.

`subsubtitle`
: Subsubtitle, centered below the subtitle.

`poet`
: Name of the poet, flush-left below the subtitle.

`composer`
: Name of the composer, flush-right below the subtitle.

`meter`
: Meter string, flush-left below the poet.

`opus`
: Name of the opus, flush-right below the composer.

`arranger`
: Name of the arranger, flush-right below the opus.

`instrument`
: Name of the instrument, centered below the arranger. Also centered at the top of pages (other than the first page).

`piece`
: Name of the piece, flush-left below the instrument.

`breakbefore`
: This forces the title to start on a new page (set to ##t or ##f).

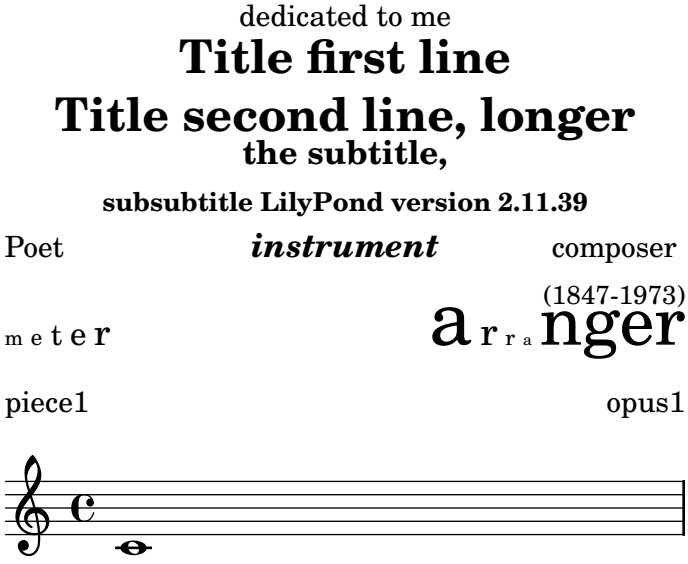
`copyright`
: Copyright notice, centered at the bottom of the first page. To insert the copyright symbol, see Section 3.1.6 [Text encoding], page 193.

`tagline`
: Centered at the bottom of the last page.

Here is a demonstration of the fields available. Note that you may use any Section 1.8.2 [Text markup], page 119, commands in the header.

```
\paper {
  line-width = 9.0\cm
  paper-height = 10.0\cm
}

\book {
  \header {
    dedication = "dedicated to me"
    title = \markup \center-align { "Title first line" "Title second line,
```

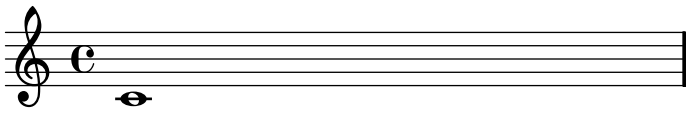
```
longer" }
    subtitle = "the subtitle,"
    subsubtitle = #(string-append "subsubtitle LilyPond version "
(lilypond-version))
    poet = "Poet"
    composer =  \markup \center-align { "composer" \small "(1847-1973)" }
    texttranslator = "Text Translator"
    meter = \markup { \teeny "m" \tiny "e" \normalsize "t" \large "e" \huge
"r" }
    arranger = \markup { \fontsize #8.5 "a" \fontsize #2.5 "r" \fontsize
#-2.5 "r" \fontsize #-5.3 "a" \fontsize #7.5 "nger" }
    instrument = \markup \bold \italic "instrument"
    piece = "Piece"
  }

  \score {
    { c'1 }
    \header {
      piece = "piece1"
      opus = "opus1"
    }
  }
  \markup {
      and now...
  }
  \score {
    { c'1 }
    \header {
      piece = "piece2"
      opus = "opus2"
    }
  }
}
```



2 ***instrument***

and now...

piece2 opus2



Music engraving by LilyPond 2.11.39—www.lilypond.org

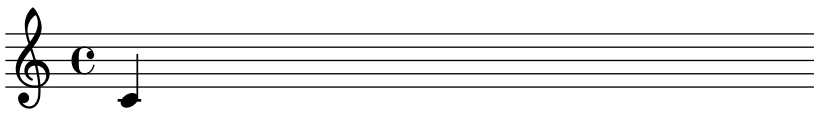
As demonstrated before, you can use multiple `\header` blocks. When same fields appear in different blocks, the latter is used. Here is a short example.

```
\header {
  composer = "Composer"
}
\header {
  piece = "Piece"
}
\score {
  \new Staff { c'4 }
  \header {
    piece = "New piece"  % overwrite previous one
  }
}
```

If you define the `\header` inside the `\score` block, then normally only the `piece` and `opus` headers will be printed. Note that the music expression must come before the `\header`.

```
\score {
  { c'4 }
  \header {
    title = "title"  % not printed
    piece = "piece"
    opus = "opus"
  }
}
```

piece opus



You may change this behavior (and print all the headers when defining `\header` inside `\score`) by using

```
\paper{
  printallheaders=##t
}
```

The default footer is empty, except for the first page, where the `copyright` field from `\header` is inserted, and the last page, where `tagline` from `\header` is added. The default tagline is "Music engraving by LilyPond (*version*)".[1]

Headers may be completely removed by setting them to false.

```
\header {
  tagline = ##f
  composer = ##f
}
```

4.1.2 Custom titles

A more advanced option is to change the definitions of the following variables in the `\paper` block. The init file '`ly/titling-init.ly`' lists the default layout.

`bookTitleMarkup`
: This is the title added at the top of the entire output document. Typically, it has the composer and the title of the piece

`scoreTitleMarkup`
: This is the title put over a `\score` block. Typically, it has the name of the movement (`piece` field).

`oddHeaderMarkup`
: This is the page header for odd-numbered pages.

`evenHeaderMarkup`
: This is the page header for even-numbered pages. If unspecified, the odd header is used instead.

  By default, headers are defined such that the page number is on the outside edge, and the instrument is centered.

`oddFooterMarkup`
: This is the page footer for odd-numbered pages.

`evenFooterMarkup`
: This is the page footer for even-numbered pages. If unspecified, the odd header is used instead.

  By default, the footer has the copyright notice on the first, and the tagline on the last page.

The following definition will put the title flush left, and the composer flush right on a single line.

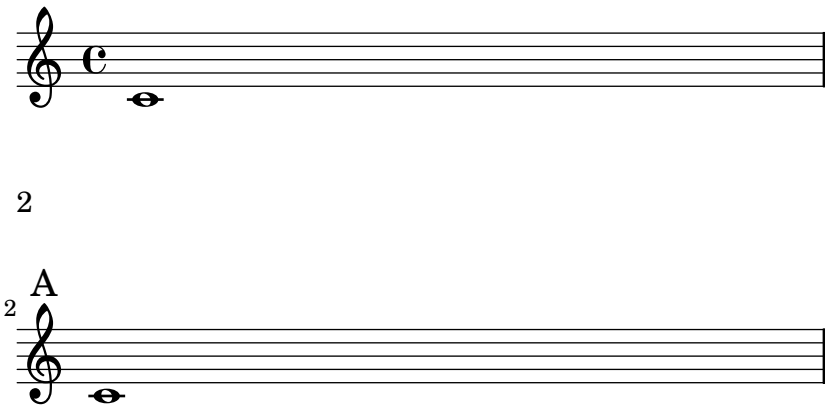
```
\paper {
  bookTitleMarkup = \markup {
   \fill-line {
     \fromproperty #'header:title
     \fromproperty #'header:composer
   }
  }
}
```

[1] Nicely printed parts are good PR for us, so please leave the tagline if you can.

4.1.3 Reference to page numbers

A particular place of a score can be marked using the `\label` command, either at top-level or inside music. This label can then be refered to in a markup, to get the number of the page where the marked point is placed, using the `\page-ref` markup command.

```
\header { tagline = ##f }
\book {
  \label #'firstScore
  \score {
    {
      c'1
      \pageBreak \mark A \label #'markA
      c'
    }
  }

  \markup { The first score begins on page \page-ref #'firstScore "0" "?" }
  \markup { Mark A is on page \page-ref #'markA "0" "?" }
}
```



The first score begins on page 1

Mark A is on page 2

The `\page-ref` markup command takes three arguments:

1. the label, a scheme symbol, eg. `#'firstScore`;
2. a markup that will be used as a gauge to estimate the dimensions of the markup;
3. a markup that will be used in place of the page number if the label is not known;

The reason why a gauge is needed is that, at the time markups are interpreted, the page breaking has not yet occured, so the page numbers are not yet known. To work around this issue, the actual markup interpretation is delayed to a later time; however, the dimensions of the markup have to be known before, so a gauge is used to decide these dimensions. If the book has between 10 and 99 pages, it may be `"00"`, ie. a two digit number.

Predefined commands

`\label` `\page-ref`

4.1.4 Table of contents

A table of contents is included using the `\markuplines` `\table-of-contents` command. The elements which should appear in the table of contents are entered with the `\tocItem` command, which may be used either at top-level, or inside a music expression.

```
\markuplines \table-of-contents
\pageBreak

\tocItem \markup "First score"
\score {
  {
    c'  % ...
    \tocItem \markup "Some particular point in the first score"
    d'  % ...
  }
}

\tocItem \markup "Second score"
\score {
  {
    e' % ...
  }
}
```

The markups which are used to format the table of contents are defined in the `\paper` block. The default ones are `tocTitleMarkup`, for formatting the title of the table, and `tocItemMarkup`, for formatting the toc elements, composed of the element title and page number. These variables may be changed by the user:

```
\paper {
  %% Translate the toc title into French:
  tocTitleMarkup = \markup \huge \column {
    \fill-line { \null "Table des matières" \null }
    \hspace #1
  }
  %% use larfer font size
  tocItemMarkup = \markup \large \fill-line {
    \fromproperty #'toc:text \fromproperty #'toc:page
  }
}
```

Note how the toc element text and page number are refered to in the `tocItemMarkup` definition.

New commands and markups may also be defined to build more elaborated table of contents:

- first, define a new markup variable in the `\paper` block
- then, define a music function which aims at adding a toc element using this markup paper variable.

In the following example, a new style is defined for entering act names in the table of contents of an opera:

```
\paper {
```

```
  tocActMarkup = \markup \large \column {
    \hspace #1
    \fill-line { \null \italic \fromproperty #'toc:text \null }
    \hspace #1
  }
}

tocAct =
#(define-music-function (parser location text) (markup?)
   (add-toc-item! 'tocActMarkup text))
```

Table of Contents

Atto Primo

Coro. Viva il nostro Alcide 1
Cesare. Presti omai l'Egizzia terra 1

Atto Secondo

Sinfonia 1
Cleopatra. V'adoro, pupille, saette d'Amore 1

See also

Init files: ‘`ly/toc-init.ly`’.

Predefined commands

`\table-of-contents` `\tocItem`

4.2 MIDI output

MIDI (Musical Instrument Digital Interface) is a standard for connecting and controlling digital instruments. A MIDI file is a series of notes in a number of tracks. It is not an actual sound file; you need special software to translate between the series of notes and actual sounds.

Pieces of music can be converted to MIDI files, so you can listen to what was entered. This is convenient for checking the music; octaves that are off or accidentals that were mistyped stand out very much when listening to the MIDI output.

Known issues and warnings

Many musically interesting effects, such as swing, articulation, slurring, etc., are not translated to midi.

The midi output allocates a channel for each staff, and one for global settings. Therefore the midi file should not have more than 15 staves (or 14 if you do not use drums). Other staves will remain silent.

Not all midi players correctly handle tempo changes in the midi output. Players that are known to work include timidity.

4.2.1 Creating MIDI files

To create a MIDI from a music piece of music, add a `\midi` block to a score, for example,

```
\score {
  ...music...
   \midi {
     \context {
       \Score
       tempoWholesPerMinute = #(ly:make-moment 72 4)
       }
     }
}
```

The tempo can be specified using the `\tempo` command within the actual music, see Section 1.6.2.1 [Metronome marks], page 98. An alternative, which does not result in a metronome mark in the printed score, is shown in the example above. In this example the tempo of quarter notes is set to 72 beats per minute. This kind of tempo specification can not take dotted note lengths as an argument. In this case, break the dotted notes into smaller units. For example, a tempo of 90 dotted quarter notes per minute can be specified as 270 eighth notes per minute

```
tempoWholesPerMinute = #(ly:make-moment 270 8)
```

If there is a `\midi` command in a `\score`, only MIDI will be produced. When notation is needed too, a `\layout` block must be added

```
\score {
  ...music...
  \midi { }
  \layout { }
}
```

Ties, dynamics, and tempo changes are interpreted. Dynamic marks, crescendi and decrescendi translate into MIDI volume levels. Dynamic marks translate to a fixed fraction of the available MIDI volume range, crescendi and decrescendi make the volume vary linearly between their two extremes. The fractions can be adjusted by `dynamicAbsoluteVolumeFunction` in `Voice` context. For each type of MIDI instrument, a volume range can be defined. This gives a basic equalizer control, which can enhance the quality of the MIDI output remarkably. The equalizer can be controlled by setting `instrumentEqualizer`, or by setting

```
\set Staff.midiMinimumVolume = #0.2
\set Staff.midiMaximumVolume = #0.8
```

To remove dynamics from the MIDI output, insert the following lines in the `\midi{}` section.

```
\midi {
  ...
  \context {
    \Voice
```

```
    \remove "Dynamic_performer"
  }
}
```

Known issues and warnings

Unterminated (de)crescendos will not render properly in the midi file, resulting in silent passages of music. The workaround is to explicitly terminate the (de)crescendo. For example,

```
{ a\< b c d\f }
```

will not work properly but

```
{ a\< b c d\!\f }
```

will.

MIDI output is only created when the `\midi` command is within a `\score` block. If you put it within an explicitly instantiated context (i.e. `\new Score`) the file will fail. To solve this, enclose the `\new Score` and the `\midi` in a `\score` block.

```
\score {
  \new Score { ...notes... }
  \midi
}
```

4.2.2 MIDI block

The MIDI block is analogous to the layout block, but it is somewhat simpler. The `\midi` block is similar to `\layout`. It can contain context definitions.

Context definitions follow precisely the same syntax as within the \layout block. Translation modules for sound are called performers. The contexts for MIDI output are defined in '`ly/performer-init.ly`'.

4.2.3 MIDI instrument names

The MIDI instrument name is set by the `Staff.midiInstrument` property. The instrument name should be chosen from the list in Section B.2 [MIDI instruments], page 287.

```
\set Staff.midiInstrument = "glockenspiel"
...notes...
```

If the selected instrument does not exactly match an instrument from the list of MIDI instruments, the Grand Piano (`"acoustic grand"`) instrument is used.

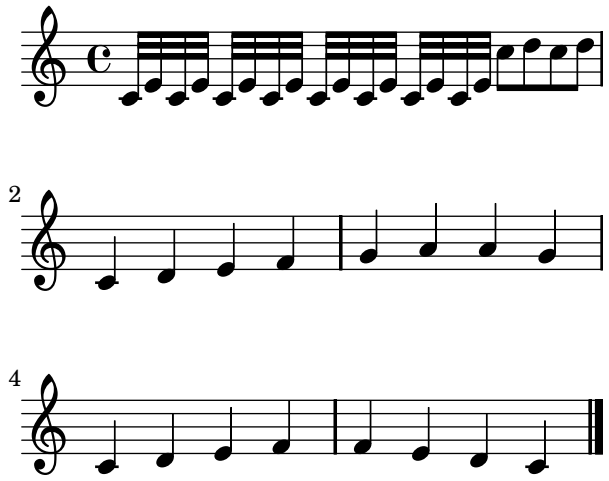
4.2.4 What goes into the MIDI? FIXME

4.2.4.1 Repeats and MIDI

With a little bit of tweaking, all types of repeats can be present in the MIDI output. This is achieved by applying the `\unfoldRepeats` music function. This function changes all repeats to unfold repeats.

```
\unfoldRepeats {
  \repeat tremolo 8 {c'32 e' }
  \repeat percent 2 { c''8 d'' }
  \repeat volta 2 {c'4 d' e' f'}
  \alternative {
    { g' a' a' g' }
    {f' e' d' c' }
  }
```

```
}
\bar "|."
```



When creating a score file using `\unfoldRepeats` for MIDI, it is necessary to make two `\score` blocks: one for MIDI (with unfolded repeats) and one for notation (with volta, tremolo, and percent repeats). For example,

```
\score {
  ..music..
  \layout { .. }
}
\score {
  \unfoldRepeats ..music..
  \midi { .. }
}
```

4.3 other midi

Micro tones are also exported to the MIDI file.

5 Spacing issues

The global paper layout is determined by three factors: the page layout, the line breaks, and the spacing. These all influence each other. The choice of spacing determines how densely each system of music is set. This influences where line breaks are chosen, and thus ultimately, how many pages a piece of music takes.

Globally speaking, this procedure happens in four steps: first, flexible distances ('springs') are chosen, based on durations. All possible line breaking combinations are tried, and a 'badness' score is calculated for each. Then the height of each possible system is estimated. Finally, a page breaking and line breaking combination is chosen so that neither the horizontal nor the vertical spacing is too cramped or stretched.

5.1 Paper and pages

This section deals with the boundaries that define the area that music can be printed inside.

5.1.1 Paper size

To change the paper size, there are two commands,

```
#(set-default-paper-size "a4")
\paper {
  #(set-paper-size "a4")
}
```

The first command sets the size of all pages. The second command sets the size of the pages that the `\paper` block applies to – if the `\paper` block is at the top of the file, then it will apply to all pages. If the `\paper` block is inside a `\book`, then the paper size will only apply to that book.

Support for the following paper sizes are included by default, `a6`, `a5`, `a4`, `a3`, `legal`, `letter`, `11x17` (also known as tabloid).

Extra sizes may be added by editing the definition for `paper-alist` in the initialization file '`scm/paper.scm`'.

If the symbol `landscape` is supplied as an argument to `set-default-paper-size`, the pages will be rotated by 90 degrees, and wider line widths will be set correspondingly.

```
#(set-default-paper-size "a6" 'landscape)
```

Setting the paper size will adjust a number of `\paper` variables (such as margins). To use a particular paper size with altered `\paper` variables, set the paper size before setting the variables.

5.1.2 Page formatting

LilyPond will do page layout, set margins, and add headers and footers to each page.

The default layout responds to the following settings in the `\paper` block.

`first-page-number`
: The value of the page number of the first page. Default is 1.

`print-first-page-number`
: If set to true, will print the page number in the first page. Default is false.

`print-page-number`
: If set to false, page numbers will not be printed. Default is true.

`paper-width`
: The width of the page. The default is taken from the current paper size, see Section 5.1.1 [Paper size], page 208.

`paper-height`
: The height of the page. The default is taken from the current paper size, see Section 5.1.1 [Paper size], page 208.

`top-margin`
: Margin between header and top of the page. Default is 5mm.

`bottom-margin`
: Margin between footer and bottom of the page. Default is 6mm.

`left-margin`
: Margin between the left side of the page and the beginning of the music. Unset by default, which means that the margins is determined based on the `paper-width` and `line-width` to center the score on the paper.

`line-width`
: The length of the systems. Default is `paper-width` minus 20mm.

`head-separation`
: Distance between the top-most music system and the page header. Default is 4mm.

`foot-separation`
: Distance between the bottom-most music system and the page footer. Default is 4mm.

`page-top-space`
: Distance from the top of the printable area to the center of the first staff. This only works for staves which are vertically small. Big staves are set with the top of their bounding box aligned to the top of the printable area. Default is 12mm.

`ragged-bottom`
: If set to true, systems will not be spread vertically across the page. This does not affect the last page. Default is false.

  This should be set to true for pieces that have only two or three systems per page, for example orchestral scores.

`ragged-last-bottom`
: If set to false, systems will be spread vertically to fill the last page. Default is true.

  Pieces that amply fill two pages or more should have this set to true.

`system-count`
: This variable, if set, specifies into how many lines a score should be broken. Unset by default.

`between-system-space`
: This dimensions determines the distance between systems. It is the ideal distance between the center of the bottom staff of one system and the center of the top staff of the next system. Default is 20mm.

  Increasing this will provide a more even appearance of the page at the cost of using more vertical space.

`between-system-padding`
: This dimension is the minimum amount of white space that will always be present between the bottom-most symbol of one system, and the top-most of the next system. Default is 4mm.

Increasing this will put systems whose bounding boxes almost touch farther apart.

`page-breaking-between-system-padding`
: This variable tricks the page breaker into thinking that `between-system-padding` is set to something different than it really is. For example, if this variable is set to something substantially larger than `between-system-padding`, then the page-breaker will put fewer systems on each page.

`horizontal-shift`
: All systems (including titles and system separators) are shifted by this amount to the right. Page markup, such as headers and footers are not affected by this. The purpose of this variable is to make space for instrument names at the left. Default is 0.

`after-title-space`
: Amount of space between the title and the first system. Default is 5mm.

`before-title-space`
: Amount of space between the last system of the previous piece and the title of the next. Default is 10mm.

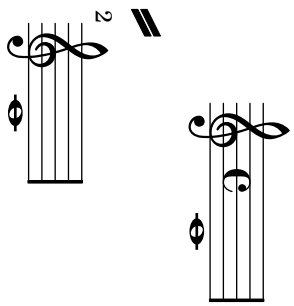
`between-title-space`
: Amount of space between consecutive titles (e.g., the title of the book and the title of a piece). Default is 2mm.

`printallheaders`
: Setting this to #t will print all headers for each \score in the output. Normally only the piece and opus \headers are printed.

`systemSeparatorMarkup`
: This contains a markup object, which will be inserted between systems. This is often used for orchestral scores. Unset by default.

  The markup command `\slashSeparator` is provided as a sensible default, for example

Music engraving by LilyPond 2.11.39—www.lilypond.org



`blank-page-force`
: The penalty for having a blank page in the middle of a score. This is not used by `ly:optimal-breaking` since it will never consider blank pages in the middle of a score. Default value is 10.

`blank-last-page-force`
: The penalty for ending the score on an odd-numbered page. Default value is 0.

`page-spacing-weight`
: The relative importance of page (vertical) spacing and line (horizontal) spacing. High values will make page spacing more important. Default value is 10.

`auto-first-page-number`
: The page breaking algorithm is affected by the first page number being odd or even. If this variable is set to #t, the page breaking algorithm will decide whether to start with an odd or even number. This will result in the first page number remaining as is or being increased by one.

Selected Snippets

The header and footer are created by the functions make-footer and make-header, defined in \paper. The default implementations are in ly/paper-defaults.ly and ly/titling-init.ly.

The page layout itself is done by two functions in the \paper block, page-music-height and page-make-stencil. The former tells the line-breaking algorithm how much space can be spent on a page, the latter creates the actual page given the system to put on it.

You can define paper block values in Scheme. In that case mm, in, pt, and cm are variables defined in paper-defaults.ly with values in millimeters. That is why the value 2 cm must be multiplied in the example

```
\paper {
 #(define bottom-margin (* 2 cm))
}
```

Example:

```
\paper{
  paper-width = 2\cm
  top-margin = 3\cm
  bottom-margin = 3\cm
  ragged-last-bottom = ##t
}
```

This second example centers page numbers at the bottom of every page.

```
\paper {
  print-page-number = ##t
  print-first-page-number = ##t
  oddHeaderMarkup = \markup \fill-line { " " }
  evenHeaderMarkup = \markup \fill-line { " " }
  oddFooterMarkup = \markup { \fill-line {
     \bold \fontsize #3 \on-the-fly #print-page-number-check-first
     \fromproperty #'page:page-number-string } }
  evenFooterMarkup = \markup { \fill-line {
     \bold \fontsize #3 \on-the-fly #print-page-number-check-first
     \fromproperty #'page:page-number-string } }
}
```

You can also define these values in Scheme. In that case `mm`, `in`, `pt`, and `cm` are variables defined in '`paper-defaults.ly`' with values in millimeters. That is why the value must be multiplied in the example

```
\paper {
  #(define bottom-margin (* 2 cm))
}
```

The header and footer are created by the functions `make-footer` and `make-header`, defined in `\paper`. The default implementations are in '`ly/paper-defaults.ly`' and '`ly/titling-init.ly`'.

The page layout itself is done by two functions in the `\paper` block, `page-music-height` and `page-make-stencil`. The former tells the line-breaking algorithm how much space can be spent on a page, the latter creates the actual page given the system to put on it.

Known issues and warnings

The option right-margin is defined but doesn't set the right margin yet. The value for the right margin has to be defined adjusting the values of `left-margin` and `line-width`.

The default page header puts the page number and the `instrument` field from the `\header` block on a line.

The titles (from the `\header{}` section) are treated as a system, so `ragged-bottom` and `ragged-last-bottom` will add space between the titles and the first system of the score.

5.2 Music layout

5.2.1 Setting the staff size

To set the staff size globally for all scores in a file (or in a `book` block, to be precise), use `set-global-staff-size`.

```
#(set-global-staff-size 14)
```

This sets the global default size to 14pt staff height and scales all fonts accordingly.

To set the staff size individually for each score, use

```
\score{
  ...
  \layout{
  #(layout-set-staff-size 15)
  }
}
```

The Feta font provides musical symbols at eight different sizes. Each font is tuned for a different staff size: at a smaller size the font becomes heavier, to match the relatively heavier staff lines. The recommended font sizes are listed in the following table:

font name	staff height (pt)	staff height (mm)	use
feta11	11.22	3.9	pocket scores
feta13	12.60	4.4	
feta14	14.14	5.0	
feta16	15.87	5.6	
feta18	17.82	6.3	song books
feta20	20	7.0	standard parts
feta23	22.45	7.9	
feta26	25.2	8.9	

These fonts are available in any sizes. The context property `fontSize` and the layout property `staff-space` (in `StaffSymbol`) can be used to tune the size for individual staves. The sizes of individual staves are relative to the global size.

See also

This manual: Section 1.7.1.1 [Selecting notation font size], page 105.

Known issues and warnings

`layout-set-staff-size` does not change the distance between the staff lines.

5.2.2 Score layout

While `\paper` contains settings that relate to the page formatting of the whole document, `\layout` contains settings for score-specific layout.

```
\layout {
  indent = 2.0\cm
  \context { \Staff
    \override VerticalAxisGroup #'minimum-Y-extent = #'(-6 . 6)
  }
  \context { \Voice
    \override TextScript #'padding = #1.0
    \override Glissando #'thickness = #3
  }
}
```

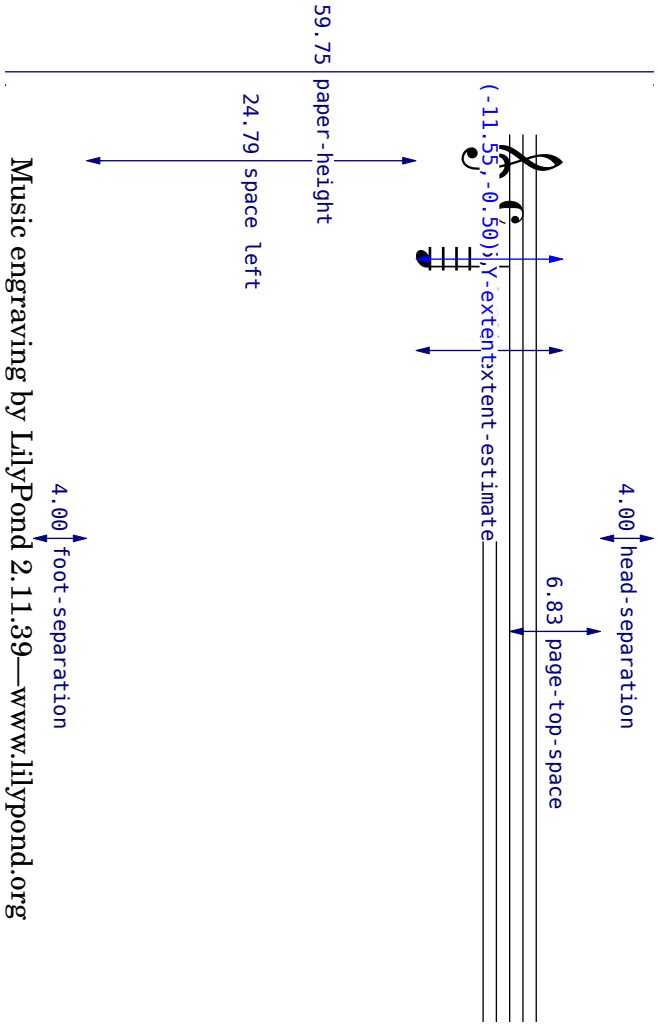
See also

This manual: Section 6.1.6 [Changing context default settings], page 256.

5.3 Displaying spacing

To graphically display the dimensions of vertical properties that may be altered for page formatting, set `annotate-spacing` in the `\paper` block, like this

```
#(set-default-paper-size "a6" 'landscape)

\book {
  \score { { c4 } }
  \paper { annotate-spacing = ##t }
}
```



Some unit dimensions are measured in staff spaces, while others are measured in millimeters. The pairs (a,b) are intervals, where a is the lower edge and b the upper edge of the interval.

5.4 Breaks

5.4.1 Line breaking

Line breaks are normally computed automatically. They are chosen so that lines look neither cramped nor loose, and that consecutive lines have similar density.

Occasionally you might want to override the automatic breaks; you can do this by specifying `\break`. This will force a line break at this point. Line breaks can only occur at places where there are bar lines. If you want to have a line break where there is no bar line, you can force an invisible bar line by entering `\bar ""`. Similarly, `\noBreak` forbids a line break at a point.

For line breaks at regular intervals use `\break` separated by skips and repeated with `\repeat`:

```
<< \repeat unfold 7 {
         s1 \noBreak s1 \noBreak
         s1 \noBreak s1 \break }
   the real music
>>
```

This makes the following 28 measures (assuming 4/4 time) be broken every 4 measures, and only there.

Predefined commands

`\break`, and `\noBreak`.

See also

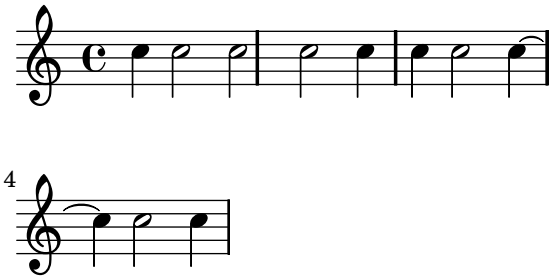
Internals: `LineBreakEvent`.

A linebreaking configuration can be saved as a `.ly` file automatically. This allows vertical alignments to be stretched to fit pages in a second formatting run. This is fairly new and complicated. More details are available in

Known issues and warnings

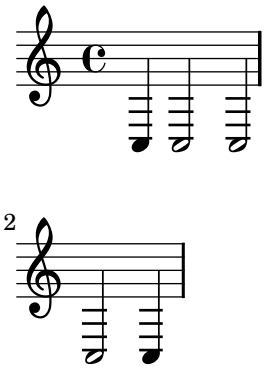
Line breaks can only occur if there is a 'proper' bar line. A note which is hanging over a bar line is not proper, such as

```
c4 c2 << c2 {s4 \break } >>  % this does nothing
c2 c4 |           % a break here would work
c4 c2 c4 ~ \break % as does this break
c4 c2 c4
```



This can be avoided by removing the `Forbid_line_break_engraver`. Note that manually forced line breaks have to be added in parallel with the music.

```
\new Voice \with {
  \remove Forbid_line_break_engraver
} {
  c4 c2 << c2 {s4 \break } >>  % now the break is allowed
  c2 c4
}
```



Similarly, line breaks are normally forbidden when beams cross bar lines. This behavior can be changed by setting `\override Beam #'breakable = ##t`.

5.4.2 Page breaking

The default page breaking may be overriden by inserting `\pageBreak` or `\noPageBreak` commands. These commands are analogous to `\break` and `\noBreak`. They should be inserted at a bar line. These commands force and forbid a page-break from happening. Of course, the `\pageBreak` command also forces a line break.

The `\pageBreak` and `\noPageBreak` commands may also be inserted at top-level, between scores and top-level markups.

Page breaks are computed by the `page-breaking` function. LilyPond provides three algorithms for computing page breaks, `ly:optimal-breaking`, `ly:page-turn-breaking` and `ly:minimal-breaking`. The default is `ly:optimal-breaking`, but the value can be changed in the `\paper` block:

```
\paper{
  #(define page-breaking ly:page-turn-breaking)
}
```

The old page breaking algorithm is called `optimal-page-breaks`. If you are having trouble with the new page breakers, you can enable the old one as a workaround.

Predefined commands

`\pageBreak` `\noPageBreak`

5.4.3 Optimal page breaking

The `ly:optimal-breaking` function is LilyPond's default method of determining page breaks. It attempts to find a page breaking that minimizes cramping and stretching, both horizontally and vertically. Unlike `ly:page-turn-breaking`, it has no concept of page turns.

5.4.4 Optimal page turning

Often it is necessary to find a page breaking configuration so that there is a rest at the end of every second page. This way, the musician can turn the page without having to miss notes. The `ly:page-turn-breaking` function attempts to find a page breaking minimizing cramping and stretching, but with the additional restriction that it is only allowed to introduce page turns in specified places.

There are two steps to using this page breaking function. First, you must enable it in the `\paper` block, as explained in Section 5.4.2 [Page breaking], page 217. Then you must tell the function where you would like to allow page breaks.

There are two ways to achieve the second step. First, you can specify each potential page turn manually, by inserting `\allowPageTurn` into your input file at the appropriate places.

If this is too tedious, you can add a `Page_turn_engraver` to a Staff or Voice context. The `Page_turn_engraver` will scan the context for sections without notes (note that it does not scan for rests; it scans for the absence of notes. This is so that single-staff polyphony with rests in one of the parts does not throw off the `Page_turn_engraver`). When it finds a sufficiently long section without notes, the `Page_turn_engraver` will insert an `\allowPageTurn` at the final bar line in that section, unless there is a 'special' bar line (such as a double bar), in which case the `\allowPageTurn` will be inserted at the final 'special' bar line in the section.

The `Page_turn_engraver` reads the context property `minimumPageTurnLength` to determine how long a note-free section must be before a page turn is considered. The default value for

`minimumPageTurnLength` is `#(ly:make-moment 1 1)`. If you want to disable page turns, you can set it to something very large.

```
\new Staff \with { \consists "Page_turn_engraver" }
{
  a4 b c d |
  R1 | % a page turn will be allowed here
  a4 b c d |
  \set Staff.minimumPageTurnLength = #(ly:make-moment 5 2)
  R1 | % a page turn will not be allowed here
  a4 b r2 |
  R1*2 | % a page turn will be allowed here
  a1
}
```

The `Page_turn_engraver` detects volta repeats. It will only allow a page turn during the repeat if there is enough time at the beginning and end of the repeat to turn the page back. The `Page_turn_engraver` can also disable page turns if the repeat is very short. If you set the context property `minimumRepeatLengthForPageTurn` then the `Page_turn_engraver` will only allow turns in repeats whose duration is longer than this value.

The page turning commands, `\pageTurn`, `\noPageTurn` and `\allowPageTurn`, may also be used at top-level, between scores and top-level markups.

Predefined commands

`\pageTurn` `\noPageTurn` `\allowPageTurn`

Known issues and warnings

There should only be one `Page_turn_engraver` in a score. If there is more than one, they will interfere with each other.

5.4.5 Minimal page breaking

The `ly:minimal-breaking` function performs minimal computations to calculate the page breaking: it fills a page with as many systems as possible before moving to the next one. Thus, it may be prefered for scores with many pages, where the other page breaking functions could be too slow or memory demanding, or a lot of texts. It is enabled using:

```
\paper {
  #(define page-breaking ly:minimal-breaking)
}
```

5.4.6 Explicit breaks

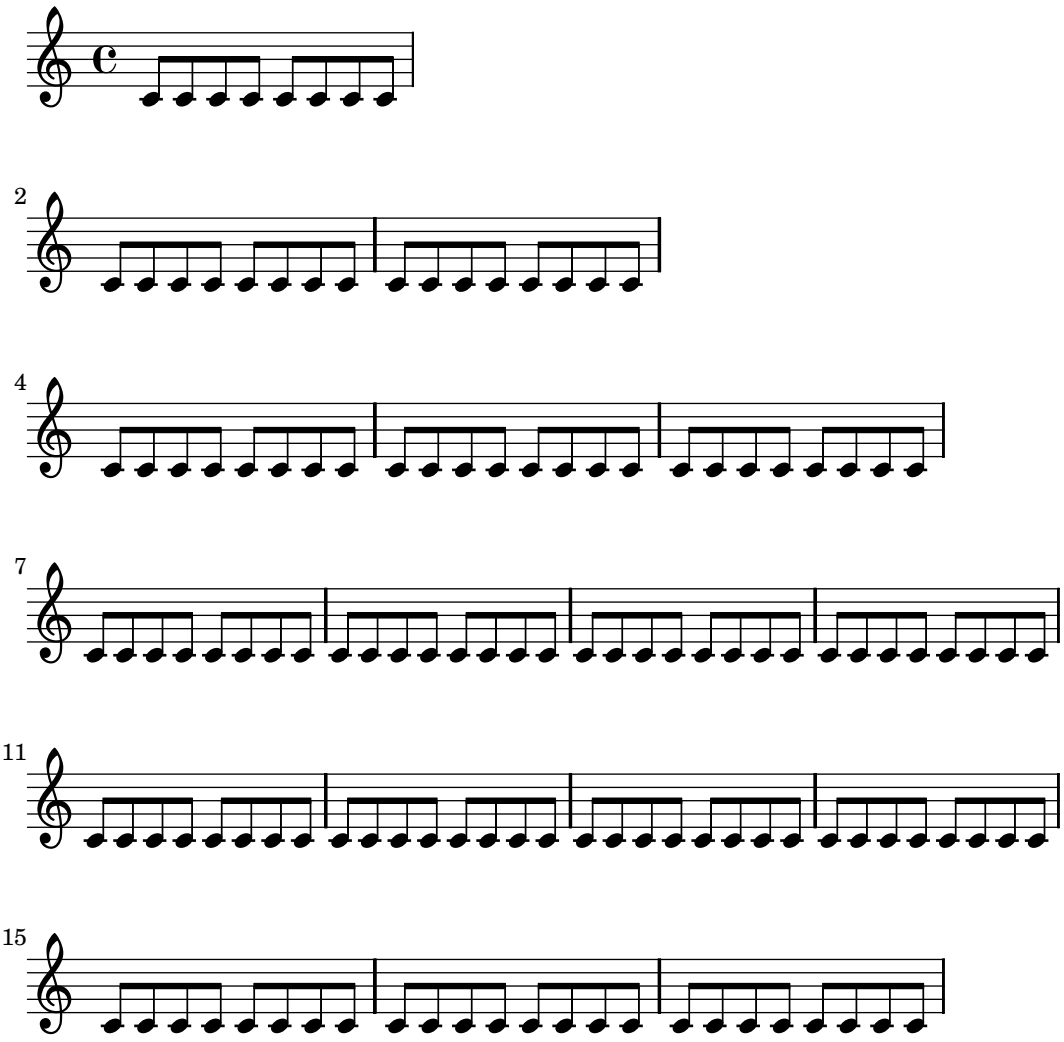
Lily sometimes rejects explicit `\break` and `\pageBreak` commands. There are two commands to override this behavior:

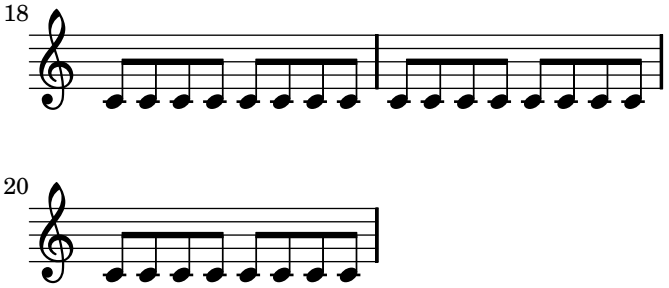
```
\override NonMusicalPaperColumn #'line-break-permission = ##f
\override NonMusicalPaperColumn #'page-break-permission = ##f
```

When `line-break-permission` is overriden to false, Lily will insert line breaks at explicit `\break` commands and nowhere else. When `page-break-permission` is overriden to false, Lily will insert page breaks at explicit `\pageBreak` commands and nowhere else.

```
\paper {
  indent = #0
```

```
  ragged-right = ##t
  ragged-bottom = ##t
}

\score {
  \new Score \with {
    \override NonMusicalPaperColumn #'line-break-permission = ##f
    \override NonMusicalPaperColumn #'page-break-permission = ##f
  } {
    \new Staff {
      \repeat unfold 2 { c'8 c'8 c'8 c'8 } \break
      \repeat unfold 4 { c'8 c'8 c'8 c'8 } \break
      \repeat unfold 6 { c'8 c'8 c'8 c'8 } \break
      \repeat unfold 8 { c'8 c'8 c'8 c'8 } \pageBreak
      \repeat unfold 8 { c'8 c'8 c'8 c'8 } \break
      \repeat unfold 6 { c'8 c'8 c'8 c'8 } \break
      \repeat unfold 4 { c'8 c'8 c'8 c'8 } \break
      \repeat unfold 2 { c'8 c'8 c'8 c'8 }
    }
  }
}
```





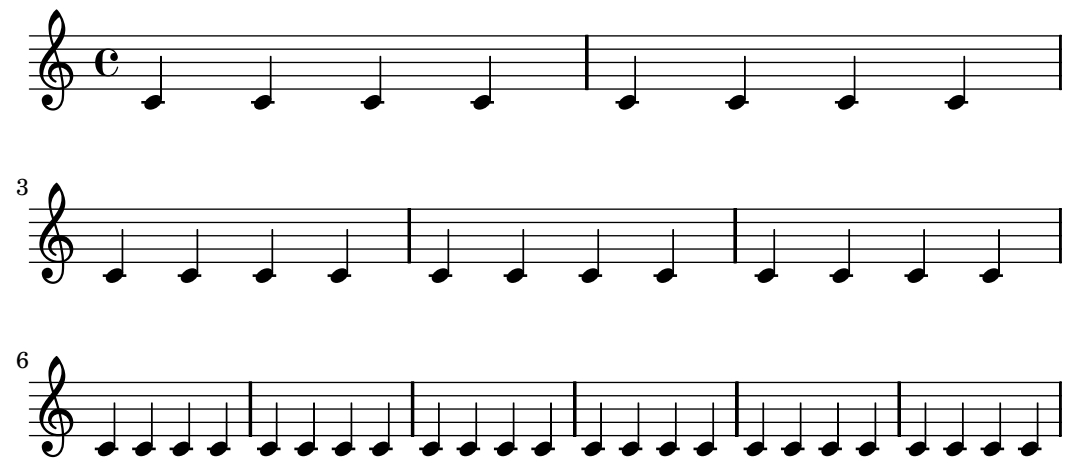
5.4.7 Using an extra voice for breaks

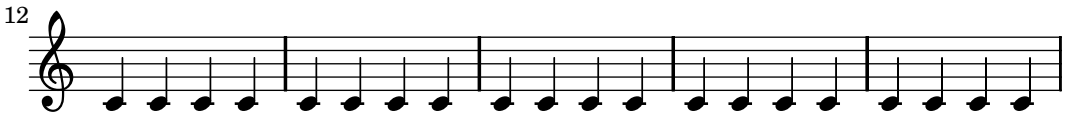
Line- and page-breaking information usually appears within note entry directly.

```
\new Score {
  \new Staff {
    \repeat unfold 2 { c'4 c'4 c'4 c'4 }
    \break
    \repeat unfold 3 { c'4 c'4 c'4 c'4 }
  }
}
```

This makes `\break` and `\pageBreak` commands easy to enter but mixes music entry with information that specifies how music should lay out on the page. You can keep music entry and line- and page-breaking information in two separate places by introducing an extra voice to contain the breaks. This extra voice contains only skips together with `\break`, `pageBreak` and other breaking layout information.

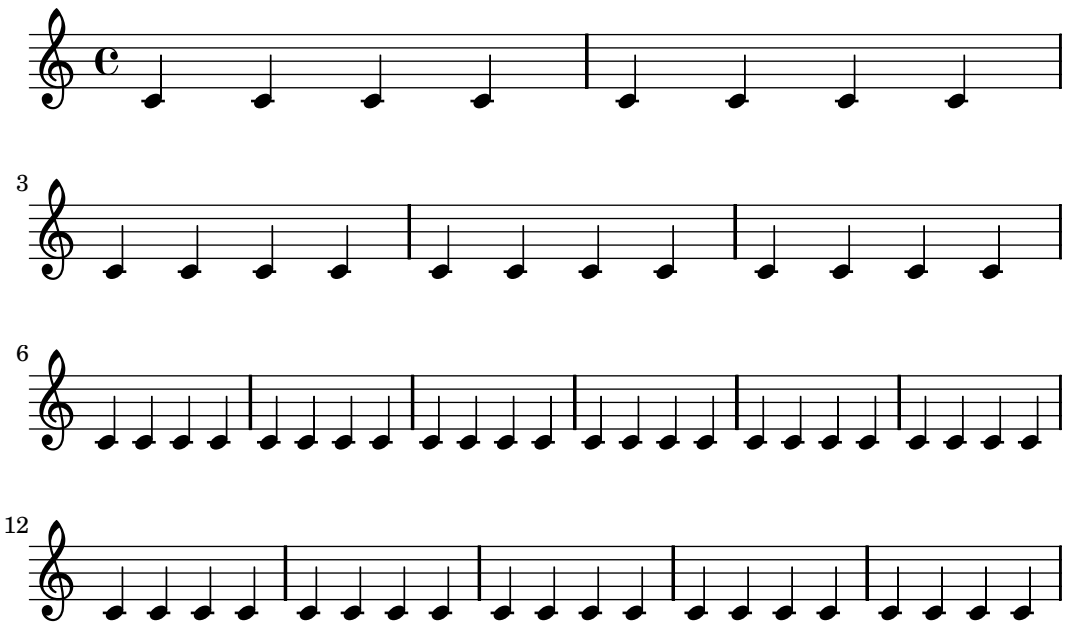
```
\new Score {
  \new Staff <<
     \new Voice {
        s1 * 2 \break
        s1 * 3 \break
        s1 * 6 \break
        s1 * 5 \break
     }
     \new Voice {
        \repeat unfold 2 { c'4 c'4 c'4 c'4 }
        \repeat unfold 3 { c'4 c'4 c'4 c'4 }
        \repeat unfold 6 { c'4 c'4 c'4 c'4 }
        \repeat unfold 5 { c'4 c'4 c'4 c'4 }
     }
  >>
}
```





This pattern becomes especially helpful when overriding `line-break-system-details` and the other useful but long properties of `NonMusicalPaperColumnGrob`, as explained in Section 5.5 [Vertical spacing], page 222.

```
\new Score {
  \new Staff <<
     \new Voice {

        \overrideProperty "Score.NonMusicalPaperColumn"
        #'line-break-system-details #'((Y-offset . 0))
        s1 * 2 \break

        \overrideProperty "Score.NonMusicalPaperColumn"
        #'line-break-system-details #'((Y-offset . 35))
        s1 * 3 \break

        \overrideProperty "Score.NonMusicalPaperColumn"
        #'line-break-system-details #'((Y-offset . 70))
        s1 * 6 \break

        \overrideProperty "Score.NonMusicalPaperColumn"
        #'line-break-system-details #'((Y-offset . 105))
        s1 * 5 \break
     }
     \new Voice {
        \repeat unfold 2 { c'4 c'4 c'4 c'4 }
        \repeat unfold 3 { c'4 c'4 c'4 c'4 }
        \repeat unfold 6 { c'4 c'4 c'4 c'4 }
        \repeat unfold 5 { c'4 c'4 c'4 c'4 }
     }
  >>
}
```



5.5 Vertical spacing

Vertical spacing is controlled by three things: the amount of space available (i.e., paper size and margins), the amount of space between systems, and the amount of space between staves inside a system.

5.5.1 Vertical spacing inside a system

The height of each system is determined automatically. To prevent staves from bumping into each other, some minimum distances are set. By changing these, you can put staves closer together. This reduces the amount of space each system requires, and may result in having more systems per page.

Normally staves are stacked vertically. To make staves maintain a distance, their vertical size is padded. This is done with the property `minimum-Y-extent`. When applied to a `VerticalAxisGroup`, it controls the size of a horizontal line, such as a staff or a line of lyrics. `minimum-Y-extent` takes a pair of numbers, so if you want to make it smaller than its default `#'(-4 . 4)` then you could set

```
\override Staff.VerticalAxisGroup #'minimum-Y-extent = #'(-3 . 3)
```

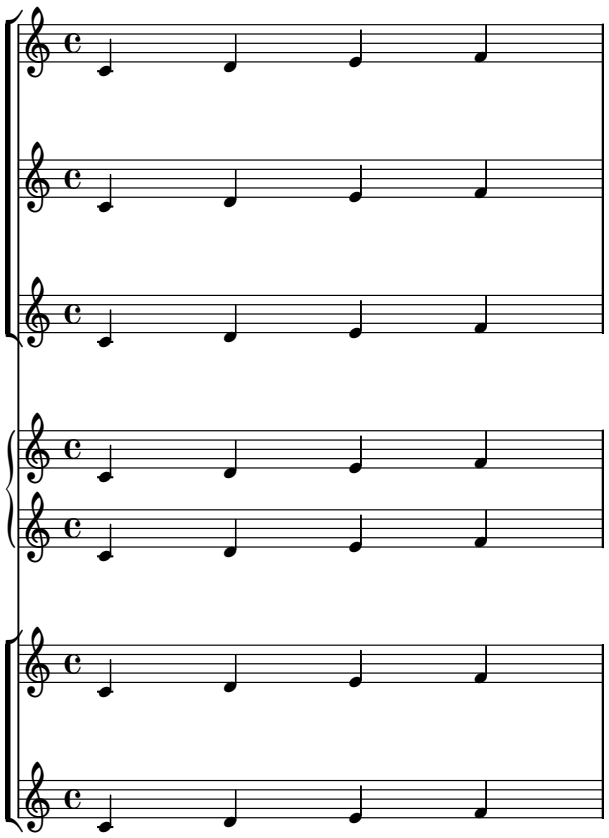
This sets the vertical size of the current staff to 3 staff spaces on either side of the center staff line. The value `(-3 . 3)` is interpreted as an interval, where the center line is the 0, so the first number is generally negative. The numbers need not match; for example, the staff can be made larger at the bottom by setting it to `(-6 . 4)`.

After page breaks are determined, the vertical spacing within each system is reevaluated in order to fill the page more evenly; if a page has space left over, systems are stretched in order to fill that space. The amount of stretching can be configured though the `max-stretch` property of the `VerticalAlignment` grob. By default, `max-stretch` is set to zero, disabling stretching. To enable stretching, a sane value for `max-stretch` is `ly:align-interface::calc-max-stretch`.

In some situations, you may want to stretch most of a system while leaving some parts fixed. For example, if a piano part occurs in the middle of an orchestral score, you may want to leave the piano staves close to each other while stretching the rest of the score. The `keep-fixed-while-stretching` property of `VerticalAxisGroup` can be used to achieve this. When set to `##t`, this property keeps its staff (or line of lyrics) from moving relative to the one directly above it. In the example above, you would override `keep-fixed-while-stretching` to `##t` in the second piano staff:

```
#(set-default-paper-size "a6")
#(set-global-staff-size 14.0)

\book {
\paper {
  ragged-last-bottom = ##f
}

\new Score \with
{
  \override VerticalAlignment #'max-stretch = #ly:align-interface::calc-max-stretch
}
{
\new GrandStaff
<<
  \new StaffGroup
  <<
    \new Staff {c' d' e' f'}
```

```
    \new Staff {c' d' e' f'}
    \new Staff {c' d' e' f'}
  >>

  \new PianoStaff
  <<
    \new Staff {c' d' e' f'}
    \new Staff \with {
      \override VerticalAxisGroup #'keep-fixed-while-stretching = ##t
    }
    {c' d' e' f'}
  >>

  \new StaffGroup
  <<
    \new Staff {c' d' e' f'}
    \new Staff {c' d' e' f'}
  >>
>>
}
}
```



See also

Internals: Vertical alignment of staves is handled by the `VerticalAlignment` object. The context parameters specifying the vertical extent are described in connection with the `Axis_group_engraver`.

Example files:

5.5.2 Vertical spacing between systems

Space between systems are controlled by four `\paper` variables,

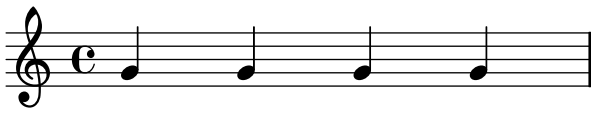
```
\paper {
  between-system-space = 1.5\cm
  between-system-padding = #1
  ragged-bottom=##f
  ragged-last-bottom=##f
}
```

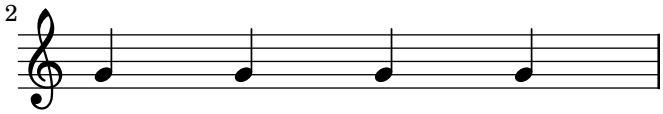
When only a couple of flat systems are placed on a page, the resulting vertical spacing may be non-eleguant: one system at the top of the page, and the other at the bottom, with a huge gap between them. To avoid this situation, the space added between the systems can be limited. This feature is activated by setting to `#t` the `page-limit-inter-system-space` variable in the `\paper` block. The paper variable `page-limit-inter-system-space-factor` determines how much the space can be increased: for instance, the value `1.3` means that the space can be 30% larger than what it would be on a ragged-bottom page.

In the following example, if the inter system space were not limited, the second system of page 1 would be placed at the page bottom. By activating the space limitation, the second system is placed closer to the first one. By setting `page-limit-inter-system-space-factor` to `1`, the spacing would the same as on a ragged-bottom page, like the last one.

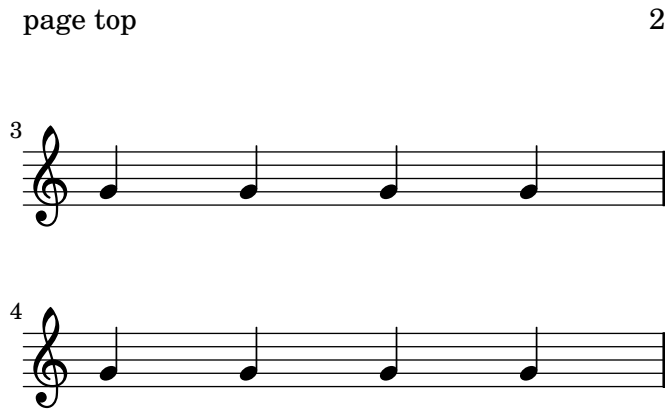
```
#(set-default-paper-size "a6")
\book {
  \paper {
    page-limit-inter-system-space = ##t
    page-limit-inter-system-space-factor = 1.3

    oddFooterMarkup = \markup "page bottom"
    evenFooterMarkup = \markup "page bottom"
    oddHeaderMarkup = \markup \fill-line {
      "page top" \fromproperty #'page:page-number-string }
    evenHeaderMarkup = \markup \fill-line {
      "page top" \fromproperty #'page:page-number-string }
  }
  \new Staff << \repeat unfold 4 { g'4 g' g' g' \break }
                { s1*2 \pageBreak } >>
}
```

page top 1





page bottom



5.5.3 Explicit staff and system positioning

One way to understand the `VerticalAxisGroup` and `\paper` settings explained in the previous two sections is as a collection of different settings that primarily concern the amount of vertical padding different staves and systems running down the page.

It is possible to approach vertical spacing in a different way using `NonMusicalPaperColumn #'line-break-system-details`. Where `VerticalAxisGroup` and `\paper` settings specify vertical padding, `NonMusicalPaperColumn #'line-break-system-details` specifies exact vertical positions on the page.

`NonMusicalPaperColumn #'line-break-system-details` accepts an associative list of five different settings:

- `X-offset`
- `Y-offset`
- `alignment-offsets`
- `alignment-extra-space`
- `fixed-alignment-extra-space`

Grob overrides, including the overrides for `NonMusicalPaperColumn` below, can occur in any of three different places in an input file:

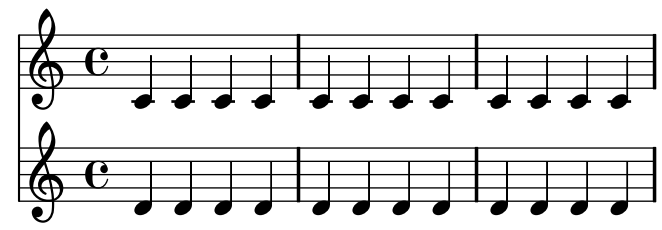
- in the middle of note entry directly
- in a `\context` block
- in the `\with` block

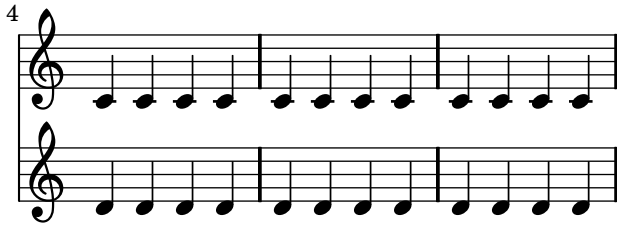
When we override `NonMusicalPaperColumn`, we use the usual `\override` command in `\context` blocks and in the `\with` block. On the other hand, when we override

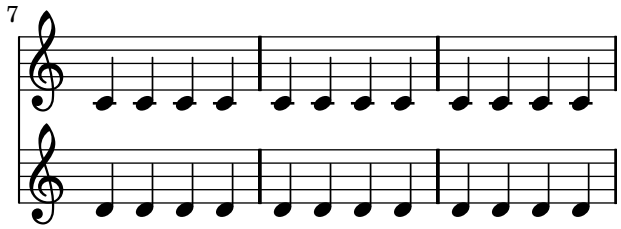
`NonMusicalPaperColumn` in the middle of note entry, use the special `\overrideProperty` command. Here are some example `NonMusicalPaperColumn` overrides with the special `\overrideProperty` command:

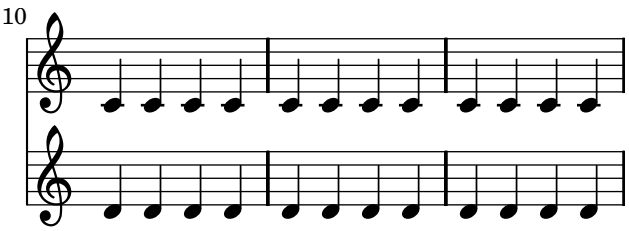
```
\overrideProperty NonMusicalPaperColumn
  #'line-break-system-details #'((X-offset . 20))

\overrideProperty NonMusicalPaperColumn
  #'line-break-system-details #'((Y-offset . 40))

\overrideProperty NonMusicalPaperColumn
  #'line-break-system-details #'((X-offset . 20) (Y-offset . 40))

\override NonMusicalPaperColumn
  #'line-break-system-details #'((alignment-offsets . (0 -15)))

\override NonMusicalPaperColumn
  #'line-break-system-details #'((X-offset . 20) (Y-offset . 40)
                                 (alignment-offsets . (0 -15)))
```

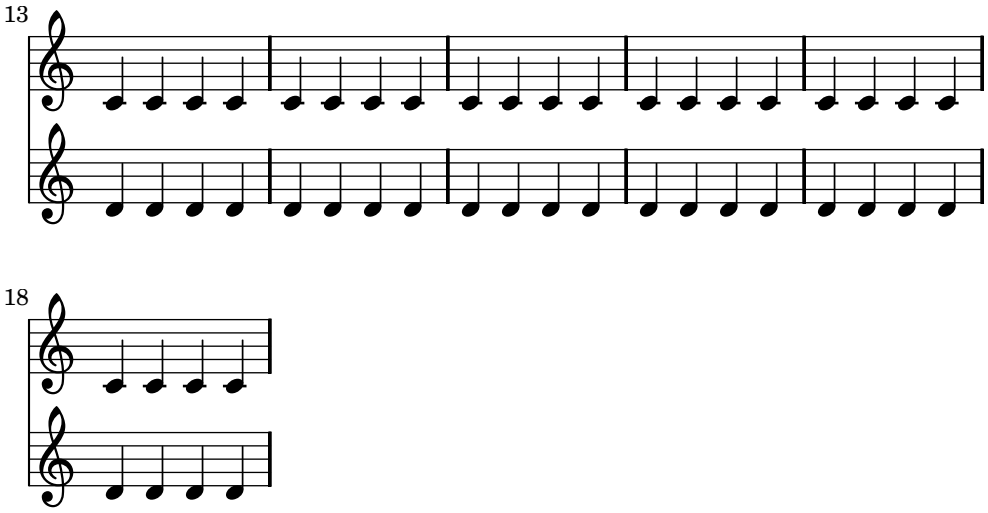
To understand how each of these different settings work, we begin by looking at an example that includes no overrides at all.





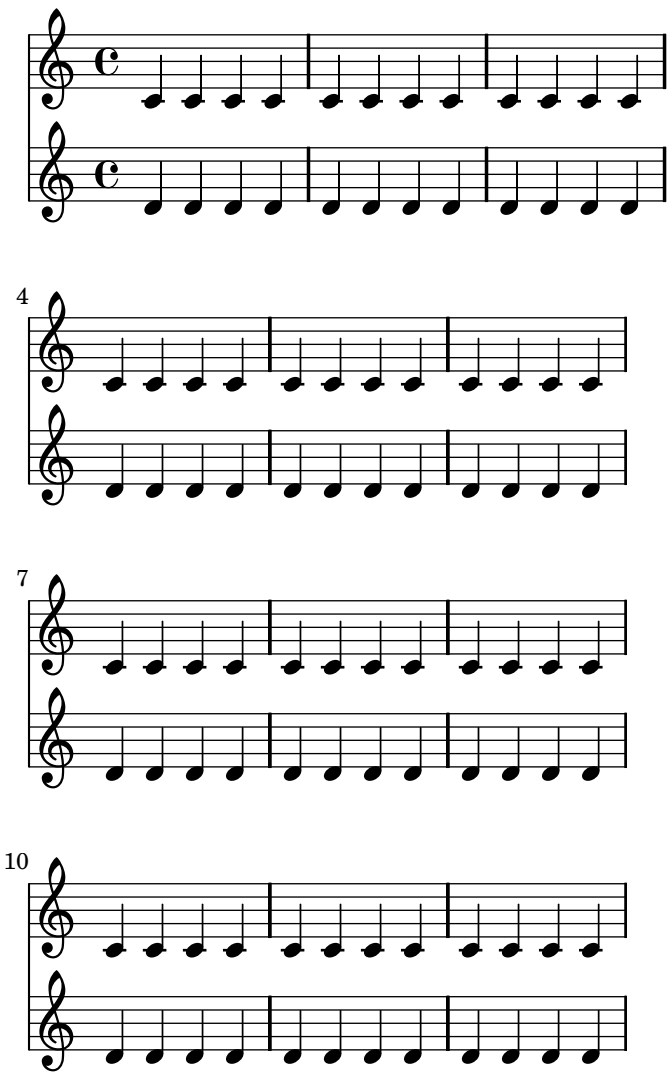


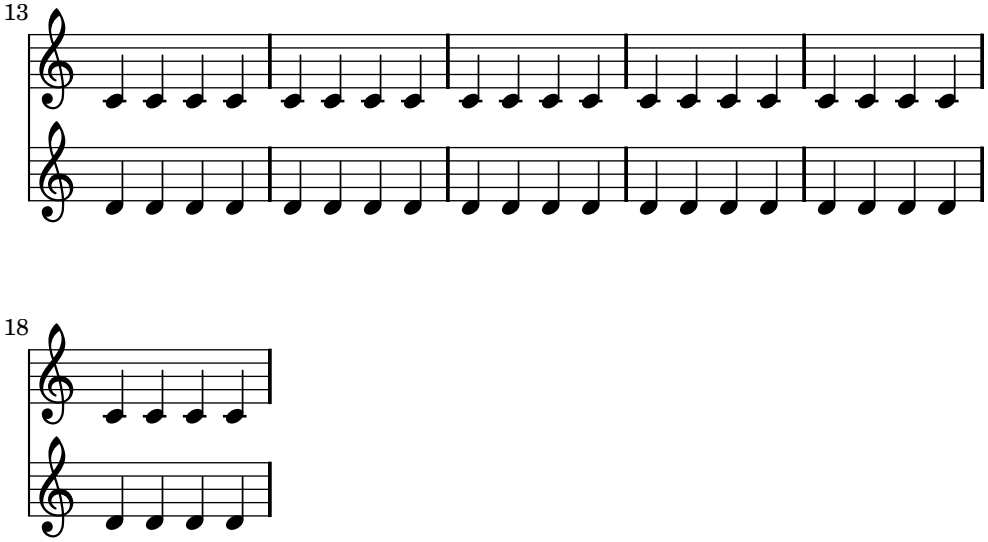




This score isolates line- and page-breaking information in a dedicated voice. This technique of creating a breaks voice will help keep layout separate from music entry as our example becomes more complicated. See Section 5.4.7 [Using an extra voice for breaks], page 220.

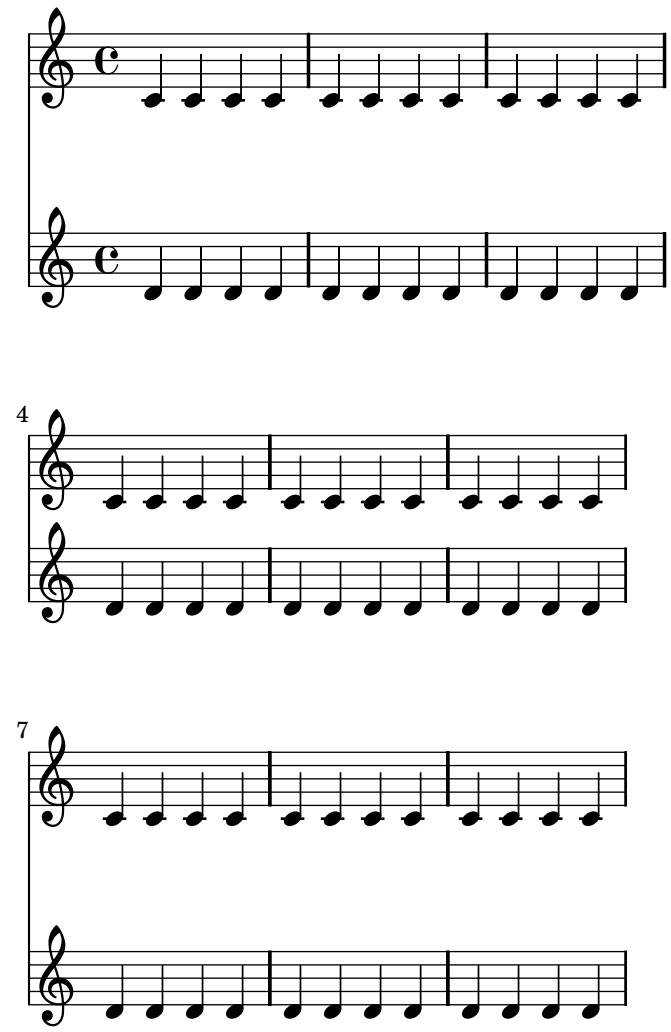
Explicit `\breaks` evenly divide the music into six measures per line. Vertical spacing results from LilyPond's defaults. To set the vertical startpoint of each system explicitly, we can set the `Y-offset` pair in the `line-break-system-details` attribute of the `NonMusicalPaperColumn` grob:

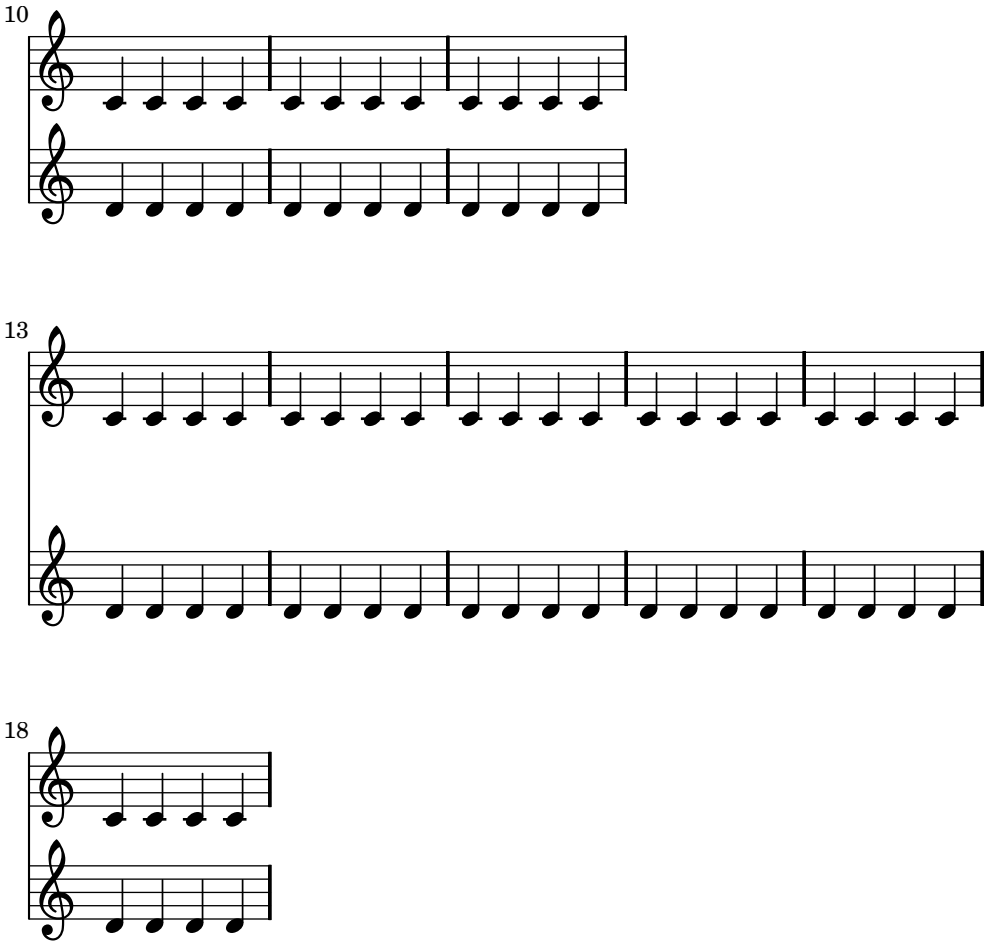




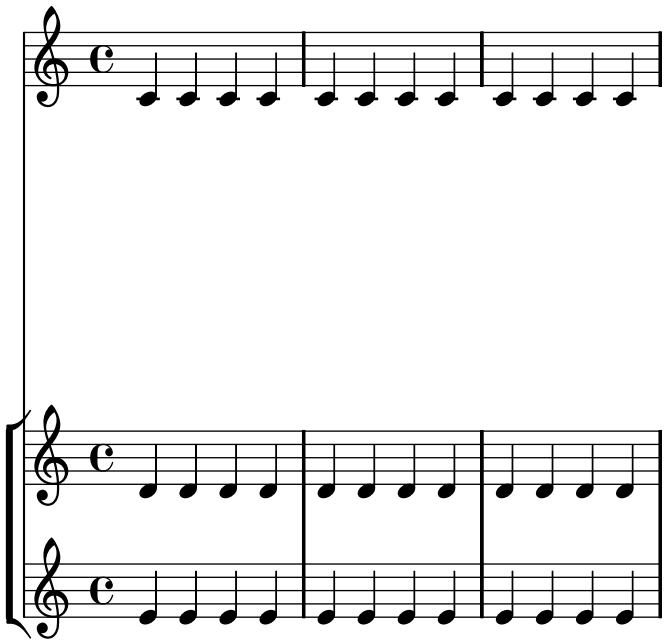
Note that `line-break-system-details` takes an associative list of potentially many values, but that we set only one value here. Note, too, that the `Y-offset` property here determines the exact vertical position on the page at which each new system will render.

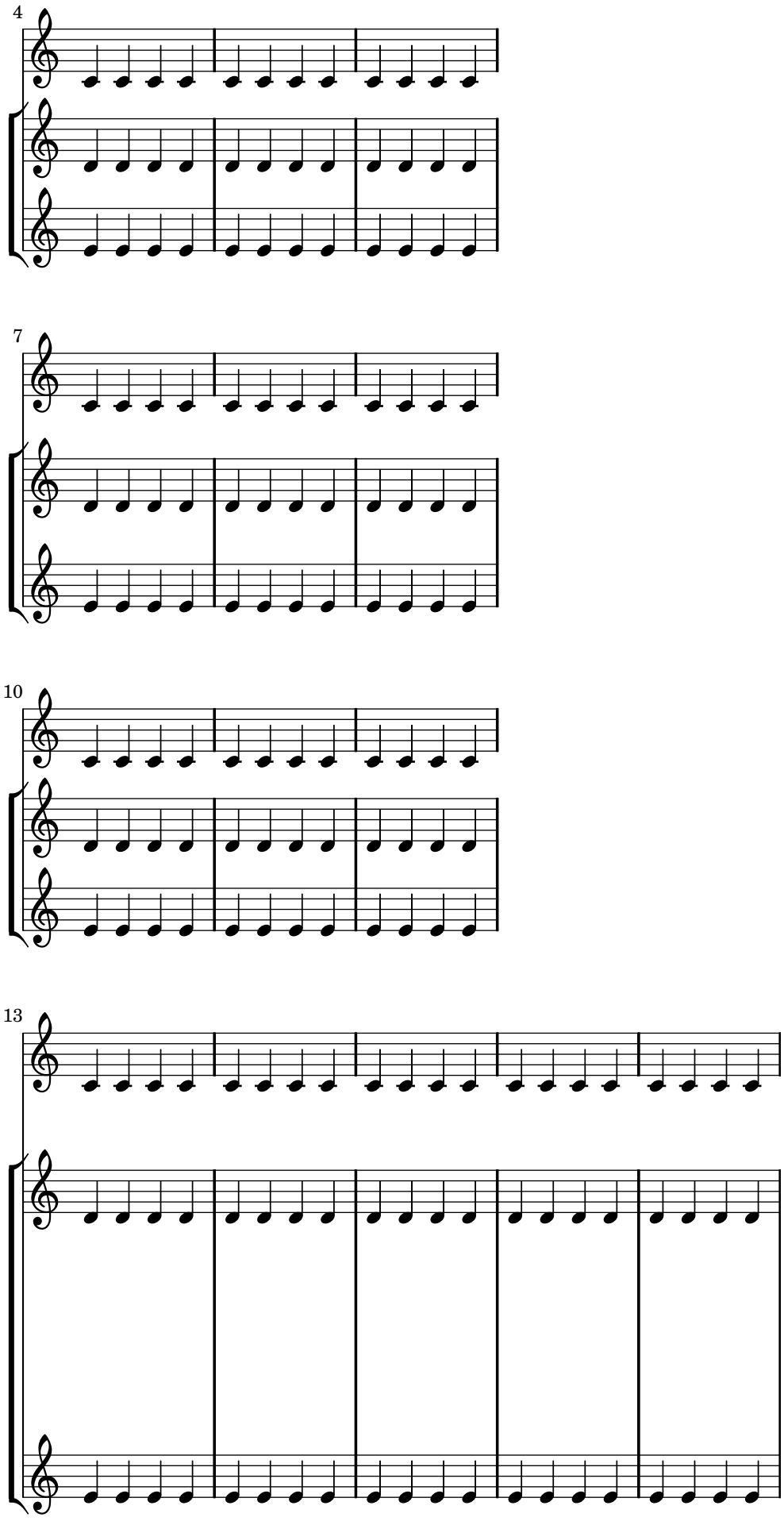
Now that we have set the vertical startpoint of each system explicitly, we can also set the vertical startpoint of each staff within each system manually. We do this using the `alignment-offsets` subproperty of `line-break-system-details`.

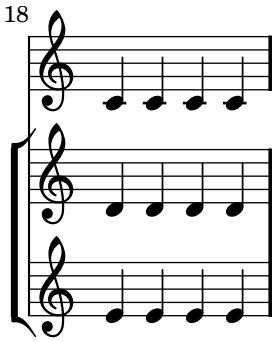




Note that here we assign two different values to the `line-break-system-details` attribute of the `NonMusicalPaperColumn` grob. Though the `line-break-system-details` attribute alist accepts many additional spacing parameters (including, for example, a corresponding `X-offset` pair), we need only set the `Y-offset` and `alignment-offsets` pairs to control the vertical startpoint of every system and every staff. Finally, note that `alignment-offsets` specifies the vertical positioning of staves but not of staff groups.




4
7
10
13



Some points to consider:

- When using `alignment-offsets`, lyrics count as a staff.
- The units of the numbers passed to `X-offset`, `Y-offset` and `alignment-offsets` are interpreted as multiples of the distance between adjacent staff lines. Positive values move staves and lyrics up, negative values move staves and lyrics down.
- Because the `NonMusicalPaperColumn #'line-break-system-details` settings given here allow the positioning of staves and systems anywhere on the page, it is possible to violate paper or margin boundaries or even to print staves or systems on top of one another. Reasonable values passed to these different settings will avoid this.

5.5.4 Two-pass vertical spacing

Warning: two-pass vertical spacing is deprecated and will be removed in a future version of LilyPond. Systems are now stretched automatically in a single pass. See Section 5.5.1 [Vertical spacing inside a system], page 222.

In order to automatically stretch systems so that they should fill the space left on a page, a two-pass technique can be used:

1. In the first pass, the amount of vertical space used to increase the height of each system is computed and dumped to a file.
2. In the second pass, spacing inside the systems are stretched according to the data in the page layout file.

The `ragged-bottom` property adds space between systems, while the two-pass technique adds space between staves inside a system.

To allow this behaviour, a `tweak-key` variable has to be set in each score `\layout` block, and the tweaks included in each score music, using the `\scoreTweak` music function.

```
%% include the generated page layout file:
\includePageLayoutFile

\score {
  \new StaffGroup <<
    \new Staff <<
      %% Include this score tweaks:
      \scoreTweak "scoreA"
      { \clef french c''1 \break c''1 }
    >>
    \new Staff { \clef soprano g'1 g'1 }
    \new Staff { \clef mezzosoprano e'1 e'1 }
    \new Staff { \clef alto g1 g1 }
    \new Staff { \clef bass c1 c1 }
  >>
  \header {
    piece = "Score with tweaks"
```

```
  }
  %% Define how to name the tweaks for this score:
  \layout { #(define tweak-key "scoreA") }
}
```

For the first pass, the `dump-tweaks` option should be set to generate the page layout file.

```
lilypond -dbackend=null -d dump-tweaks <file>.ly
lilypond <file>.ly
```

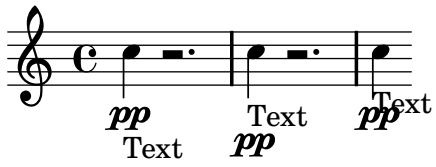
5.5.5 Vertical collision avoidance

Intuitively, there are some objects in musical notation that belong to the staff and there are other objects that should be placed outside the staff. Objects belonging outside the staff include things such as rehearsal marks, text and dynamic markings (from now on, these will be called outside-staff objects). LilyPond's rule for the vertical placement of outside-staff objects is to place them as close to the staff as possible but not so close that they collide with another object.

LilyPond uses the `outside-staff-priority` property to determine whether a grob is an outside-staff object: if `outside-staff-priority` is a number, the grob is an outside-staff object. In addition, `outside-staff-priority` tells LilyPond in which order the objects should be placed.

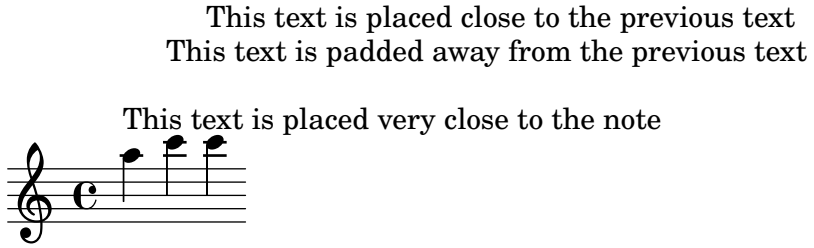
First, LilyPond places all the objects that do not belong outside the staff. Then it sorts the outside-staff objects according to their `outside-staff-priority` (in increasing order). One by one, LilyPond takes the outside-staff objects and places them so that they do not collide with any objects that have already been placed. That is, if two outside-staff grobs are competing for the same space, the one with the lower `outside-staff-priority` will be placed closer to the staff.

```
c4_"Text"\pp
r2.
\once \override TextScript #'outside-staff-priority = #1
c4_"Text"\pp % this time the text will be closer to the staff
r2.
% by setting outside-staff-priority to a non-number, we
% disable the automatic collision avoidance
\once \override TextScript #'outside-staff-priority = ##f
\once \override DynamicLineSpanner #'outside-staff-priority = ##f
c4_"Text"\pp % now they will collide
```



The vertical padding between an outside-staff object and the previously-positioned grobs can be controlled with `outside-staff-padding`.

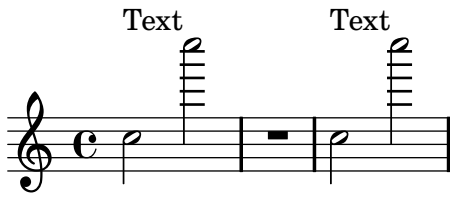
```
\once \override TextScript #'outside-staff-padding = #0
a'^"This text is placed very close to the note"
\once \override TextScript #'outside-staff-padding = #3
c^"This text is padded away from the previous text"
c^"This text is placed close to the previous text"
```



TODO: this example doesn't work any more ?

By default, outside-staff objects are placed without regard to their horizontal distance from the previously-posititioned grobs. This can lead to situations in which objects are placed very close to each other horizontally. Setting `outside-staff-horizontal-padding` causes an object to be offset vertically so that such a situation doesn't occur.

```
% the markup is too close to the following note
c2^"Text"
c''2
% setting outside-staff-horizontal-padding fixes this
R1
\once \override TextScript #'outside-staff-horizontal-padding = #1
c,,2^"Text"
c''2
```



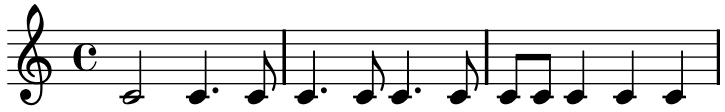
5.6 Horizontal Spacing

5.6.1 Horizontal spacing overview

The spacing engine translates differences in durations into stretchable distances ('springs') of differing lengths. Longer durations get more space, shorter durations get less. The shortest durations get a fixed amount of space (which is controlled by `shortest-duration-space` in the `SpacingSpanner` object). The longer the duration, the more space it gets: doubling a duration adds a fixed amount (this amount is controlled by `spacing-increment`) of space to the note.

For example, the following piece contains lots of half, quarter, and 8th notes; the eighth note is followed by 1 note head width (NHW). The quarter note is followed by 2 NHW, the half by 3 NHW, etc.

```
c2 c4. c8 c4. c8 c4. c8 c8
c8 c4 c4 c4
```



Normally, `spacing-increment` is set to 1.2 staff space, which is approximately the width of a note head, and `shortest-duration-space` is set to 2.0, meaning that the shortest note gets 2.4 staff space (2.0 times the `spacing-increment`) of horizontal space. This space is counted from the left edge of the symbol, so the shortest notes are generally followed by one NHW of space.

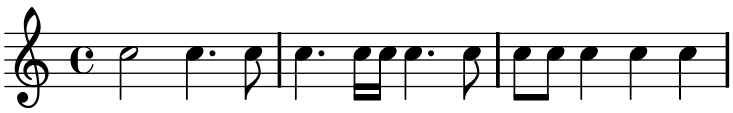
If one would follow the above procedure exactly, then adding a single 32nd note to a score that uses 8th and 16th notes, would widen up the entire score a lot. The shortest note is no longer a 16th, but a 32nd, thus adding 1 NHW to every note. To prevent this, the shortest duration for spacing is not the shortest note in the score, but rather the one which occurs most frequently.

The most common shortest duration is determined as follows: in every measure, the shortest duration is determined. The most common shortest duration is taken as the basis for the spacing, with the stipulation that this shortest duration should always be equal to or shorter than an 8th note. The shortest duration is printed when you run `lilypond` with the `--verbose` option.

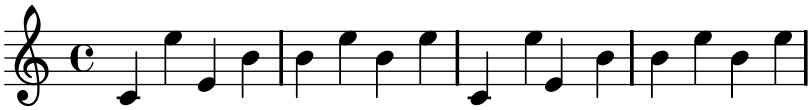
These durations may also be customized. If you set the `common-shortest-duration` in `SpacingSpanner`, then this sets the base duration for spacing. The maximum duration for this base (normally an 8th), is set through `base-shortest-duration`.

Notes that are even shorter than the common shortest note are followed by a space that is proportional to their duration relative to the common shortest note. So if we were to add only a few 16th notes to the example above, they would be followed by half a NHW:

```
c2 c4. c8 c4. c16[ c] c4. c8 c8 c8 c4 c4 c4
```



In the introduction (see learning manual, ⟨undefined⟩ [Engraving], page ⟨undefined⟩), it was explained that stem directions influence spacing. This is controlled with the `stem-spacing-correction` property in the `NoteSpacing`, object. These are generated for every `Voice` context. The `StaffSpacing` object (generated in `Staff` context) contains the same property for controlling the stem/bar line spacing. The following example shows these corrections, once with default settings, and once with exaggerated corrections:



Proportional notation is supported; see Section 5.6.5 [Proportional notation], page 239.

See also

Internals: `SpacingSpanner`, `NoteSpacing`, `StaffSpacing`, `SeparationItem`, and `SeparatingGroupSpanner`.

Known issues and warnings

There is no convenient mechanism to manually override spacing. The following work-around may be used to insert extra space into a score.

```
\once \override Score.SeparationItem #'padding = #1
```

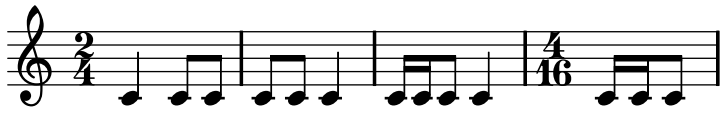
No work-around exists for decreasing the amount of space.

5.6.2 New spacing area

New sections with different spacing parameters can be started with `newSpacingSection`. This is useful when there are sections with a different notions of long and short notes.

In the following example, the time signature change introduces a new section, and hence the 16ths notes are spaced wider.

```
\time 2/4
c4 c8 c
c8 c c4 c16[ c c8] c4
\newSpacingSection
\time 4/16
c16[ c c8]
```

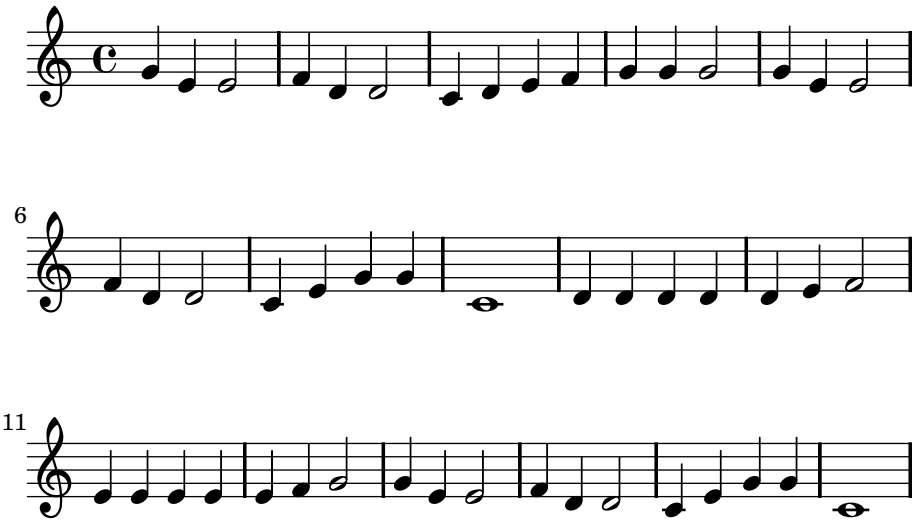


The `\newSpacingSection` command creates a new `SpacingSpanner` object, and hence new `\overrides` may be used in that location.

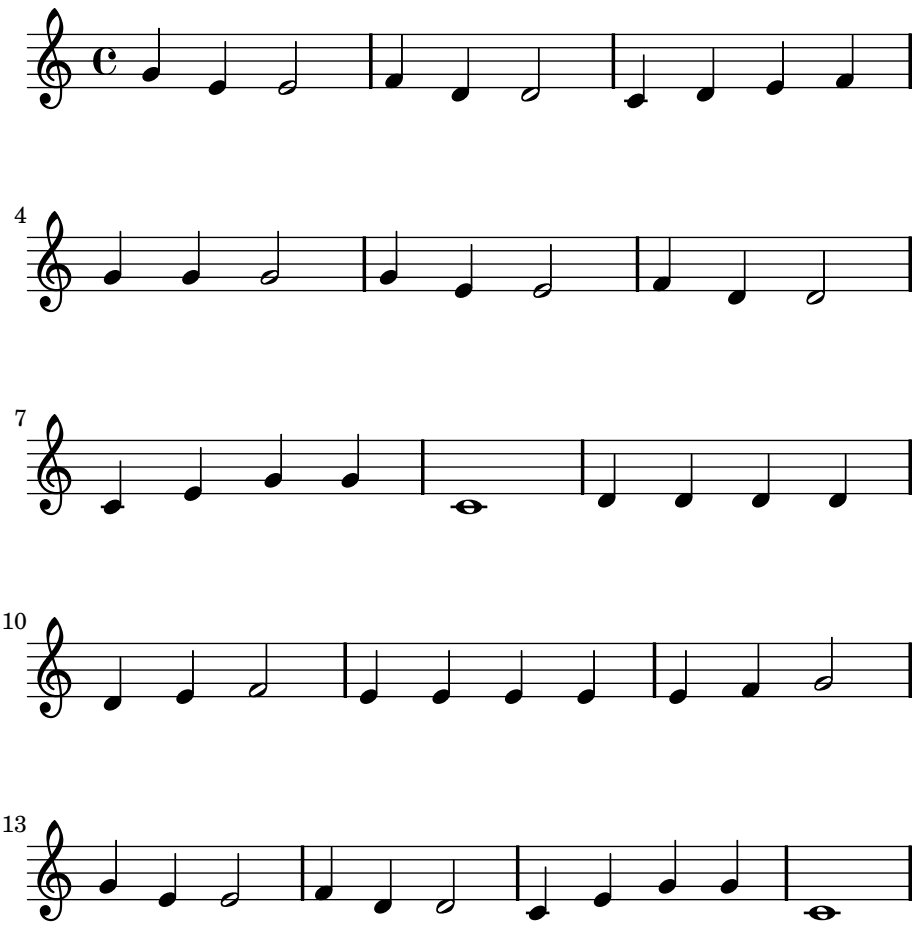
5.6.3 Changing horizontal spacing

Horizontal spacing may be altered with the `base-shortest-duration` property. Here we compare the same music; once without altering the property, and then altered. Larger values of `ly:make-moment` will produce smaller music. Note that `ly:make-moment` constructs a duration, so `1 4` is a longer duration than `1 16`.

```
\score {
  \relative c'' {
    g4 e e2 | f4 d d2 | c4 d e f | g4 g g2 |
    g4 e e2 | f4 d d2 | c4 e g g | c,1 |
    d4 d d d | d4 e f2 | e4 e e e | e4 f g2 |
    g4 e e2 | f4 d d2 | c4 e g g | c,1 |
  }
}
```



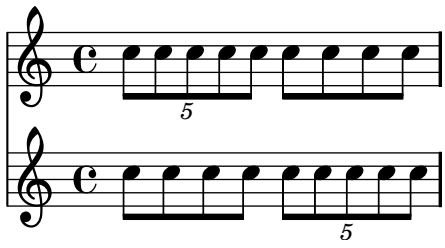
```
\score {
  \relative c'' {
    g4 e e2 | f4 d d2 | c4 d e f | g4 g g2 |
    g4 e e2 | f4 d d2 | c4 e g g | c,1 |
    d4 d d d | d4 e f2 | e4 e e e | e4 f g2 |
    g4 e e2 | f4 d d2 | c4 e g g | c,1 |
  }
  \layout {
    \context {
      \Score
      \override SpacingSpanner
                #'base-shortest-duration = #(ly:make-moment 1 16)
    }
  }
}
```



Selected Snippets

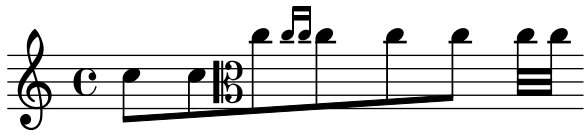
By default, spacing in tuplets depends on various non-duration factors (such as accidentals, clef changes, etc). To disregard such symbols and force uniform equal-duration spacing, use `Score.SpacingSpanner #'uniform-stretching`. This property can only be changed at the beginning of a score,

```
\new Score \with {
  \override SpacingSpanner #'uniform-stretching = ##t
} <<
  \new Staff{
    \times 4/5 {
      c8 c8 c8 c8 c8
    }
    c8 c8 c8 c8
  }
  \new Staff{
    c8 c8 c8 c8
    \times 4/5 {
      c8 c8 c8 c8 c8
    }
  }
>>
```



When `strict-note-spacing` is set, notes are spaced without regard for clefs, bar lines, and grace notes,

```
\override Score.SpacingSpanner #'strict-note-spacing = ##t
\new Staff { c8[ c \clef alto c \grace { c16[ c] } c8 c c]  c32[ c32] }
```



5.6.4 Line length

The most basic settings influencing the spacing are `indent` and `line-width`. They are set in the `\layout` block. They control the indentation of the first line of music, and the lengths of the lines.

If `ragged-right` is set to true in the `\layout` block, then systems ends at their natural horizontal length, instead of being spread horizontally to fill the whole line. This is useful for short fragments, and for checking how tight the natural spacing is.

The option `ragged-last` is similar to `ragged-right`, but only affects the last line of the piece. No restrictions are put on that line. The result is similar to formatting text paragraphs. In a paragraph, the last line simply takes its natural horizontal length.

```
\layout {
  indent = #0
  line-width = #150
  ragged-last = ##t
}
```

5.6.5 Proportional notation

LilyPond supports proportional notation, a type of horizontal spacing in which each note consumes an amount of horizontal space exactly equivalent to its rhythmic duration. This type of proportional spacing is comparable to horizontal spacing on top of graph paper. Some late 20th- and early 21st-century scores use proportional notation to clarify complex rhythmic relationships or to faciliate the placement of timelines or other graphics directly in the score.

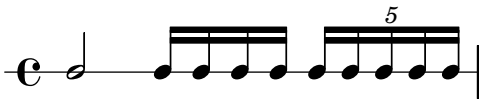
LilyPond supports five different settings for proportional notation, which may be used together or alone:

- `proportionalNotationDuration`
- `uniform-stretching`
- `strict-note-spacing`
- `\remove Separating_line_group_engraver`
- `\override PaperColumn #'used = ##t`

In the examples that follow, we explore these five different proportional notation settings and examine how these settings interact.

We start with the following one-measure example, which uses classical spacing with ragged-right turned on.

```
\new Score <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
>>
```



Notice that the half note which begins the measure takes up far less than half of the horizontal space of the measure. Likewise, the sixteenth notes and sixteenth-note quintuplets (or twentieth notes) which end the measure together take up far more than half the horizontal space of the measure.

In classical engraving, this spacing may be exactly what we want because we can borrow horizontal space from the half note and conserve horizontal space across the measure as a whole.

On the other hand, if we want to insert a measured timeline or other graphic above or below our score, we need proportional notation. We turn proportional notation on with the proportionalNotationDuration setting.

```
\new Score \with {
  proportionalNotationDuration = #(ly:make-moment 1 20)
} <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
```

```
    }
  }
>>
```

The half note at the beginning of the measure and the faster notes in the second half of the measure now occupy equal amounts of horizontal space. We could place a measured timeline or graphic above or below this example.

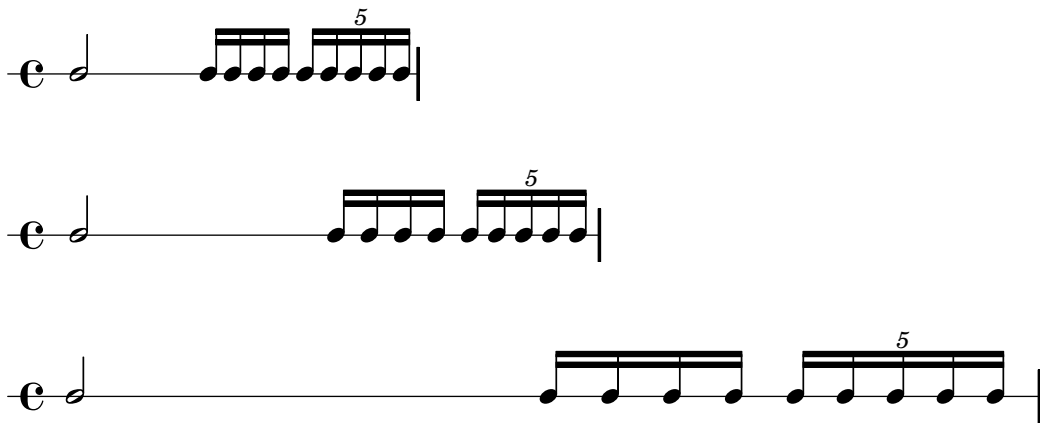
The `proportionalNotationDuration` setting is a context setting that lives in `Score`. Recall that context settings appear in one of three locations in our input file – in a `\with` block, in a `\context` block, or directly in music entry preceeded by the `\set` command. As with all context settings, users can pick which of the three different locations they would like to set `proportionalNotationDuration`.

The `proportionalNotationDuration` setting takes a single argument, which is the reference duration against which all music will be spaced. The LilyPond Scheme function make-moment takes two arguments – a numerator and denominator which together express some fraction of a whole note. The call `#(ly:make-moment 1 20)` therefore produces a reference duration of a twentieth note. The values `#(ly:make-moment 1 16)`, `#(ly:make-moment 1 8)`, and `#(ly:make-moment 3 97)` are all possible as well.

How do we select the right reference duration to pass to `proportionalNotationDuration`? Usually by a process of trial and error, beginning with a duration close to the fastest (or smallest) duration in the piece. Smaller reference durations space music loosely; larger reference durations space music tightly.

```
\new Score \with {
  proportionalNotationDuration = #(ly:make-moment 1 8)
} <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
>>

\new Score \with {
  proportionalNotationDuration = #(ly:make-moment 1 16)
} <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
>>

\new Score \with {
```

```
    proportionalNotationDuration = #(ly:make-moment 1 32)
  } <<
    \new RhythmicStaff {
      c'2
      c'16 c'16 c'16 c'16
      \times 4/5 {
        c'16 c'16 c'16 c'16 c'16
      }
    }
  >>
```

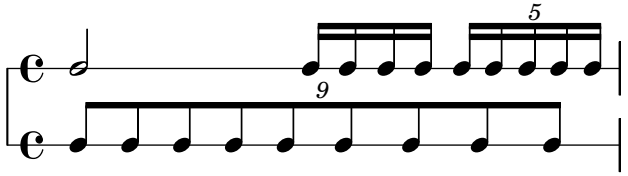


Note that too large a reference duration – such as the eighth note, above – spaces music too tightly and can cause note head collisions. Note also that proportional notation in general takes up more horizontal space that does classical spacing. Proportional spacing provides rhythmic clarity at the expense of horizontal space.

Next we examine how to optimally space overlapping tuplets.

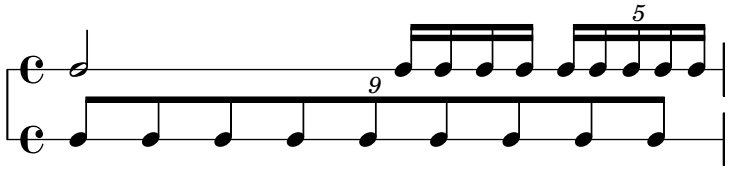
We start by examining what happens to our original example, with classical spacing, when we add a second staff with a different type of tuplet.

```
\new Score <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
  \new RhythmicStaff {
    \times 8/9 {
      c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8
    }
  }
>>
```



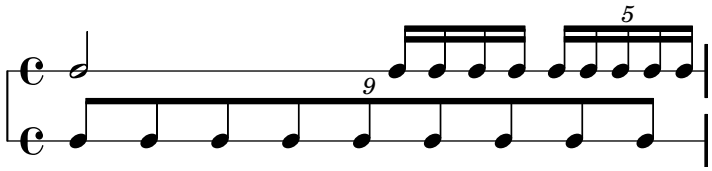
The spacing is bad because the evenly notes of the bottom staff do not stretch uniformly. Classical engraving includes very few complex triplets and so classical engraving rules can generate this type of result. Setting `proportionalNotationDuration` remedies this situation considerably.

```
\new Score \with {
  proportionalNotationDuration = #(ly:make-moment 1 20)
} <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
  \new RhythmicStaff {
    \times 8/9 {
      c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8
    }
  }
>>
```



But if we look very carefully we can see that notes of the second half of the 9-tuplet space ever so slightly more widely than do the notes of the first half of the 9-tuplet. To ensure uniform stretching, we turn on `uniform-stretching`, which is a property of `SpacingSpanner`.

```
\new Score \with {
  proportionalNotationDuration = #(ly:make-moment 1 20)
  \override SpacingSpanner #'uniform-stretching = ##t
} <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
  \new RhythmicStaff {
    \times 8/9 {
      c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8
    }
  }
>>
```



Our two-staff example now spaces exactly, our rhythmic relationships are visually clear, and we can include a measured timeline or graphic if we want.

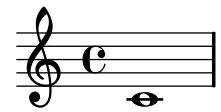
Note that the LilyPond's proportional notation package expects that all proportional scores set the SpacingSpanner's 'uniform-stretching attribute to ##t. Setting proportionalNotationDuration without also setting the SpacingSpanner's 'uniform-stretching attribute to ##t will, for example, cause Skips to consume an incorrect amount of horizontal space.

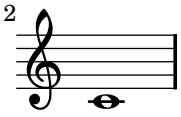
The SpacingSpanner is an abstract grob that lives in the Score context. As with our settings of proportionalNotationDuration, overrides to the SpacingSpanner can occur in any of three different places in our input file – in the Score \with block, in a Score \context block, or in note entry directly.

There is by default only one `SpacingSpanner` per `Score`. This means that, by default, `uniform-stretching` is either turned on for the entire score or turned off for the entire score. We can, however, override this behavior and turn on different spacing features at different places in the score. We do this with the command `\newSpacingSection`. See Section 5.6.2 [New spacing area], page 236, for more info.

Next we examine the effects of the `Separating_line_group_engraver` and see why proportional scores frequently remove this engraver. The following example shows that there is a small amount of "preferatory" space just before the first note in each system.

```
\paper {
  indent = #0
}

\new Staff {
  c'1
  \break
  c'1
}
```

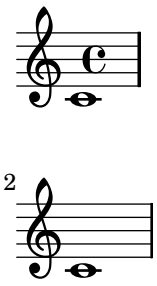




The amount of this preferatory space is the same whether after a time signature, a key signature or a clef. `Separating_line_group_engraver` is responsible for this space. Removing `Separating_line_group_engraver` reduces this space to zero.

```
\paper {
  indent = #0
}

\new Staff \with {
  \remove Separating_line_group_engraver
```

```
} {
  c'1
  \break
  c'1
}
```

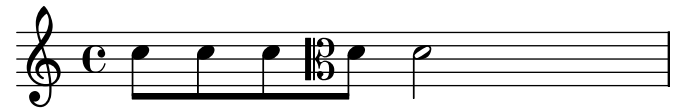


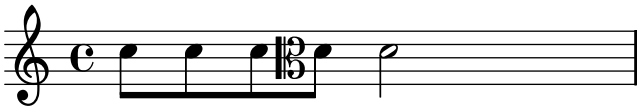
Nonmusical elements like time signatures, key signatures, clefs and accidentals are problemmatic in proportional notation. None of these elements has rhythmic duration. But all of these elements consume horizontal space. Different proportional scores approach these problems differently.

It may be possible to avoid spacing problems with key signatures simply by not having any. This is a valid option since most proportional scores are contemporary music. The same may be true of time signatures, especially for those scores that include a measured timeline or other graphic. But these scores are exceptional and most proportional scores include at least some time signatures. Clefs and accidentals are even more essential.

So what strategies exist for spacing nonmusical elements in a proportional context? One good option is the `strict-note-spacing` property of `SpacingSpanner`. Compare the two scores below:

```
\new Staff {
  \set Score.proportionalNotationDuration = #(ly:make-moment 1 16)
  c''8
  c''8
  c''8
  \clef alto
  d'8
  d'2
}

\new Staff {
  \set Score.proportionalNotationDuration = #(ly:make-moment 1 16)
  \override Score.SpacingSpanner #'strict-note-spacing = ##t
  c''8
  c''8
  c''8
  \clef alto
  d'8
  d'2
}
```





Both scores are proportional, but the spacing in the first score is too loose because of the clef change. The spacing of the second score remains strict, however, because strict-note-spacing is turned on. Turning on strict-note-spacing causes the width of time signatures, key signatures, clefs and accidentals to play no part in the spacing algorithm.

In addition to the settings given here, there are other settings that frequently appear in proportional scores. These include:

- `\override SpacingSpanner #'strict-grace-spacing = ##t`
- `tupletFullLength = ##t`
- `\override Beam #'breakable = ##t`
- `\override Glissando #'breakable = ##t`
- `\override TextSpanner #'breakable = ##t`
- `\remove Forbid_line_break_engraver in the Voice context`

These settings space grace notes strictly, extend tuplet brackets to mark both rhythmic start- and stop-points, and allow spanning elements to break across systems and pages. See the respective parts of the manual for these related settings.

5.7 Page layout MOVED FROM LM

5.7.1 Introduction to layout

The global paper layout is determined by three factors: the page layout, the line breaks, and the spacing. These all influence each other. The choice of spacing determines how densely each system of music is set. This influences where line breaks are chosen, and thus ultimately, how many pages a piece of music takes.

Settings which influence layout may be placed in two blocks. The `\paper {...}` block is placed outside any `\score {...}` blocks and contains settings that relate to the entire document. The `\layout {...}` block is placed within a `\score {...}` block and contains settings for that particular score. If you have only one `\score {...}` block the two have the same effect. In general the commands shown in this section can be placed in either.

Much more detail on the options for tweaking the laying out of music are contained in notation reference, Chapter 5 [Spacing issues], page 208 .

5.7.2 Global sizes

TODO Check all these examples

The default **paper size** which LilyPond assumes in laying out the music is A4. This may be changed in two ways:

```
#(set-default-paper-size "a6")

\paper {
#(set-paper-size "letter")
}
```

The first command sets the size of all pages. The second command sets the size of the pages to which the \paper block applies – if the \paper block is at the top of the file, then it will apply to all pages. Support for the following paper sizes is available: a6, a5, a4, a3, legal, letter, 11x17 (also known as tabloid). Setting the paper size automatically sets suitable margins and line length.

If the symbol `landscape` is supplied as an argument to `set-default-paper-size`, the pages will be rotated by 90 degrees, and wider line widths will be set correspondingly, e.g.

```
#(set-default-paper-size "a6" 'landscape)
```

The default **staff size** is set to 20 points. This may be changed in two ways:

```
#(set-global-staff-size 14)

\paper {
#(set-global-staff-size 16)
}
```

The first command sets the size in all pages. The second command sets the size in the pages to which the \paper block applies - if the \paper block is at the top of the file, then it will apply to all pages. All the fonts are automatically scaled to suit the new value of the staff size.

5.7.3 Line breaks

Line breaks are normally determined automatically. They are chosen so that lines look neither cramped nor loose, and consecutive lines have similar density. Occasionally you might want to override the automatic breaks; you can do this by specifying `\break`. This will force a line break at this point. However, line breaks can only occur at the end of 'complete' bars, i.e., where there are no notes or tuplets left 'hanging' over the bar line. If you want to have a line break where there is no bar line, you can force an invisible bar line by entering `\bar ""`, although again there must be no notes left hanging over in any of the staves at this point, or it will be ignored.

The opposite command, `\noBreak`, forbids a line break at the bar line where it is inserted.

The most basic settings influencing line spacing are `indent` and `line-width`. They are set in the `\layout` block. They control the indentation of the first line of music, and the lengths of the lines.

If `ragged-right` is set to true in the `\layout` block, then systems end at their natural horizontal length, instead of being spread horizontally to fill the whole line. This is useful for short fragments, and for checking how tight the natural spacing is.

The option `ragged-last` is similar to `ragged-right`, but affects only the last line of the piece.

```
\layout {
indent = #0
line-width = #150
ragged-last = ##t
}
```

5.7.4 Page breaks

The default page breaking may be overriden by inserting `\pageBreak` or `\noPageBreak` commands. These commands are analogous to the `\break` and `\noBreak` commands discused above and force or forbid a page-break at the point where they are inserted. Of course, the `\pageBreak` command also forces a line break. Like `\break`, the `\pageBreak` command is effective only at the end of a 'complete' bar as defined above. For more details see notation reference, Section 5.4.2 [Page breaking], page 217 and following sections.

There are also analogous settings to `ragged-right` and `ragged-last` which have the same effect on vertical spacing: `ragged-bottom` and `ragged-last-bottom`. If set to `##t` the systems on all pages or just the last page respectively will not be justified vertically.

For more details see notation reference, Section 5.5 [Vertical spacing], page 222 .

5.7.5 Fitting music onto fewer pages

Sometimes you can end up with one or two staves on a second (or third, or fourth...) page. This is annoying, especially if you look at previous pages and it looks like there is plenty of room left on those.

When investigating layout issues, `annotate-spacing` is an invaluable tool. This command prints the values of various layout spacing commands; see notation reference, Section 5.3 [Displaying spacing], page 214 , for more details. From the output of `annotate-spacing`, we can see which margins we may wish to alter.

Other than margins, there are a few other options to save space:

- You may tell LilyPond to place systems as close together as possible (to fit as many systems as possible onto a page), but then to space those systems out so that there is no blank space at the bottom of the page.

```
\paper {
  between-system-padding = #0.1
  between-system-space = #0.1
  ragged-last-bottom = ##f
  ragged-bottom = ##f
}
```

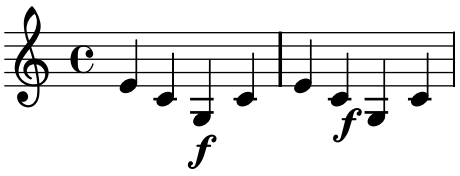
- You may force the number of systems (i.e., if LilyPond wants to typeset some music with 11 systems, you could force it to use 10).

```
\paper {
  system-count = #10
}
```

- Avoid (or reduce) objects which increase the vertical size of a system. For example, volta repeats (or alternate repeats) require extra space. If these repeats are spread over two systems, they will take up more space than one system with the volta repeats and another system without.

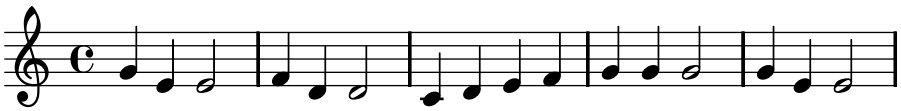
  Another example is moving dynamics which 'stick out' of a system, as in the second bar here:

```
e4 c g\f c
\override DynamicText #'extra-offset = #'( -2.2 . 2.0)
e4 c g\f c
```



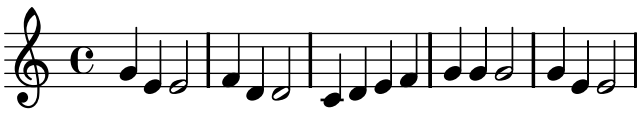
- Alter the horizontal spacing via `SpacingSpanner`. See notation reference, Section 5.6.3 [Changing horizontal spacing], page 236 , for more details. Here's an example first showing the default behaviour:

```
\score {
  \relative c'' {
    g4 e e2 |
    f4 d d2 |
    c4 d e f |
    g4 g g2 |
    g4 e e2 |
  }
}
```



and now with `common-shortest-duration` increased from the value of `1/4` (a quarter note is the most common in this example) to `1/2`:

```
\score {
  \relative c'' {
    g4 e e2 |
    f4 d d2 |
    c4 d e f |
    g4 g g2 |
    g4 e e2 |
  }
  \layout {
    \context {
      \Score
      \override SpacingSpanner
                #'common-shortest-duration = #(ly:make-moment 1 2)
    }
  }
}
```



Note that this override cannot be modified dynamically, so it must always be placed in a `\context{..}` block so that it applies to the whole score.

TODO Add description of using \context in this way earlier if it is not already anywhere -td

6 Changing defaults

The purpose of LilyPond's design is to provide the finest output quality as a default. Nevertheless, it may happen that you need to change this default layout. The layout is controlled through a large number of proverbial 'knobs and switches.' This chapter does not list each and every knob. Rather, it outlines what groups of controls are available and explains how to lookup which knob to use for a particular effect.

The controls available for tuning are described in a separate document, the Internals Reference manual. That manual lists all different variables, functions and options available in LilyPond. It is written as a HTML document, which is available on-line, but is also included with the LilyPond documentation package.

There are four areas where the default settings may be changed:

- Automatic notation: changing the automatic creation of notation elements. For example, changing the beaming rules.
- Output: changing the appearance of individual objects. For example, changing stem directions or the location of subscripts.
- Context: changing aspects of the translation from music events to notation. For example, giving each staff a separate time signature.
- Page layout: changing the appearance of the spacing, line breaks, and page dimensions. These modifications are discussed in Chapter 4 [Non-musical notation], page 198, and Chapter 5 [Spacing issues], page 208.

Internally, LilyPond uses Scheme (a LISP dialect) to provide infrastructure. Overriding layout decisions in effect accesses the program internals, which requires Scheme input. Scheme elements are introduced in a `.ly` file with the hash mark `#`.[1]

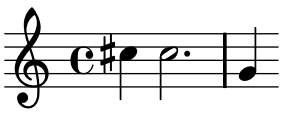
6.1 Interpretation contexts

This section describes what contexts are, and how to modify them.

6.1.1 Contexts explained

When music is printed, a lot of notational elements must be added to the output. For example, compare the input and output of the following example:

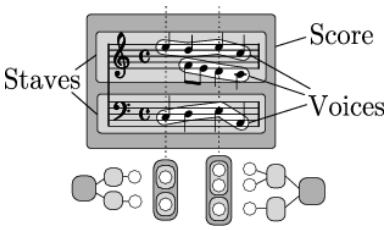
```
cis4 cis2. g4
```



The input is rather sparse, but in the output, bar lines, accidentals, clef, and time signature are added. LilyPond *interprets* the input. During this step, the musical information is inspected in time order, similar to reading a score from left to right. While reading the input, the program remembers where measure boundaries are, and which pitches require explicit accidentals. This information can be presented on several levels. For example, the effect of an accidental is limited to a single staff, while a bar line must be synchronized across the entire score.

[1] learning manual, ⟨undefined⟩ [Scheme tutorial], page ⟨undefined⟩ , contains a short tutorial on entering numbers, lists, strings, and symbols in Scheme.

Within LilyPond, these rules and bits of information are grouped in *Contexts*. Some examples of contexts are `Voice`, `Staff`, and `Score`. They are hierarchical, for example: a `Staff` can contain many `Voice`s, and a `Score` can contain many `Staff` contexts.



Each context has the responsibility for enforcing some notation rules, creating some notation objects and maintaining the associated properties. For example, the `Voice` context may introduce an accidental and then the `Staff` context maintains the rule to show or suppress the accidental for the remainder of the measure. The synchronization of bar lines is handled at `Score` context.

However, in some music we may not want the bar lines to be synchronized – consider a polymetric score in 4/4 and 3/4 time. In such cases, we must modify the default settings of the `Score` and `Staff` contexts.

For very simple scores, contexts are created implicitly, and you need not be aware of them. For larger pieces, such as anything with more than one staff, they must be created explicitly to make sure that you get as many staves as you need, and that they are in the correct order. For typesetting pieces with specialized notation, it can be useful to modify existing or to define new contexts.

A complete description of all available contexts is in the program reference, see Translation ↦ Context.

6.1.2 Creating contexts

For scores with only one voice and one staff, contexts are created automatically. For more complex scores, it is necessary to create them by hand. There are three commands that do this.

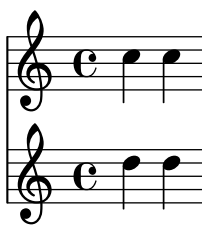
- The easiest command is `\new`, and it also the quickest to type. It is prepended to a music expression, for example

  ```
  \new type music expression
  ```

  where *type* is a context name (like `Staff` or `Voice`). This command creates a new context, and starts interpreting the *music expression* with that.

  A practical application of `\new` is a score with many staves. Each part that should be on its own staff, is preceded with `\new Staff`.

  ```
  <<
    \new Staff { c4 c }
    \new Staff { d4 d }
  >>
  ```



The `\new` command may also give a name to the context,

```
\new type = id music
```

However, this user specified name is only used if there is no other context already earlier with the same name.

- Like `\new`, the `\context` command also directs a music expression to a context object, but gives the context an explicit name. The syntax is

  ```
  \context type = id music
  ```

  This form will search for an existing context of type *type* called *id*. If that context does not exist yet, a new context with the specified name is created. This is useful if the context is referred to later on. For example, when setting lyrics the melody is in a named context

  ```
  \context Voice = "tenor" music
  ```

  so the texts can be properly aligned to its notes,

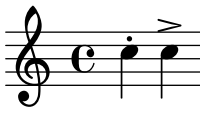
  ```
  \new Lyrics \lyricsto "tenor" lyrics
  ```

  Another possible use of named contexts is funneling two different music expressions into one context. In the following example, articulations and notes are entered separately,

  ```
  music = { c4 c4 }
  arts = { s4-. s4-> }
  ```

  They are combined by sending both to the same `Voice` context,

  ```
  <<
    \new Staff \context Voice = "A" \music
    \context Voice = "A" \arts
  >>
  ```

  With this mechanism, it is possible to define an Urtext (original edition), with the option to put several distinct articulations on the same notes.

- The third command for creating contexts is

  ```
  \context type music
  ```

  This is similar to `\context` with `= id`, but matches any context of type *type*, regardless of its given name.

  This variant is used with music expressions that can be interpreted at several levels. For example, the `\applyOutput` command (see Section 7.5.2 [Running a function on all layout objects], page 283). Without an explicit `\context`, it is usually applied to `Voice`

  ```
  \applyOutput #'context #function   % apply to Voice
  ```

  To have it interpreted at the `Score` or `Staff` level use these forms

  ```
  \applyOutput #'Score #function
  \applyOutput #'Staff #function
  ```

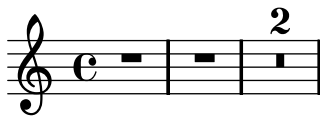
6.1.3 Changing context properties on the fly

Each context can have different *properties*, variables contained in that context. They can be changed during the interpretation step. This is achieved by inserting the `\set` command in the music,

```
\set context.prop = #value
```

For example,

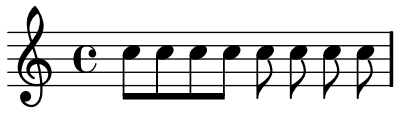
```
R1*2
\set Score.skipBars = ##t
R1*2
```



This command skips measures that have no notes. The result is that multi-rests are condensed. The value assigned is a Scheme object. In this case, it is `#t`, the boolean True value.

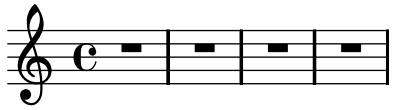
If the *context* argument is left out, then the current bottom-most context (typically `ChordNames`, `Voice`, or `Lyrics`) is used. In this example,

```
c8 c c c
\set autoBeaming = ##f
c8 c c c
```



the *context* argument to `\set` is left out, so automatic beaming is switched off in the current `Voice`. Note that the bottom-most context does not always contain the property that you wish to change – for example, attempting to set the `skipBars` property (of the bottom-most context, in this case `Voice`) will have no effect.

```
R1*2
\set skipBars = ##t
R1*2
```



Contexts are hierarchical, so if a bigger context was specified, for example `Staff`, then the change would also apply to all `Voice`s in the current stave. The change is applied 'on-the-fly', during the music, so that the setting only affects the second group of eighth notes.

There is also an `\unset` command,

`\unset` *`context.prop`*

which removes the definition of *prop*. This command removes the definition only if it is set in *context*, so

```
\set Staff.autoBeaming = ##f
```

introduces a property setting at `Staff` level. The setting also applies to the current `Voice`. However,

```
\unset Voice.autoBeaming
```

does not have any effect. To cancel this setting, the `\unset` must be specified on the same level as the original `\set`. In other words, undoing the effect of `Staff.autoBeaming = ##f` requires

```
\unset Staff.autoBeaming
```

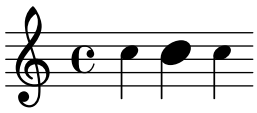
Like `\set`, the *context* argument does not have to be specified for a bottom context, so the two statements

```
\set Voice.autoBeaming = ##t
\set autoBeaming = ##t
```

are equivalent.

Settings that should only apply to a single time-step can be entered with `\once`, for example in

```
c4
\once \set fontSize = #4.7
c4
c4
```



the property `fontSize` is unset automatically after the second note.

A full description of all available context properties is in the program reference, see Translation ↦ Tunable context properties.

6.1.4 Modifying context plug-ins

Notation contexts (like `Score` and `Staff`) not only store properties, they also contain plug-ins called 'engravers' that create notation elements. For example, the `Voice` context contains a `Note_head_engraver` and the `Staff` context contains a `Key_signature_engraver`.

For a full a description of each plug-in, see Internals Reference ↦ Translation ↦ Engravers. Every context described in Internals Reference ↦ Translation ↦ Context. lists the engravers used for that context.

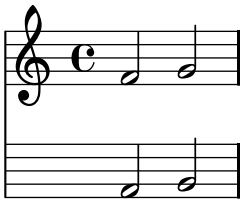
It can be useful to shuffle around these plug-ins. This is done by starting a new context with `\new` or `\context`, and modifying it,

```
\new context \with {
  \consists ...
  \consists ...
  \remove ...
  \remove ...
  etc.
}
{
  ..music..
}
```

where the ... should be the name of an engraver. Here is a simple example which removes `Time_signature_engraver` and `Clef_engraver` from a `Staff` context,

```
<<
  \new Staff {
    f2 g
  }
  \new Staff \with {
     \remove "Time_signature_engraver"
     \remove "Clef_engraver"
  } {
    f2 g2
  }
```

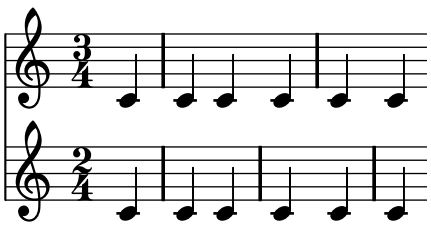
```
>>
```



In the second staff there are no time signature or clef symbols. This is a rather crude method of making objects disappear since it will affect the entire staff. This method also influences the spacing, which may or may not be desirable. A more sophisticated method of blanking objects is shown in learning manual, ⟨undefined⟩ [Common tweaks], page ⟨undefined⟩ .

The next example shows a practical application. Bar lines and time signatures are normally synchronized across the score. This is done by the `Timing_translator` and `Default_bar_line_engraver`. This plug-in keeps an administration of time signature, location within the measure, etc. By moving thes engraver from `Score` to `Staff` context, we can have a score where each staff has its own time signature.

```
\new Score \with {
  \remove "Timing_translator"
  \remove "Default_bar_line_engraver"
} <<
  \new Staff \with {
    \consists "Timing_translator"
    \consists "Default_bar_line_engraver"
  } {
      \time 3/4
      c4 c c c c c
  }
  \new Staff \with {
    \consists "Timing_translator"
    \consists "Default_bar_line_engraver"
  } {
       \time 2/4
       c4 c c c c c
  }
>>
```



6.1.5 Layout tunings within contexts

Each context is responsible for creating certain types of graphical objects. The settings used for printing these objects are also stored by context. By changing these settings, the appearance of objects can be altered.

The syntax for this is

```
\override context.name #'property = #value
```

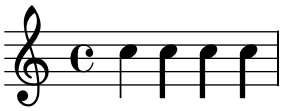
Here *name* is the name of a graphical object, like `Stem` or `NoteHead`, and *property* is an internal variable of the formatting system ('grob property' or 'layout property'). The latter is a symbol, so it must be quoted. The subsection Section 6.2.1 [Constructing a tweak], page 259, explains what to fill in for *name*, *property*, and *value*. Here we only discuss the functionality of this command.

The command

```
\override Staff.Stem #'thickness = #4.0
```

makes stems thicker (the default is 1.3, with staff line thickness as a unit). Since the command specifies `Staff` as context, it only applies to the current staff. Other staves will keep their normal appearance. Here we see the command in action:

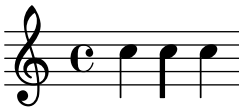
```
c4
\override Staff.Stem #'thickness = #4.0
c4
c4
c4
```



The `\override` command changes the definition of the `Stem` within the current `Staff`. After the command is interpreted all stems are thickened.

Analogous to `\set`, the *context* argument may be left out, causing the default context `Voice` to be used. Adding `\once` applies the change during one timestep only.

```
c4
\once \override Stem #'thickness = #4.0
c4
c4
```



The `\override` must be done before the object is started. Therefore, when altering *Spanner* objects such as slurs or beams, the `\override` command must be executed at the moment when the object is created. In this example,

```
\override Slur #'thickness = #3.0
c8[( c
\override Beam #'thickness = #0.6
c8 c])
```

the slur is fatter but the beam is not. This is because the command for `Beam` comes after the Beam is started, so it has no effect.

Analogous to `\unset`, the `\revert` command for a context undoes an `\override` command; like with `\unset`, it only affects settings that were made in the same context. In other words, the `\revert` in the next example does not do anything.

```
\override Voice.Stem #'thickness = #4.0
\revert Staff.Stem #'thickness
```

Some tweakable options are called 'subproperties' and reside inside properties. To tweak those, use commands of the form

```
\override context.name #'property #'subproperty = #value
```

such as

```
\override Stem #'details #'beamed-lengths = #'(4 4 3)
```

See also

Internals: `OverrideProperty`, `RevertProperty`, `PropertySet`, `Backend`, and `All layout objects`.

Known issues and warnings

The back-end is not very strict in type-checking object properties. Cyclic references in Scheme values for properties can cause hangs or crashes, or both.

6.1.6 Changing context default settings

The adjustments of the previous subsections (Section 6.1.3 [Changing context properties on the fly], page 251, Section 6.1.4 [Modifying context plug-ins], page 253, and Section 6.1.5 [Layout tunings within contexts], page 254) can also be entered separately from the music in the `\layout` block,

```
\layout {
  ...
  \context {
    \Staff

    \set fontSize = #-2
    \override Stem #'thickness = #4.0
    \remove "Time_signature_engraver"
  }
}
```

The `\Staff` command brings in the existing definition of the staff context so that it can be modified.

The statements

```
\set fontSize = #-2
\override Stem #'thickness = #4.0
\remove "Time_signature_engraver"
```

affect all staves in the score. Other contexts can be modified analogously.

The `\set` keyword is optional within the `\layout` block, so

```
\context {
  ...
  fontSize = #-2
}
```

will also work.

Known issues and warnings

It is not possible to collect context changes in a variable and apply them to a `\context` definition by referring to that variable.

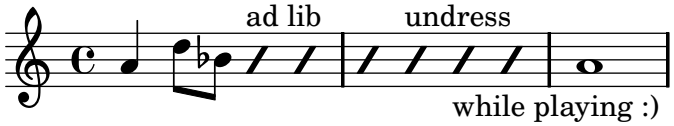
The `\RemoveEmptyStaffContext` will overwrite your current `\Staff` settings. If you wish to change the defaults for a staff which uses `\RemoveEmptyStaffContext`, you must do so after calling `\RemoveEmptyStaffContext`, ie

```
\layout {
  \context {
    \RemoveEmptyStaffContext

    \override Stem #'thickness = #4.0
  }
}
```

6.1.7 Defining new contexts

Specific contexts, like `Staff` and `Voice`, are made of simple building blocks. It is possible to create new types of contexts with different combinations of engraver plug-ins.

The next example shows how to build a different type of `Voice` context from scratch. It will be similar to `Voice`, but only prints centered slash note heads. It can be used to indicate improvisation in jazz pieces,



These settings are defined within a `\context` block inside a `\layout` block,

```
\layout {
  \context {
    ...
  }
}
```

In the following discussion, the example input shown should go in place of the ... in the previous fragment.

First it is necessary to define a name for the new context:

```
\name ImproVoice
```

Since it is similar to the `Voice`, we want commands that work on (existing) `Voice`s to remain working. This is achieved by giving the new context an alias `Voice`,

```
\alias Voice
```

The context will print notes and instructive texts, so we need to add the engravers which provide this functionality,

```
\consists Note_heads_engraver
\consists Text_engraver
```

but we only need this on the center line,

```
\consists Pitch_squash_engraver
squashedPosition = #0
```

The `Pitch_squash_engraver` modifies note heads (created by `Note_heads_engraver`) and sets their vertical position to the value of `squashedPosition`, in this case `0`, the center line.

The notes look like a slash, and have no stem,

```
\override NoteHead #'style = #'slash
\override Stem #'transparent = ##t
```

All these plug-ins have to cooperate, and this is achieved with a special plug-in, which must be marked with the keyword `\type`. This should always be `Engraver_group`,

```
\type "Engraver_group"
```

Put together, we get

```
\context {
  \name ImproVoice
  \type "Engraver_group"
  \consists "Note_heads_engraver"
  \consists "Text_engraver"
  \consists Pitch_squash_engraver
  squashedPosition = #0
  \override NoteHead #'style = #'slash
  \override Stem #'transparent = ##t
  \alias Voice
}
```

Contexts form hierarchies. We want to hang the `ImproVoice` under `Staff`, just like normal `Voice`s. Therefore, we modify the `Staff` definition with the `\accepts` command,

```
\context {
  \Staff
  \accepts ImproVoice
}
```

The opposite of `\accepts` is `\denies`, which is sometimes needed when reusing existing context definitions.

Putting both into a `\layout` block, like

```
\layout {
  \context {
    \name ImproVoice
    ...
  }
  \context {
    \Staff
    \accepts "ImproVoice"
  }
}
```

Then the output at the start of this subsection can be entered as

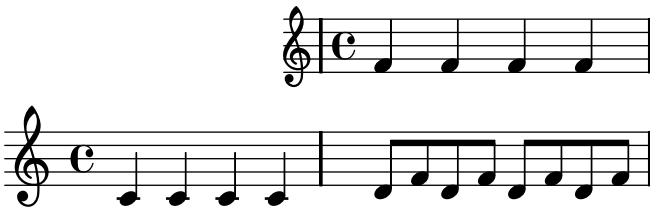
```
\relative c'' {
  a4 d8 bes8
  \new ImproVoice {
```

```
      c4^"ad lib" c
      c4 c^"undress"
      c c_"while playing :)"
    }
    a1
}
```

6.1.8 Aligning contexts

New contexts may be aligned above or below exisiting contexts. This could be useful in setting up a vocal staff (learning manual, ⟨undefined⟩ [Vocal ensembles], page ⟨undefined⟩) and in ossia,

FIXME: this section doens't work in pdf. (?)



6.1.9 Vertical grouping of grobs

The VerticalAlignment and VerticalAxisGroup grobs work together. VerticalAxisGroup groups together different grobs like Staff, Lyrics, etc. VerticalAlignment then vertically aligns the different grobs grouped together by VerticalAxisGroup. There is usually only one VerticalAlignment per score but every Staff, Lyrics, etc. has its own VerticalAxisGroup.

6.2 The `\override` command

In the previous section, we have already touched on a command that changes layout details: the `\override` command. In this section, we will look in more detail at how to use the command in practice. The general syntax of this command is:

```
\override context.layout_object #'layout_property = #value
```

This will set the *layout_property* of the specified *layout_object*, which is a member of the *context*, to the *value*.

6.2.1 Constructing a tweak

Commands which change output generally look like

```
\override Voice.Stem #'thickness = #3.0
```

To construct this tweak we must determine these bits of information:

- the context: here `Voice`.
- the layout object: here `Stem`.
- the layout property: here `thickness`.
- a sensible value: here `3.0`.

Some tweakable options are called 'subproperties' and reside inside properties. To tweak those, use commands in the form

```
\override Stem #'details #'beamed-lengths = #'(4 4 3)
```

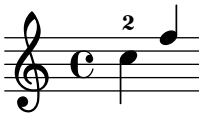
For many properties, regardless of the data type of the property, setting the property to false (`##f`) will result in turning it off, causing Lilypond to ignore that property entirely. This is particularly useful for turning off grob properties which may otherwise be causing problems.

We demonstrate how to glean this information from the notation manual and the program reference.

6.2.2 Navigating the program reference

Suppose we want to move the fingering indication in the fragment below:

```
c-2
\stemUp
f
```



If you visit the documentation on fingering instructions (in Section 1.7.1.2 [Fingering instructions], page 106), you will notice:

See also

Internals Reference: `Fingering`.

The programmer's reference is available as an HTML document. It is highly recommended that you read it in HTML form, either online or by downloading the HTML documentation. This section will be much more difficult to understand if you are using the PDF manual.

Follow the link to `Fingering`. At the top of the page, you will see

> Fingering objects are created by: `Fingering_engraver` and `New_fingering_engraver`.

By following related links inside the program reference, we can follow the flow of information within the program:

- `Fingering`: `Fingering` objects are created by: `Fingering_engraver`
- `Fingering_engraver`: Music types accepted: `fingering-event`
- `fingering-event`: Music event type `fingering-event` is in Music expressions named `FingerEvent`

This path goes against the flow of information in the program: it starts from the output, and ends at the input event. You could also start at an input event, and read with the flow of information, eventually ending up at the output object(s).

The program reference can also be browsed like a normal document. It contains chapters on `Music definitions` on `Translation`, and the `Backend`. Every chapter lists all the definitions used and all properties that may be tuned.

6.2.3 Layout interfaces

The HTML page that we found in the previous section describes the layout object called `Fingering`. Such an object is a symbol within the score. It has properties that store numbers (like thicknesses and directions), but also pointers to related objects. A layout object is also called a *Grob*, which is short for Graphical Object. For more details about Grobs, see `grob-interface`.

The page for `Fingering` lists the definitions for the `Fingering` object. For example, the page says

`padding` (dimension, in staff space):

`0.5`

which means that the number will be kept at a distance of at least 0.5 of the note head.

Each layout object may have several functions as a notational or typographical element. For example, the Fingering object has the following aspects

- Its size is independent of the horizontal spacing, unlike slurs or beams.
- It is a piece of text. Granted, it is usually a very short text.
- That piece of text is typeset with a font, unlike slurs or beams.
- Horizontally, the center of the symbol should be aligned to the center of the note head.
- Vertically, the symbol is placed next to the note and the staff.
- The vertical position is also coordinated with other superscript and subscript symbols.

Each of these aspects is captured in so-called *interface*s, which are listed on the `Fingering` page at the bottom

> This object supports the following interfaces: `item-interface`, `self-alignment-interface`, `side-position-interface`, `text-interface`, `text-script-interface`, `font-interface`, `finger-interface`, and `grob-interface`.

Clicking any of the links will take you to the page of the respective object interface. Each interface has a number of properties. Some of them are not user-serviceable ('Internal properties'), but others can be modified.

We have been talking of *the* `Fingering` object, but actually it does not amount to much. The initialization file (see learning manual, ⟨undefined⟩ [Default files], page ⟨undefined⟩) '`scm/define-grobs.scm`' shows the soul of the 'object',

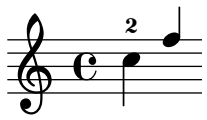
```
(Fingering
  . ((padding . 0.5)
     (avoid-slur . around)
     (slur-padding . 0.2)
     (staff-padding . 0.5)
     (self-alignment-X . 0)
     (self-alignment-Y . 0)
     (script-priority . 100)
     (stencil . ,ly:text-interface::print)
     (direction . ,ly:script-interface::calc-direction)
     (font-encoding . fetaNumber)
     (font-size . -5)  ; don't overlap when next to heads.
     (meta . ((class . Item)
     (interfaces . (finger-interface
                    font-interface
                    text-script-interface
                    text-interface
                    side-position-interface
                    self-alignment-interface
                    item-interface))))))
```

As you can see, the `Fingering` object is nothing more than a bunch of variable settings, and the webpage in the Internals Reference is directly generated from this definition.

6.2.4 Determining the grob property

Recall that we wanted to change the position of the **2** in

```
c-2
\stemUp
f
```



Since the **2** is vertically positioned next to its note, we have to meddle with the interface associated with this positioning. This is done using `side-position-interface`. The page for this interface says

> `side-position-interface`
>
> Position a victim object (this one) next to other objects (the support). The property `direction` signifies where to put the victim object relative to the support (left or right, up or down?)

Below this description, the variable `padding` is described as

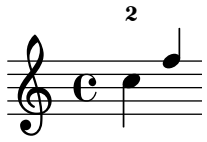
> `padding` (dimension, in staff space)
>
> Add this much extra space between objects that are next to each other.

By increasing the value of `padding`, we can move the fingering away from the note head. The following command inserts 3 staff spaces of white between the note and the fingering:

```
\once \override Voice.Fingering #'padding = #3
```

Inserting this command before the Fingering object is created, i.e., before `c2`, yields the following result:

```
\once \override Voice.Fingering #'padding = #3
c-2
\stemUp
f
```



In this case, the context for this tweak is `Voice`. This fact can also be deduced from the program reference, for the page for the `Fingering_engraver` plug-in says

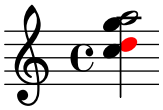
> Fingering_engraver is part of contexts: ... `Voice`

6.2.5 Objects connected to the input

In some cases, it is possible to take a short-cut for tuning graphical objects. For objects that result directly from a piece of the input, you can use the `\tweak` function, for example

```
<
  c
  \tweak #'color #red d
  g
  \tweak #'duration-log #1  a
>4-\tweak #'padding #10 -.
```

.



As you can see, properties are set in the objects directly, without mentioning the grob name or context where this should be applied.

This technique only works for objects that are directly connected to an `Event` from the input, for example

- note heads, caused by chord-pitch (i.e., notes inside a chord)
- articulation signs, caused by articulation instructions

It notably does not work for stems and accidentals (these are caused by note heads, not by music events) or clefs (these are not caused by music inputs, but rather by the change of a property value).

There are very few objects which are *directly* connected to output. A normal note (like `c4`) is not directly connected to output, so

```
\tweak #'color #red c4
```

does not change color. See Section 7.3.1 [Displaying music expressions], page 274, for details.

6.2.6 Using Scheme code instead of `\tweak`

The main disadvantage of `\tweak` is its syntactical inflexibility. For example, the following produces a syntax error.

```
F = \tweak #'font-size #-3 -\flageolet

\relative c'' {
  c4^\F c4_\F
}
```

With other words, `\tweak` doesn't behave like an articulation regarding the syntax; in particular, it can't be attached with `^` and `_`.

Using Scheme, this problem can be circumvented. The route to the result is given in Section 7.3.4 [Adding articulation to notes (example)], page 277, especially how to use `\displayMusic` as a helping guide.

```
F = #(let ((m (make-music 'ArticulationEvent
                          'articulation-type "flageolet")))
        (set! (ly:music-property m 'tweaks)
              (acons 'font-size -3
                     (ly:music-property m 'tweaks)))
        m)

\relative c'' {
  c4^\F c4_\F
}
```

Here, the `tweaks` properties of the flageolet object `m` (created with `make-music`) are extracted with `ly:music-property`, a new key-value pair to change the font size is prepended to the property list with the `acons` Scheme function, and the result is finally written back with `set!`. The last element of the `let` block is the return value, `m` itself.

6.2.7 \set vs. \override

We have seen two methods of changing properties: `\set` and `\override`. There are actually two different kinds of properties.

Contexts can have properties, which are usually named in `studlyCaps`. They mostly control the translation from music to notatino, eg. `localKeySignature` (for determining whether to print accidentals), `measurePosition` (for determining when to print a bar line). Context properties can change value over time while interpreting a piece of music; `measurePosition` is an obvious example of this. Context properties are modified with `\set`.

There is a special type of context property: the element description. These properties are named in `StudlyCaps` (starting with capital letters). They contain the 'default settings' for said graphical object as an association list. See '`scm/define-grobs.scm`' to see what kind of settings there are. Element descriptions may be modified with `\override`.

`\override` is actually a shorthand;

```
\override context.name #'property = #value
```

is more or less equivalent to

```
\set context.name #'property = #(cons (cons 'property value) <previous value of context)
```

The value of `context` (the alist) is used to initalize the properties of individual grobs. Grobs also have properties, named in Scheme style, with `dashed-words`. The values of grob properties change during the formatting process: formatting basically amounts to computing properties using callback functions.

`fontSize` is a special property: it is equivalent to entering `\override ... #'font-size` for all pertinent objects. Since this is a common change, the special property (modified with `\set`) was created.

6.2.8 Difficult tweaks

There are a few classes of difficult adjustments.

- One type of difficult adjustment is the appearance of spanner objects, such as slur and tie. Initially, only one of these objects is created, and they can be adjusted with the normal mechanism. However, in some cases the spanners cross line breaks. If this happens, these objects are cloned. A separate object is created for every system that it is in. These are clones of the original object and inherit all properties, including `\override`s.

  In other words, an `\override` always affects all pieces of a broken spanner. To change only one part of a spanner at a line break, it is necessary to hook into the formatting process. The `after-line-breaking` callback contains the Scheme procedure that is called after the line breaks have been determined, and layout objects have been split over different systems.

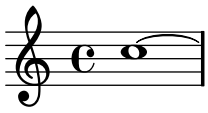
  In the following example, we define a procedure `my-callback`. This procedure

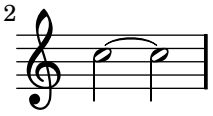
  - determines if we have been split across line breaks
  - if yes, retrieves all the split objects
  - checks if we are the last of the split objects
  - if yes, it sets `extra-offset`.

  This procedure is installed into `Tie`, so the last part of the broken tie is translated up.

```
#(define (my-callback grob)
  (let* (
         ; have we been split?
         (orig (ly:grob-original grob))
```

```
          ; if yes, get the split pieces (our siblings)
          (siblings (if (ly:grob? orig)
                        (ly:spanner-broken-into orig) '() )))

   (if (and (>= (length siblings) 2)
            (eq? (car (last-pair siblings)) grob))
     (ly:grob-set-property! grob 'extra-offset '(-2 . 5)))))

\relative c'' {
  \override Tie #'after-line-breaking =
  #my-callback
  c1 ~ \break c2 ~ c
}
```





When applying this trick, the new `after-line-breaking` callback should also call the old one `after-line-breaking`, if there is one. For example, if using this with `Hairpin`, `ly:hairpin::after-line-breaking` should also be called.

- Some objects cannot be changed with `\override` for technical reasons. Examples of those are `NonMusicalPaperColumn` and `PaperColumn`. They can be changed with the `\overrideProperty` function, which works similar to `\once \override`, but uses a different syntax.

  ```
  \overrideProperty
  #"Score.NonMusicalPaperColumn"  % Grob name
  #'line-break-system-details     % Property name
  #'((next-padding . 20))         % Value
  ```

  Note, however, that `\override`, applied to `NoteMusicalPaperColumn` and `PaperColumn`, still works as expected within `\context` blocks.

7 Interfaces for programmers

Advanced tweaks may be performed by using Scheme. If you are not familiar with Scheme, you may wish to read our learning manual, ⟨undefined⟩ [Scheme tutorial], page ⟨undefined⟩ .

7.1 Music functions

This section discusses how to create music functions within LilyPond.

7.1.1 Overview of music functions

Making a function which substitutes a variable into LilyPond code is easy. The general form of these functions is

```
function =
#(define-music-function (parser location var1 var2... )
                         (var1-type? var2-type?...)
  #{
    ...music...
  #})
```

where

argi	*i*th variable
argi-type?	type of variable
...music...	normal LilyPond input, using variables as `#$var1`.

There following input types may be used as variables in a music function. This list is not exhaustive; see other documentation specifically about Scheme for more variable types.

Input type	*argi-type?* **notation**
Integer	`integer?`
Float (decimal number)	`number?`
Text string	`string?`
Markup	`markup?`
Music expression	`ly:music?`
A pair of variables	`pair?`

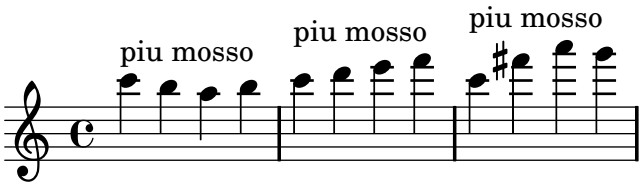
The `parser` and `location` argument are mandatory, and are used in some advanced situations. The `parser` argument is used to access to the value of another LilyPond variable. The `location` argument is used to set the 'origin' of the music expression that is built by the music function, so that in case of a syntax error LilyPond can tell the user an appropriate place to look in the input file.

7.1.2 Simple substitution functions

Here is a simple example,

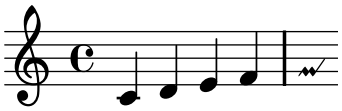
```
padText = #(define-music-function (parser location padding) (number?)
  #{
    \once \override TextScript #'padding = #$padding
  #})

\relative c''' {
  c4^"piu mosso" b a b
  \padText #1.8
  c4^"piu mosso" d e f
  \padText #2.6
```

```
  c4^"piu mosso" fis a g
}
```



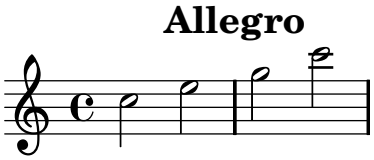
Music expressions may be substituted as well,

```
custosNote = #(define-music-function (parser location note)
                                     (ly:music?)
  #{
    \once \override Voice.NoteHead #'stencil =
      #ly:text-interface::print
    \once \override Voice.NoteHead #'text =
      \markup \musicglyph #"custodes.mensural.u0"
    \once \override Voice.Stem #'stencil = ##f
    $note
  #})

{ c' d' e' f' \custosNote g' }
```



Multiple variables may be used,

```
tempoMark = #(define-music-function (parser location padding marktext)
                                    (number? string?)
#{
  \once \override Score . RehearsalMark #'padding = $padding
  \once \override Score . RehearsalMark #'extra-spacing-width = #'(+inf.0 . -inf.0)
  \mark \markup { \bold $marktext }
#})

\relative c'' {
c2 e
\tempoMark #3.0 #"Allegro"
g c
}
```



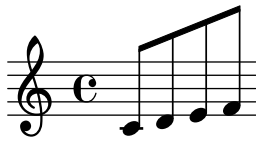
7.1.3 Paired substitution functions

Some `\override` commands require a pair of numbers (called a `cons cell` in Scheme). To pass these numbers into a function, either use a `pair?` variable, or insert the `cons` into the music function.

```
manualBeam =
#(define-music-function (parser location beg-end)
                        (pair?)
#{
  \once \override Beam #'positions = #$beg-end
#})

\relative {
  \manualBeam #'(3 . 6) c8 d e f
}
```

or

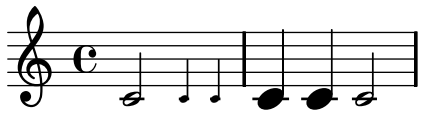
```
manualBeam =
#(define-music-function (parser location beg end)
                        (number? number?)
#{
  \once \override Beam #'positions = #(cons $beg $end)
#})

\relative {
  \manualBeam #3 #6 c8 d e f
}
```



7.1.4 Mathematics in functions

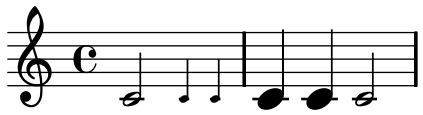
Music functions can involve Scheme programming in addition to simple substitution,

```
AltOn = #(define-music-function (parser location mag) (number?)
  #{ \override Stem #'length = #$(* 7.0 mag)
     \override NoteHead #'font-size =
       #$(inexact->exact (* (/ 6.0 (log 2.0)) (log mag))) #})

AltOff = {
  \revert Stem #'length
  \revert NoteHead #'font-size
}

{ c'2 \AltOn #0.5 c'4 c'
  \AltOn #1.5 c' c' \AltOff c'2 }
```



This example may be rewritten to pass in music expressions,

```
withAlt = #(define-music-function (parser location mag music) (number? ly:music?)
  #{ \override Stem #'length = #$(* 7.0 mag)
     \override NoteHead #'font-size =
       #$(inexact->exact (* (/ 6.0 (log 2.0)) (log mag)))
     $music
     \revert Stem #'length
     \revert NoteHead #'font-size #})

{ c'2 \withAlt #0.5 {c'4 c'}
  \withAlt #1.5 {c' c'} c'2 }
```



7.1.5 Void functions

A music function must return a music expression, but sometimes we may want to have a function which does not involve music (such as turning off Point and Click). To do this, we return a `void` music expression.

That is why the form that is returned is the `(make-music ...)`. With the `'void` property set to `#t`, the parser is told to actually disregard this returned music expression. Thus the important part of the void music function is the processing done by the function, not the music expression that is returned.

```
noPointAndClick =
#(define-music-function (parser location) ()
   (ly:set-option 'point-and-click #f)
   (make-music 'SequentialMusic 'void #t))
...
\noPointAndClick   % disable point and click
```

7.1.6 Functions without arguments

In most cases a function without arguments should be written with an variable,

```
dolce = \markup{ \italic \bold dolce }
```

However, in rare cases it may be useful to create a music function without arguments,

```
displayBarNum =
#(define-music-function (parser location) ()
   (if (eq? #t (ly:get-option 'display-bar-numbers))
       #{ \once \override Score.BarNumber #'break-visibility = ##f #}
       #{#}))
```

To actually display bar numbers where this function is called, invoke lilypond with

```
lilypond -d display-bar-numbers FILENAME.ly
```

7.1.7 Overview of available music functions

The following commands are music functions

`autochange` - *music* (music)
(undocumented; fixme)

`endSpanners` - *music* (music)
(undocumented; fixme)

`musicMap` - *proc* (procedure) *mus* (music)
(undocumented; fixme)

`scoreTweak` - *name* (string)
(undocumented; fixme)

`killCues` - *music* (music)
(undocumented; fixme)

`appoggiatura` - *music* (music)
(undocumented; fixme)

`cueDuring` - *what* (string) *dir* (direction) *main-music* (music)
(undocumented; fixme)

`breathe` -
(undocumented; fixme)

`acciaccatura` - *music* (music)
(undocumented; fixme)

`removeWithTag` - *tag* (symbol) *music* (music)
(undocumented; fixme)

`tocItem` - *text* (markup)
Add a line to the table of content, using the `tocItemMarkup` paper variable markup

`applyContext` - *proc* (procedure)
(undocumented; fixme)

`pitchedTrill` - *main-note* (music) *secondary-note* (music)
(undocumented; fixme)

`assertBeamQuant` - *l* (pair) *r* (pair)
(undocumented; fixme)

`oldaddlyrics` - *music* (music) *lyrics* (music)
(undocumented; fixme)

`addQuote` - *name* (string) *music* (music)
(undocumented; fixme)

`featherDurations` - *factor* (moment) *argument* (music)
(undocumented; fixme)

`transposition` - *pitch-note* (music)
(undocumented; fixme)

`bendAfter` - *delta* (integer)
(undocumented; fixme)

`noPageTurn` -
(undocumented; fixme)

`noPageBreak` -
(undocumented; fixme)

`balloonText` - *offset* (pair of numbers) *text* (markup)
(undocumented; fixme)

`barNumberCheck` - *n* (integer)
(undocumented; fixme)

`addInstrumentDefinition` - *name* (string) *lst* (list)
(undocumented; fixme)

`shiftDurations` - *dur* (integer) *dots* (integer) *arg* (music)
(undocumented; fixme)

`unfoldRepeats` - *music* (music)
(undocumented; fixme)

`applyMusic` - *func* (procedure) *music* (music)
(undocumented; fixme)

`quoteDuring` - *what* (string) *main-music* (music)
(undocumented; fixme)

`keepWithTag` - *tag* (symbol) *music* (music)
(undocumented; fixme)

`displayLilyMusic` - *music* (music)
(undocumented; fixme)

`rightHandFinger` - *finger* (number or string)
(undocumented; fixme)

`bar` - *type* (string)
(undocumented; fixme)

`tag` - *tag* (symbol) *arg* (music)
(undocumented; fixme)

`assertBeamSlope` - *comp* (procedure)
(undocumented; fixme)

`tweak` - *sym* (symbol) *val* (any type) *arg* (music)
(undocumented; fixme)

`transposedCueDuring` - *what* (string) *dir* (direction) *pitch-note* (music) *main-music* (music)
(undocumented; fixme)

`overrideProperty` - *name* (string) *property* (symbol) *value* (any type)
(undocumented; fixme)

`withMusicProperty` - *sym* (symbol) *val* (any type) *music* (music)
(undocumented; fixme)

`octave` - *pitch-note* (music)
(undocumented; fixme)

`applyOutput` - *ctx* (symbol) *proc* (procedure)
(undocumented; fixme)

`afterGrace` - *main* (music) *grace* (music)
(undocumented; fixme)

`allowPageTurn` -
(undocumented; fixme)

`compressMusic` - *fraction* (pair of numbers) *music* (music)
(undocumented; fixme)

`displayMusic` - *music* (music)
(undocumented; fixme)

`pageTurn` -
(undocumented; fixme)

`balloonGrobText` - *grob-name* (symbol) *offset* (pair of numbers) *text* (markup)
(undocumented; fixme)

`includePageLayoutFile` -
(undocumented; fixme)

`instrumentSwitch` - *name* (string)
(undocumented; fixme)

`makeClusters` - *arg* (music)
(undocumented; fixme)

`parallelMusic` - *voice-ids* (list) *music* (music)
(undocumented; fixme)

`resetRelativeOctave` - *reference-note* (music)
(undocumented; fixme)

`label` - *label* (symbol)
(undocumented; fixme)

`grace` - *music* (music)
(undocumented; fixme)

`spacingTweaks` - *parameters* (list)
(undocumented; fixme)

`clef` - *type* (string)
(undocumented; fixme)

`partcombine` - *part1* (music) *part2* (music)
(undocumented; fixme)

`parenthesize` - *arg* (music)
(undocumented; fixme)

`pageBreak` -
(undocumented; fixme)

7.2 Programmer interfaces

This section contains information about mixing LilyPond and Scheme.

7.2.1 Input variables and Scheme

The input format supports the notion of variables: in the following example, a music expression is assigned to a variable with the name `traLaLa`.

```
traLaLa = { c'4 d'4 }
```

There is also a form of scoping: in the following example, the `\layout` block also contains a `traLaLa` variable, which is independent of the outer `\traLaLa`.

```
traLaLa = { c'4 d'4 }
\layout { traLaLa = 1.0 }
```

In effect, each input file is a scope, and all `\header`, `\midi`, and `\layout` blocks are scopes nested inside that toplevel scope.

Both variables and scoping are implemented in the GUILE module system. An anonymous Scheme module is attached to each scope. An assignment of the form

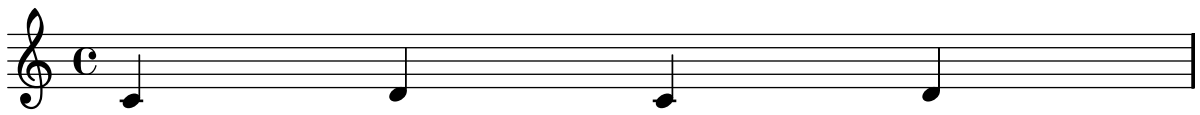
```
traLaLa = { c'4 d'4 }
```

is internally converted to a Scheme definition

```
(define traLaLa Scheme value of `... ')
```

This means that input variables and Scheme variables may be freely mixed. In the following example, a music fragment is stored in the variable `traLaLa`, and duplicated using Scheme. The result is imported in a `\score` block by means of a second variable `twice`:

```
traLaLa = { c'4 d'4 }

%% dummy action to deal with parser lookahead
#(display "this needs to be here, sorry!")

#(define newLa (map ly:music-deep-copy
  (list traLaLa traLaLa)))
#(define twice
  (make-sequential-music newLa))

{ \twice }
```



Due to parser lookahead

In this example, the assignment happens after parser has verified that nothing interesting happens after `traLaLa = { ... }`. Without the dummy statement in the above example, the `newLa` definition is executed before `traLaLa` is defined, leading to a syntax error.

The above example shows how to 'export' music expressions from the input to the Scheme interpreter. The opposite is also possible. By wrapping a Scheme value in the function `ly:export`, a Scheme value is interpreted as if it were entered in LilyPond syntax. Instead of defining `\twice`, the example above could also have been written as

```
...
{ #(ly:export (make-sequential-music (list newLa))) }
```

Scheme code is evaluated as soon as the parser encounters it. To define some Scheme code in a macro (to be called later), use Section 7.1.5 [Void functions], page 269, or

```
#(define (nopc)
  (ly:set-option 'point-and-click #f))

...
#(nopc)
{ c'4 }
```

Known issues and warnings

Mixing Scheme and LilyPond variables is not possible with the `--safe` option.

7.2.2 Internal music representation

When a music expression is parsed, it is converted into a set of Scheme music objects. The defining property of a music object is that it takes up time. Time is a rational number that measures the length of a piece of music in whole notes.

A music object has three kinds of types:

- music name: Each music expression has a name. For example, a note leads to a `NoteEvent`, and `\simultaneous` leads to a `SimultaneousMusic`. A list of all expressions available is in the Internals Reference manual, under `Music expressions`.
- 'type' or interface: Each music name has several 'types' or interfaces, for example, a note is an `event`, but it is also a `note-event`, a `rhythmic-event`, and a `melodic-event`. All classes of music are listed in the Internals Reference, under `Music classes`.
- C++ object: Each music object is represented by an object of the C++ class `Music`.

The actual information of a music expression is stored in properties. For example, a `NoteEvent` has `pitch` and `duration` properties that store the pitch and duration of that note. A list of all properties available is in the internals manual, under `Music properties`.

A compound music expression is a music object that contains other music objects in its properties. A list of objects can be stored in the `elements` property of a music object, or a single 'child' music object in the `element` object. For example, `SequentialMusic` has its children in `elements`, and `GraceMusic` has its single argument in `element`. The body of a repeat is stored in the `element` property of `RepeatedMusic`, and the alternatives in `elements`.

7.3 Building complicated functions

This section explains how to gather the information necessary to create complicated music functions.

7.3.1 Displaying music expressions

When writing a music function it is often instructive to inspect how a music expression is stored internally. This can be done with the music function `\displayMusic`

```
{
  \displayMusic { c'4\f }
}
```

will display

```
(make-music
  'SequentialMusic
  'elements
  (list (make-music
          'EventChord
          'elements
          (list (make-music
                  'NoteEvent
                  'duration
                  (ly:make-duration 2 0 1 1)
                  'pitch
                  (ly:make-pitch 0 0 0))
                (make-music
                  'AbsoluteDynamicEvent
                  'text
                  "f")))))
```

By default, LilyPond will print these messages to the console along with all the other messages. To split up these messages and save the results of `\display{STUFF}`, redirect the output to a file.

```
lilypond file.ly >display.txt
```

With a bit of reformatting, the above information is easier to read,

```
(make-music 'SequentialMusic
  'elements (list (make-music 'EventChord
                    'elements (list (make-music 'NoteEvent
                                      'duration (ly:make-duration 2 0 1 1)
                                      'pitch (ly:make-pitch 0 0 0))
                                    (make-music 'AbsoluteDynamicEvent
                                      'text "f")))))
```

A `{ ... }` music sequence has the name `SequentialMusic`, and its inner expressions are stored as a list in its `'elements` property. A note is represented as an `EventChord` expression, containing a `NoteEvent` object (storing the duration and pitch properties) and any extra information (in this case, an `AbsoluteDynamicEvent` with a `"f"` text property.

7.3.2 Music properties

The `NoteEvent` object is the first object of the `'elements` property of `someNote`.

```
someNote = c'
\displayMusic \someNote
===>
(make-music
  'EventChord
  'elements
  (list (make-music
          'NoteEvent
          'duration
          (ly:make-duration 2 0 1 1)
          'pitch
          (ly:make-pitch 0 0 0))))
```

The `display-scheme-music` function is the function used by `\displayMusic` to display the Scheme representation of a music expression.

```
#(display-scheme-music (first (ly:music-property someNote 'elements)))
===>
(make-music
  'NoteEvent
  'duration
  (ly:make-duration 2 0 1 1)
  'pitch
  (ly:make-pitch 0 0 0))
```

Then the note pitch is accessed through the `'pitch` property of the `NoteEvent` object,

```
#(display-scheme-music
   (ly:music-property (first (ly:music-property someNote 'elements))
                      'pitch))
===>
(ly:make-pitch 0 0 0)
```

The note pitch can be changed by setting this 'pitch property,

```
#(set! (ly:music-property (first (ly:music-property someNote 'elements))
                          'pitch)
       (ly:make-pitch 0 1 0)) ;; set the pitch to d'.
\displayLilyMusic \someNote
===>
d'
```

7.3.3 Doubling a note with slurs (example)

Suppose we want to create a function which translates input like `a` into `a( a)`. We begin by examining the internal representation of the music we want to end up with.

```
\displayMusic{ a'( a') }
===>
(make-music
  'SequentialMusic
  'elements
  (list (make-music
          'EventChord
          'elements
          (list (make-music
                  'NoteEvent
                  'duration
                  (ly:make-duration 2 0 1 1)
                  'pitch
                  (ly:make-pitch 0 5 0))
                (make-music
                  'SlurEvent
                  'span-direction
                  -1)))
        (make-music
          'EventChord
          'elements
          (list (make-music
                  'NoteEvent
                  'duration
                  (ly:make-duration 2 0 1 1)
                  'pitch
                  (ly:make-pitch 0 5 0))
                (make-music
                  'SlurEvent
                  'span-direction
                  1)))))
```

The bad news is that the `SlurEvent` expressions must be added 'inside' the note (or more precisely, inside the `EventChord` expression).

Now we examine the input,

```
(make-music
  'SequentialMusic
  'elements
  (list (make-music
          'EventChord
          'elements
          (list (make-music
                  'NoteEvent
                  'duration
                  (ly:make-duration 2 0 1 1)
                  'pitch
                  (ly:make-pitch 0 5 0))))))
```

So in our function, we need to clone this expression (so that we have two notes to build the sequence), add `SlurEvents` to the `'elements` property of each one, and finally make a `SequentialMusic` with the two `EventChords`.

```
doubleSlur = #(define-music-function (parser location note) (ly:music?)
         "Return: { note ( note ) }.
         `note' is supposed to be an EventChord."
         (let ((note2 (ly:music-deep-copy note)))
           (set! (ly:music-property note 'elements)
                 (cons (make-music 'SlurEvent 'span-direction -1)
                       (ly:music-property note 'elements)))
           (set! (ly:music-property note2 'elements)
                 (cons (make-music 'SlurEvent 'span-direction 1)
                       (ly:music-property note2 'elements)))
           (make-music 'SequentialMusic 'elements (list note note2))))
```

7.3.4 Adding articulation to notes (example)

The easy way to add articulation to notes is to merge two music expressions into one context, as explained in Section 6.1.2 [Creating contexts], page 250. However, suppose that we want to write a music function which does this.

A `$variable` inside the `#{...#}` notation is like using a regular `\variable` in classical LilyPond notation. We know that

```
{ \music -. -> }
```

will not work in LilyPond. We could avoid this problem by attaching the articulation to a fake note,

```
{ << \music s1*0-.-> }
```

but for the sake of this example, we will learn how to do this in Scheme. We begin by examining our input and desired output,

```
%  input
\displayMusic c4
===>
(make-music
  'EventChord
  'elements
  (list (make-music
          'NoteEvent
          'duration
          (ly:make-duration 2 0 1 1)
          'pitch
          (ly:make-pitch -1 0 0))))
=====
%  desired output
\displayMusic c4->
===>
(make-music
  'EventChord
  'elements
  (list (make-music
          'NoteEvent
          'duration
          (ly:make-duration 2 0 1 1)
```

```
        'pitch
        (ly:make-pitch -1 0 0))
      (make-music
        'ArticulationEvent
        'articulation-type
        "marcato")))
```

We see that a note (`c4`) is represented as an `EventChord` expression, with a `NoteEvent` expression in its elements list. To add a marcato articulation, an `ArticulationEvent` expression must be added to the elements property of the `EventChord` expression.

To build this function, we begin with

```
(define (add-marcato event-chord)
  "Add a marcato ArticulationEvent to the elements of `event-chord',
  which is supposed to be an EventChord expression."
  (let ((result-event-chord (ly:music-deep-copy event-chord)))
    (set! (ly:music-property result-event-chord 'elements)
          (cons (make-music 'ArticulationEvent
                  'articulation-type "marcato")
                (ly:music-property result-event-chord 'elements)))
    result-event-chord))
```

The first line is the way to define a function in Scheme: the function name is `add-marcato`, and has one variable called `event-chord`. In Scheme, the type of variable is often clear from its name. (this is good practice in other programming languages, too!)

```
"Add a marcato..."
```

is a description of what the function does. This is not strictly necessary, but just like clear variable names, it is good practice.

```
(let ((result-event-chord (ly:music-deep-copy event-chord)))
```

`let` is used to declare local variables. Here we use one local variable, named `result-event-chord`, to which we give the value `(ly:music-deep-copy event-chord)`. `ly:music-deep-copy` is a function specific to LilyPond, like all functions prefixed by `ly:`. It is use to make a copy of a music expression. Here we copy `event-chord` (the parameter of the function). Recall that our purpose is to add a marcato to an `EventChord` expression. It is better to not modify the `EventChord` which was given as an argument, because it may be used elsewhere.

Now we have a `result-event-chord`, which is a `NoteEventChord` expression and is a copy of `event-chord`. We add the marcato to its elements list property.

```
(set! place new-value)
```

Here, what we want to set (the 'place') is the 'elements' property of `result-event-chord` expression.

```
(ly:music-property result-event-chord 'elements)
```

`ly:music-property` is the function used to access music properties (the `'elements`, `'duration`, `'pitch`, etc, that we see in the `\displayMusic` output above). The new value is the former elements property, with an extra item: the `ArticulationEvent` expression, which we copy from the `\displayMusic` output,

```
(cons (make-music 'ArticulationEvent
        'articulation-type "marcato")
      (ly:music-property result-event-chord 'elements))
```

`cons` is used to add an element to a list without modifying the original list. This is what we want: the same list as before, plus the new `ArticulationEvent` expression. The order inside the elements property is not important here.

Finally, once we have added the marcato articulation to its `elements` property, we can return `result-event-chord`, hence the last line of the function.

Now we transform the `add-marcato` function into a music function,

```
addMarcato = #(define-music-function (parser location event-chord)
                                      (ly:music?)
    "Add a marcato ArticulationEvent to the elements of `event-chord',
    which is supposed to be an EventChord expression."
    (let ((result-event-chord (ly:music-deep-copy event-chord)))
      (set! (ly:music-property result-event-chord 'elements)
            (cons (make-music 'ArticulationEvent
                    'articulation-type "marcato")
                  (ly:music-property result-event-chord 'elements)))
      result-event-chord))
```

We may verify that this music function works correctly,

```
\displayMusic \addMarcato c4
```

7.4 Markup programmer interface

Markups are implemented as special Scheme functions which produce a Stencil object given a number of arguments.

7.4.1 Markup construction in Scheme

The `markup` macro builds markup expressions in Scheme while providing a LilyPond-like syntax. For example,

```
(markup #:column (#:line (#:bold #:italic "hello" #:raise 0.4 "world")
                  #:bigger #:line ("foo" "bar" "baz")))
```

is equivalent to:

```
\markup \column { \line { \bold \italic "hello" \raise #0.4 "world" }
                  \bigger \line { foo bar baz } }
```

This example demonstrates the main translation rules between regular LilyPond markup syntax and Scheme markup syntax.

LilyPond	**Scheme**
`\markup markup1`	`(markup markup1)`
`\markup { markup1 markup2 ... }`	`(markup markup1 markup2 ... )`
`\command`	`#:command`
`\variable`	`variable`
`\center-align { ... }`	`#:center-align ( ... )`
`string`	`"string"`
`#scheme-arg`	`scheme-arg`

The whole Scheme language is accessible inside the `markup` macro. For example, You may use function calls inside `markup` in order to manipulate character strings. This is useful when defining new markup commands (see Section 7.4.3 [New markup command definition], page 280).

Known issues and warnings

The markup-list argument of commands such as `#:line`, `#:center`, and `#:column` cannot be a variable or the result of a function call.

```
(markup #:line (function-that-returns-markups))
```

is invalid. One should use the `make-line-markup`, `make-center-markup`, or `make-column-markup` functions instead,

```
(markup (make-line-markup (function-that-returns-markups)))
```

7.4.2 How markups work internally

In a markup like

```
\raise #0.5 "text example"
```

`\raise` is actually represented by the `raise-markup` function. The markup expression is stored as

```
(list raise-markup 0.5 (list simple-markup "text example"))
```

When the markup is converted to printable objects (Stencils), the `raise-markup` function is called as

```
(apply raise-markup
       \layout object
       list of property alists
       0.5
       the "text example" markup)
```

The `raise-markup` function first creates the stencil for the `text example` string, and then it raises that Stencil by 0.5 staff space. This is a rather simple example; more complex examples are in the rest of this section, and in '`scm/define-markup-commands.scm`'.

7.4.3 New markup command definition

New markup commands can be defined with the `define-markup-command` Scheme macro.

```
(define-markup-command (command-name layout props arg1 arg2 ...)
            (arg1-type? arg2-type? ...)
  ..command body..)
```

The arguments are

argi *i*th command argument

argi-type? a type predicate for the i*th* argument

layout the 'layout' definition

props a list of alists, containing all active properties.

As a simple example, we show how to add a `\smallcaps` command, which selects a small caps font. Normally we could select the small caps font,

```
\markup { \override #'(font-shape . caps) Text-in-caps }
```

This selects the caps font by setting the `font-shape` property to `#'caps` for interpreting `Text-in-caps`.

To make the above available as `\smallcaps` command, we must define a function using `define-markup-command`. The command should take a single argument of type `markup`. Therefore the start of the definition should read

```
(define-markup-command (smallcaps layout props argument) (markup?)
```

What follows is the content of the command: we should interpret the `argument` as a markup, i.e.,

```
(interpret-markup layout ... argument)
```

This interpretation should add `'(font-shape . caps)` to the active properties, so we substitute the following for the ... in the above example:

```
(cons (list '(font-shape . caps) ) props)
```

The variable `props` is a list of alists, and we prepend to it by cons'ing a list with the extra setting.

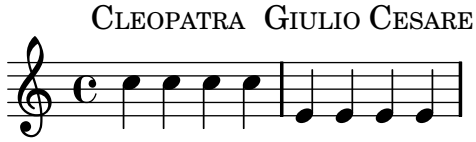
Suppose that we are typesetting a recitative in an opera and we would like to define a command that will show character names in a custom manner. Names should be printed with small caps and moved a bit to the left and top. We will define a `\character` command which takes into account the necessary translation and uses the newly defined `\smallcaps` command:

```
#(define-markup-command (character layout props name) (string?)
  "Print the character name in small caps, translated to the left and
  top.  Syntax: \\character #\"name\""
  (interpret-markup layout props
   (markup #:hspace 0 #:translate (cons -3 1) #:smallcaps name)))
```

There is one complication that needs explanation: texts above and below the staff are moved vertically to be at a certain distance (the `padding` property) from the staff and the notes. To make sure that this mechanism does not annihilate the vertical effect of our `#:translate`, we add an empty string (`#:hspace 0`) before the translated text. Now the `#:hspace 0` will be put above the notes, and the `name` is moved in relation to that empty string. The net effect is that the text is moved to the upper left.

The final result is as follows:

```
{
  c''^\markup \character #"Cleopatra"
  e'^\markup \character #"Giulio Cesare"
}
```



We have used the `caps` font shape, but suppose that our font does not have a small-caps variant. In that case we have to fake the small caps font by setting a string in upcase with the first letter a little larger:

```
#(define-markup-command (smallcaps layout props str) (string?)
  "Print the string argument in small caps."
  (interpret-markup layout props
   (make-line-markup
    (map (lambda (s)
          (if (= (string-length s) 0)
              s
              (markup #:large (string-upcase (substring s 0 1))
                      #:translate (cons -0.6 0)
                      #:tiny (string-upcase (substring s 1)))))
         (string-split str #\Space)))))
```

The `smallcaps` command first splits its string argument into tokens separated by spaces (`(string-split str #\Space)`); for each token, a markup is built with the first letter made large and upcased (`#:large (string-upcase (substring s 0 1))`), and a second markup built with the following letters made tiny and upcased (`#:tiny (string-upcase (substring s 1))`). As LilyPond introduces a space between markups on a line, the second markup is translated to the left (`#:translate (cons -0.6 0) ...`). Then, the markups built for each token are put in

a line by `(make-line-markup ...)`. Finally, the resulting markup is passed to the `interpret-markup` function, with the `layout` and `props` arguments.

Note: there is now an internal command `\smallCaps` which can be used to set text in small caps. See Section B.6 [Overview of text markup commands], page 306, for details.

7.4.4 New markup list command definition

Markup list commands are defined with the `define-markup-list-command` Scheme macro, which is similar to the `define-markup-command` macro described in Section 7.4.3 [New markup command definition], page 280, except that where the later returns a single stencil, the former returns a list stencils.

In the following example, a `\paragraph` markup list command is defined, which returns a list of justified lines, the first one being indented. The indent width is taken from the `props` argument.

```
#(define-markup-list-command (paragraph layout props args) (markup-list?)
   (let ((indent (chain-assoc-get 'par-indent props 2)))
     (interpret-markup-list layout props
       (make-justified-lines-markup-list (cons (make-hspace-markup indent)
                                               args)))))
```

Besides the usual `layout` and `props` arguments, the `paragraph` markup list command takes a markup list argument, named `args`. The predicate for markup lists is `markup-list?`.

First, the function gets the indent width, a property here named `par-indent`, from the property list `props` If the property is not found, the default value is `2`. Then, a list of justified lines is made using the `make-justified-lines-markup-list` function, which is related to the `\justified-lines` built-in markup list command. An horizontal space is added at the beginning using the `make-hspace-markup` function. Finally, the markup list is interpreted using the `interpret-markup-list` function.

This new markup list command can be used as follows:

```
\markuplines {
  \paragraph {
    The art of music typography is called \italic {(plate) engraving.}
    The term derives from the traditional process of music printing.
    Just a few decades ago, sheet music was made by cutting and stamping
    the music into a zinc or pewter plate in mirror image.
  }
  \override-lines #'(par-indent . 4) \paragraph {
    The plate would be inked, the depressions caused by the cutting
    and stamping would hold ink.  An image was formed by pressing paper
    to the plate.  The stamping and cutting was completely done by
    hand.
  }
}
```

7.5 Contexts for programmers

7.5.1 Context evaluation

Contexts can be modified during interpretation with Scheme code. The syntax for this is

```
\applyContext function
```

function should be a Scheme function taking a single argument, being the context to apply it to. The following code will print the current bar number on the standard output during the compile:

```
\applyContext
  #(lambda (x)
    (format #t "\nWe were called in barnumber ~a.\n"
     (ly:context-property x 'currentBarNumber)))
```

7.5.2 Running a function on all layout objects

The most versatile way of tuning an object is `\applyOutput`. Its syntax is

```
\applyOutput context proc
```

where *proc* is a Scheme function, taking three arguments.

When interpreted, the function *proc* is called for every layout object found in the context *context*, with the following arguments:

- the layout object itself,
- the context where the layout object was created, and
- the context where `\applyOutput` is processed.

In addition, the cause of the layout object, i.e., the music expression or object that was responsible for creating it, is in the object property `cause`. For example, for a note head, this is a `NoteHead` event, and for a `Stem` object, this is a `NoteHead` object.

Here is a function to use for `\applyOutput`; it blanks note-heads on the center-line:

```
(define (blanker grob grob-origin context)
 (if (and (memq (ly:grob-property grob 'interfaces)
                note-head-interface)
          (eq? (ly:grob-property grob 'staff-position) 0))
     (set! (ly:grob-property grob 'transparent) #t)))
```

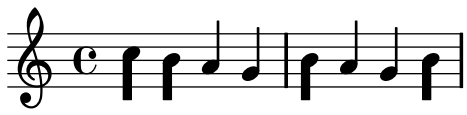
7.6 Scheme procedures as properties

Properties (like thickness, direction, etc.) can be set at fixed values with \override, e.g.

```
\override Stem #'thickness = #2.0
```

Properties can also be set to a Scheme procedure,

```
\override Stem #'thickness = #(lambda (grob)
    (if (= UP (ly:grob-property grob 'direction))
        2.0
        7.0))
c b a g b a g b
```



In this case, the procedure is executed as soon as the value of the property is requested during the formatting process.

Most of the typesetting engine is driven by such callbacks. Properties that typically use callbacks include

`stencil` The printing routine, that constructs a drawing for the symbol

`X-offset` The routine that sets the horizontal position

`X-extent` The routine that computes the width of an object

The procedure always takes a single argument, being the grob.

If routines with multiple arguments must be called, the current grob can be inserted with a grob closure. Here is a setting from `AccidentalSuggestion`,

```
(X-offset .
  ,(ly:make-simple-closure
    `(,+
        ,(ly:make-simple-closure
           (list ly:self-alignment-interface::centered-on-x-parent))
      ,(ly:make-simple-closure
           (list ly:self-alignment-interface::x-aligned-on-self)))))
```

In this example, both `ly:self-alignment-interface::x-aligned-on-self` and `ly:self-alignment-interface::centered-on-x-parent` are called with the grob as argument. The results are added with the `+` function. To ensure that this addition is properly executed, the whole thing is enclosed in `ly:make-simple-closure`.

In fact, using a single procedure as property value is equivalent to

```
(ly:make-simple-closure (ly:make-simple-closure (list proc)))
```

The inner `ly:make-simple-closure` supplies the grob as argument to *proc*, the outer ensures that result of the function is returned, rather than the `simple-closure` object.

Appendix A Literature list

If you need to know more about music notation, here are some interesting titles to read.

Ignatzek 1995

Klaus Ignatzek, Die Jazzmethode für Klavier. Schott's Söhne 1995. Mainz, Germany ISBN 3-7957-5140-3.

A tutorial introduction to playing Jazz on the piano. One of the first chapters contains an overview of chords in common use for Jazz music.

Gerou 1996

Tom Gerou and Linda Lusk, Essential Dictionary of Music Notation. Alfred Publishing, Van Nuys CA ISBN 0-88284-768-6.

A concise, alphabetically ordered list of typesetting and music (notation) issues, covering most of the normal cases.

Read 1968

Gardner Read, Music Notation: A Manual of Modern Practice. Taplinger Publishing, New York (2nd edition).

A standard work on music notation.

Ross 1987 Ted Ross, Teach yourself the art of music engraving and processing. Hansen House, Miami, Florida 1987.

This book is about music engraving, i.e., professional typesetting. It contains directions on stamping, use of pens and notational conventions. The sections on reproduction technicalities and history are also interesting.

Schirmer 2001

The G.Schirmer/AMP Manual of Style and Usage. G.Schirmer/AMP, NY, 2001. (This book can be ordered from the rental department.)

This manual specifically focuses on preparing print for publication by Schirmer. It discusses many details that are not in other, normal notation books. It also gives a good idea of what is necessary to bring printouts to publication quality.

Stone 1980

Kurt Stone, Music Notation in the Twentieth Century. Norton, New York 1980.

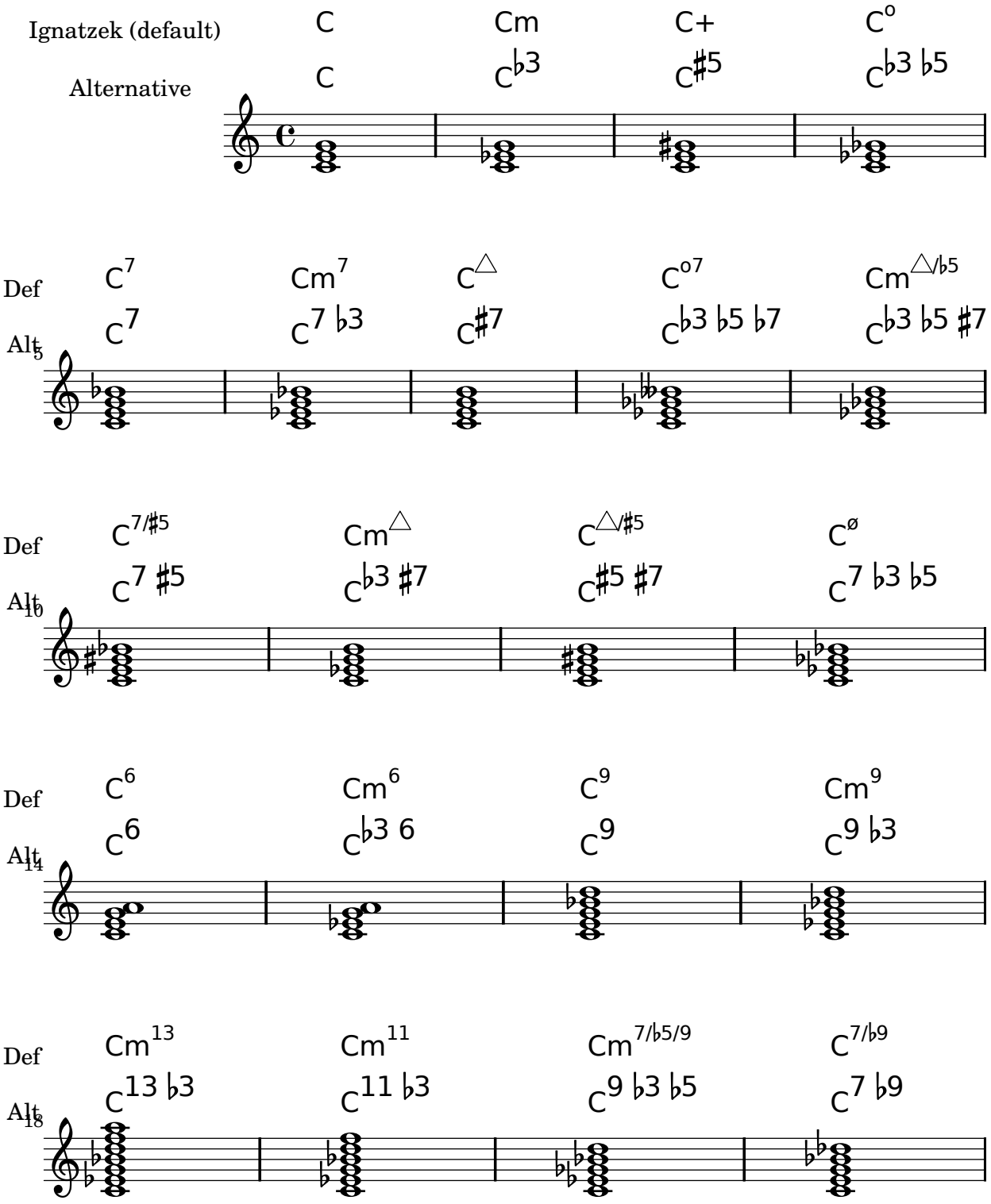
This book describes music notation for modern serious music, but starts out with a thorough overview of existing traditional notation practices.

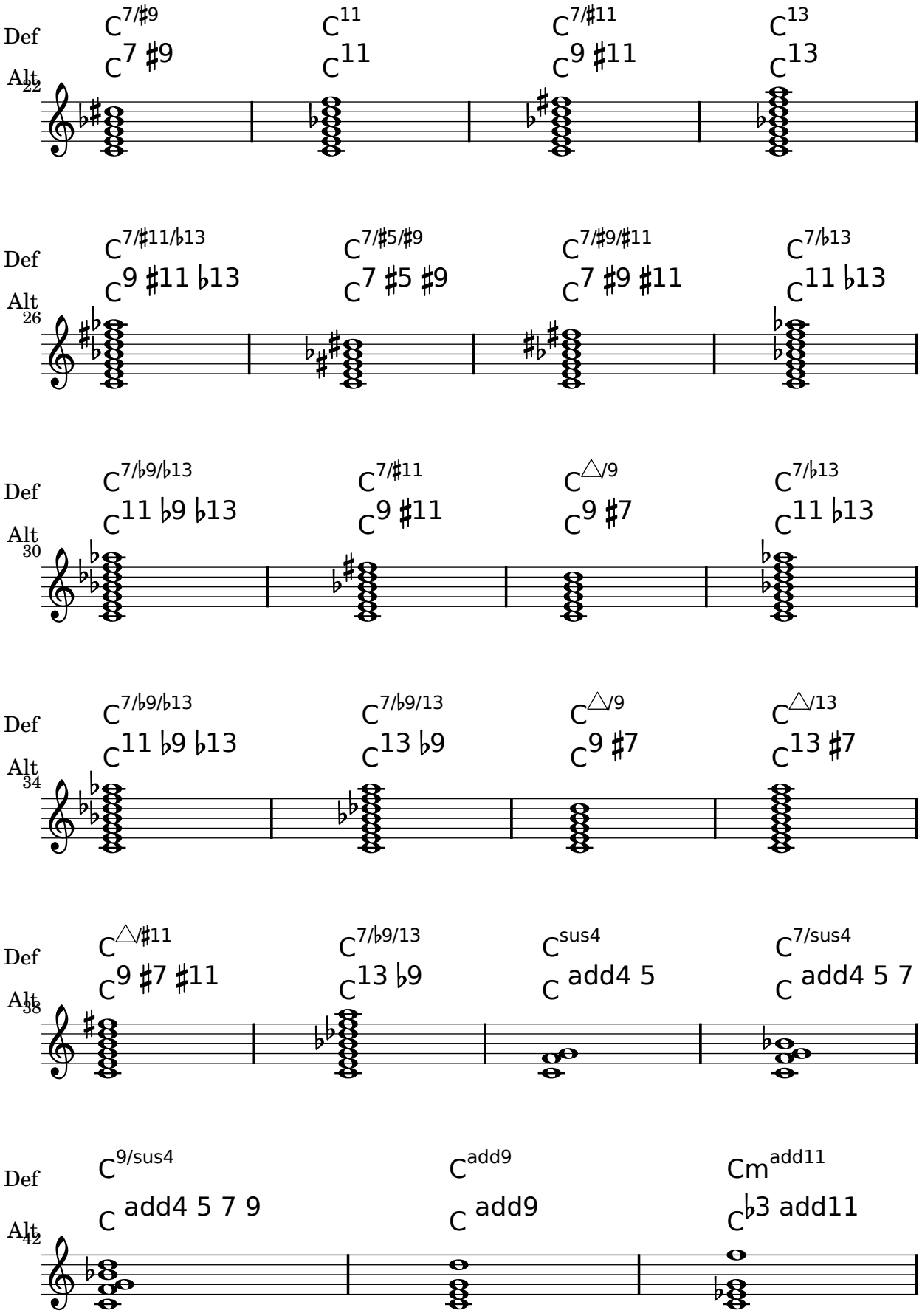
The source archive includes a more elaborate BibTEX bibliography of over 100 entries in '`Documentation/bibliography/`'.

Appendix B Notation manual tables

B.1 Chord name chart

The following charts shows two standard systems for printing chord names, along with the pitches they represent.





B.2 MIDI instruments

The following is a list of names that can be used for the `midiInstrument` property.

acoustic grand	contrabass	lead 7 (fifths)
bright acoustic	tremolo strings	lead 8 (bass+lead)
electric grand	pizzicato strings	pad 1 (new age)
honky-tonk	orchestral strings	pad 2 (warm)
electric piano 1	timpani	pad 3 (polysynth)
electric piano 2	string ensemble 1	pad 4 (choir)
harpsichord	string ensemble 2	pad 5 (bowed)
clav	synthstrings 1	pad 6 (metallic)
celesta	synthstrings 2	pad 7 (halo)
glockenspiel	choir aahs	pad 8 (sweep)
music box	voice oohs	fx 1 (rain)
vibraphone	synth voice	fx 2 (soundtrack)
marimba	orchestra hit	fx 3 (crystal)
xylophone	trumpet	fx 4 (atmosphere)
tubular bells	trombone	fx 5 (brightness)
dulcimer	tuba	fx 6 (goblins)
drawbar organ	muted trumpet	fx 7 (echoes)
percussive organ	french horn	fx 8 (sci-fi)
rock organ	brass section	sitar
church organ	synthbrass 1	banjo
reed organ	synthbrass 2	shamisen
accordion	soprano sax	koto
harmonica	alto sax	kalimba
concertina	tenor sax	bagpipe
acoustic guitar (nylon)	baritone sax	fiddle
acoustic guitar (steel)	oboe	shanai
electric guitar (jazz)	english horn	tinkle bell
electric guitar (clean)	bassoon	agogo
electric guitar (muted)	clarinet	steel drums
overdriven guitar	piccolo	woodblock
distorted guitar	flute	taiko drum
guitar harmonics	recorder	melodic tom
acoustic bass	pan flute	synth drum
electric bass (finger)	blown bottle	reverse cymbal
electric bass (pick)	shakuhachi	guitar fret noise
fretless bass	whistle	breath noise
slap bass 1	ocarina	seashore
slap bass 2	lead 1 (square)	bird tweet
synth bass 1	lead 2 (sawtooth)	telephone ring
synth bass 2	lead 3 (calliope)	helicopter
violin	lead 4 (chiff)	applause
viola	lead 5 (charang)	gunshot
cello	lead 6 (voice)	

B.3 List of colors

Normal colors

Usage syntax is detailed in Section 1.7.1.4 [Coloring objects], page 107.

black	white	red	green
blue	cyan	magenta	yellow
grey	darkred	darkgreen	darkblue

```
darkcyan    darkmagenta    darkyellow
```

X color names

X color names come several variants:

Any name that is spelled as a single word with capitalisation (e.g. 'LightSlateBlue') can also be spelled as space separated words without capitalisation (e.g. 'light slate blue').

The word 'grey' can always be spelled 'gray' (e.g. 'DarkSlateGray').

Some names can take a numerical suffix (e.g. 'LightSalmon4').

Color Names without a numerical suffix:

```
snow            GhostWhite      WhiteSmoke      gainsboro       FloralWhite
OldLace         linen           AntiqueWhite    PapayaWhip      BlanchedAlmond
bisque          PeachPuff       NavajoWhite     moccasin        cornsilk
ivory           LemonChiffon    seashell        honeydew        MintCream
azure           AliceBlue       lavender        LavenderBlush   MistyRose
white           black           DarkSlateGrey   DimGrey         SlateGrey
LightSlateGrey  grey            LightGrey       MidnightBlue    navy
NavyBlue        CornflowerBlue  DarkSlateBlue   SlateBlue       MediumSlateBlue
LightSlateBlue  MediumBlue      RoyalBlue       blue            DodgerBlue
DeepSkyBlue     SkyBlue         LightSkyBlue    SteelBlue       LightSteelBlue
LightBlue       PowderBlue      PaleTurquoise   DarkTurquoise   MediumTurquoise
turquoise       cyan            LightCyan       CadetBlue       MediumAquamarine
aquamarine      DarkGreen       DarkOliveGreen  DarkSeaGreen    SeaGreen
MediumSeaGreen  LightSeaGreen   PaleGreen       SpringGreen     LawnGreen
green           chartreuse      MediumSpringGreen       GreenYellow     LimeGreen
YellowGreen     ForestGreen     OliveDrab       DarkKhaki       khaki
PaleGoldenrod   LightGoldenrodYellow    LightYellow     yellow  gold
LightGoldenrod  goldenrod       DarkGoldenrod   RosyBrown       IndianRed
SaddleBrown     sienna          peru            burlywood       beige
wheat           SandyBrown      tan             chocolate       firebrick
brown           DarkSalmon      salmon          LightSalmon     orange
DarkOrange      coral           LightCoral      tomato          OrangeRed
red             HotPink         DeepPink        pink            LightPink
PaleVioletRed   maroon          MediumVioletRed VioletRed       magenta
violet          plum            orchid          MediumOrchid    DarkOrchid
DarkViolet      BlueViolet      purple          MediumPurple    thistle
DarkGrey        DarkBlue        DarkCyan        DarkMagenta     DarkRed
LightGreen
```

Color names with a numerical suffix

In the following names the suffix N can be a number in the range 1-4:

```
snowN           seashellN       AntiqueWhiteN   bisqueN         PeachPuffN
NavajoWhiteN    LemonChiffonN   cornsilkN       ivoryN          honeydewN
LavenderBlushN  MistyRoseN      azureN          SlateBlueN      RoyalBlueN
blueN           DodgerBlueN     SteelBlueN      DeepSkyBlueN    SkyBlueN
LightSkyBlueN   LightSteelBlueN LightBlueN      LightCyanN      PaleTurquoiseN
CadetBlueN      turquoiseN      cyanN           aquamarineN     DarkSeaGreenN
SeaGreenN       PaleGreenN      SpringGreenN    greenN          chartreuseN
OliveDrabN      DarkOliveGreenN khakiN          LightGoldenrodN LightYellowN
yellowN         goldN           goldenrodN      DarkGoldenrodN  RosyBrownN
IndianRedN      siennaN         burlywoodN      wheatN          tanN
```

chocolateN	firebrickN	brownN	salmonN	LightSalmonN
orangeN	DarkOrangeN	coralN	tomatoN	OrangeRedN
redN	DeepPinkN	HotPinkN	pinkN	LightPinkN
PaleVioletRedN	maroonN	VioletRedN	magentaN	orchidN
plumN	MediumOrchidN	DarkOrchidN	purpleN	MediumPurpleN
thistleN				

Grey Scale

A grey scale can be obtained using:

```
greyN
```

Where N is in the range 0-100.

B.4 The Feta font

The following symbols are available in the Emmentaler font and may be accessed directly using text markup such as `g^\markup { \musicglyph #"scripts.segno" }`, see Section 1.8.2 [Text markup], page 119.

space		plus	+
comma	,	hyphen	-
period	.	zero	0
one	1	two	2
three	3	four	4
five	5	six	6
seven	7	eight	8
nine	9	f	*f*

m		p	
r		s	
z		rests.0	
rests.1		rests.0o	
rests.1o		rests.M3	
rests.M2		rests.M1	
rests.2		rests.2classical	
rests.3		rests.4	
rests.5		rests.6	
rests.7		accidentals.sharp	
accidentals.sharp .slashslash.stem		accidentals.sharp .slashslashslash.stemstem	
accidentals.sharp .slashslashslash.stem		accidentals.sharp .slashslash.stemstemstem	
accidentals.natural		accidentals.flat	

accidentals.flat.slash 

accidentals.flat.slashslash 

accidentals.mirroredflat.flat 

accidentals.mirroredflat 

accidentals.mirroredflat.backslash 

accidentals.flatflat 

accidentals.flatflat.slash 

accidentals.doublesharp

accidentals.rightparen

accidentals.leftparen

arrowheads.open.01

arrowheads.open.0M1

arrowheads.open.11 

arrowheads.open.1M1 

arrowheads.close.01

arrowheads.close.0M1

arrowheads.close.11

arrowheads.close.1M1

dots.dot

noteheads.uM2 

noteheads.dM2 

noteheads.sM1 

noteheads.s0 

noteheads.s1 

noteheads.s2 

noteheads.s0diamond 

noteheads.s1diamond		noteheads.s2diamond	
noteheads.s0triangle		noteheads.d1triangle	
noteheads.u1triangle		noteheads.u2triangle	
noteheads.d2triangle		noteheads.s0slash	
noteheads.s1slash		noteheads.s2slash	
noteheads.s0cross		noteheads.s1cross	
noteheads.s2cross		noteheads.s2xcircle	
noteheads.s0do		noteheads.d1do	
noteheads.u1do		noteheads.d2do	
noteheads.u2do		noteheads.s0re	
noteheads.u1re		noteheads.d1re	
noteheads.u2re		noteheads.d2re	
noteheads.s0mi		noteheads.s1mi	

noteheads.s2mi		noteheads.u0fa	
noteheads.d0fa		noteheads.u1fa	
noteheads.d1fa		noteheads.u2fa	
noteheads.d2fa		noteheads.s0la	
noteheads.s1la		noteheads.s2la	
noteheads.s0ti		noteheads.u1ti	
noteheads.d1ti		noteheads.u2ti	
noteheads.d2ti		scripts.ufermata	
scripts.dfermata		scripts.ushortfermata	
scripts.dshortfermata		scripts.ulongfermata	
scripts.dlongfermata		scripts.uverylongfermata	
scripts.dverylongfermata		scripts.thumb	
scripts.sforzato		scripts.espr	

scripts.staccato		scripts.ustaccatissimo	
scripts.dstaccatissimo		scripts.tenuto	
scripts.uportato		scripts.dportato	
scripts.umarcato		scripts.dmarcato	
scripts.open		scripts.stopped	
scripts.upbow		scripts.downbow	
scripts.reverseturn		scripts.turn	
scripts.trill		scripts.upedalheel	
scripts.dpedalheel		scripts.upedaltoe	
scripts.dpedaltoe		scripts.flageolet	
scripts.segno		scripts.coda	
scripts.varcoda		scripts.rcomma	
scripts.lcomma		scripts.rvarcomma	

scripts.lvarcomma 

scripts.arpeggio 

scripts.trill_element

scripts.arpeggio.arrow.M1 

scripts.arpeggio.arrow.1 

scripts.trilelement

scripts.prall

scripts.mordent 

scripts.prallprall

scripts.prallmordent 

scripts.upprall 

scripts.upmordent 

scripts.pralldown 

scripts.downprall 

scripts.downmordent 

scripts.prallup 

scripts.lineprall 

scripts.caesura.curved 

scripts.caesura.straight 

flags.u3 

flags.u4 

flags.u5 

flags.u6 

flags.d3 

flags.ugrace 

flags.dgrace 

flags.d4		flags.d5	
flags.d6		clefs.C	
clefs.C_change		clefs.F	
clefs.F_change		clefs.G	
clefs.G_change		clefs.percussion	
clefs.percussion_change		clefs.tab	
clefs.tab_change		timesig.C44	
timesig.C22		pedal.*	
pedal.M		pedal..	
pedal.P		pedal.d	
pedal.e		pedal.Ped	
brackettips.up		brackettips.down	
accordion.accDiscant		accordion.accDot	

accordion.accFreebase		accordion.accStdbase	
accordion.accBayanbase		accordion.accOldEE	
rests.M3neomensural		rests.M2neomensural	
rests.M1neomensural		rests.0neomensural	
rests.1neomensural		rests.2neomensural	
rests.3neomensural		rests.4neomensural	
rests.M3mensural		rests.M2mensural	
rests.M1mensural		rests.0mensural	
rests.1mensural		rests.2mensural	
rests.3mensural		rests.4mensural	
noteheads.slneomensural		noteheads.sM3neomensural	
noteheads.sM2neomensural		noteheads.sM1neomensural	
noteheads.s0harmonic		noteheads.s2harmonic	

noteheads.s0neomensural 

noteheads.s1neomensural 

noteheads.s2neomensural 

noteheads.slmensural 

noteheads.sM3mensural 

noteheads.sM2mensural 

noteheads.sM1mensural 

noteheads.s0mensural

noteheads.s1mensural

noteheads.s2mensural

noteheads.s0petrucci 

noteheads.s1petrucci 

noteheads.s2petrucci 

noteheads
.svaticana.punctum

noteheads.svaticana
.punctum.cavum

noteheads.svaticana
.linea.punctum

noteheads.svaticana
.linea.punctum.cavum

noteheads.svaticana
.inclinatum

noteheads.svaticana.lpes

noteheads
.svaticana.vlpes

noteheads.svaticana.upes

noteheads
.svaticana.vupes

noteheads
.svaticana.plica

noteheads
.svaticana.vplica

noteheads
.svaticana.epiphonus

noteheads.svaticana
.vepiphonus

noteheads.svaticana.reverse.plica

noteheads.svaticana.reverse.vplica

noteheads.svaticana.inner.cephalicus

noteheads.svaticana.cephalicus

noteheads.svaticana.quilisma

noteheads.ssolesmes.incl.parvum

noteheads.ssolesmes.auct.asc

noteheads.ssolesmes.auct.desc

noteheads.ssolesmes.incl.auctum

noteheads.ssolesmes.stropha

noteheads.ssolesmes.stropha.aucta

noteheads.ssolesmes.oriscus

noteheads.smedicaea.inclinatum

noteheads.smedicaea.punctum

noteheads.smedicaea.rvirga

noteheads.smedicaea.virga

noteheads.shufnagel.punctum

noteheads.shufnagel.virga

noteheads.shufnagel.lpes

clefs.vaticana.do

clefs.vaticana.do_change

clefs.vaticana.fa

clefs.vaticana.fa_change

clefs.medicaea.do

clefs.medicaea.do_change

clefs.medicaea.fa

clefs.medicaea.fa_change 

clefs.neomensural.c 

clefs.neomensural .c_change 

clefs.petrucci.c1 

clefs.petrucci.c1_change 

clefs.petrucci.c2 

clefs.petrucci.c2_change 

clefs.petrucci.c3 

clefs.petrucci.c3_change 

clefs.petrucci.c4 

clefs.petrucci.c4_change 

clefs.petrucci.c5 

clefs.petrucci.c5_change 

clefs.mensural.c 

clefs.mensural.c_change 

clefs.petrucci.f 

clefs.petrucci.f_change 

clefs.mensural.f 

clefs.mensural.f_change

clefs.petrucci.g

clefs.petrucci.g_change 

clefs.mensural.g 

clefs.mensural.g_change 

clefs.hufnagel.do 

clefs.hufnagel.do_change 

clefs.hufnagel.fa 

clefs.hufnagel.fa_change 

clefs.hufnagel.do.fa 

clefs.hufnagel
.do.fa_change 

custodes.hufnagel.u0 

custodes.hufnagel.u1 

custodes.hufnagel.u2 

custodes.hufnagel.d0 

custodes.hufnagel.d1 

custodes.hufnagel.d2 

custodes.medicaea.u0

custodes.medicaea.u1

custodes.medicaea.u2

custodes.medicaea.d0

custodes.medicaea.d1

custodes.medicaea.d2

custodes.vaticana.u0

custodes.vaticana.u1

custodes.vaticana.u2

custodes.vaticana.d0

custodes.vaticana.d1

custodes.vaticana.d2

custodes.mensural.u0 

custodes.mensural.u1 

custodes.mensural.u2 

custodes.mensural.d0		custodes.mensural.d1	
custodes.mensural.d2		accidentals.medicaeaM1	
accidentals.vaticanaM1		accidentals.vaticana0	
accidentals.mensural1		accidentals.mensuralM1	
accidentals.hufnagelM1		flags.mensuralu03	
flags.mensuralu13		flags.mensuralu23	
flags.mensurald03		flags.mensurald13	
flags.mensurald23		flags.mensuralu04	
flags.mensuralu14		flags.mensuralu24	
flags.mensurald04		flags.mensurald14	
flags.mensurald24		flags.mensuralu05	
flags.mensuralu15		flags.mensuralu25	
flags.mensurald05		flags.mensurald15	

flags.mensurald25

flags.mensuralu06 

flags.mensuralu16

flags.mensuralu26

flags.mensurald06 

flags.mensurald16

flags.mensurald26

timesig.mensural44 

timesig.mensural22 

timesig.mensural32 

timesig.mensural64 

timesig.mensural94 

timesig.mensural34 

timesig.mensural68 

timesig.mensural98 

timesig.mensural48 

timesig.mensural68alt 

timesig.mensural24 

timesig.neomensural44 

timesig.neomensural22 

timesig.neomensural32 

timesig.neomensural64 

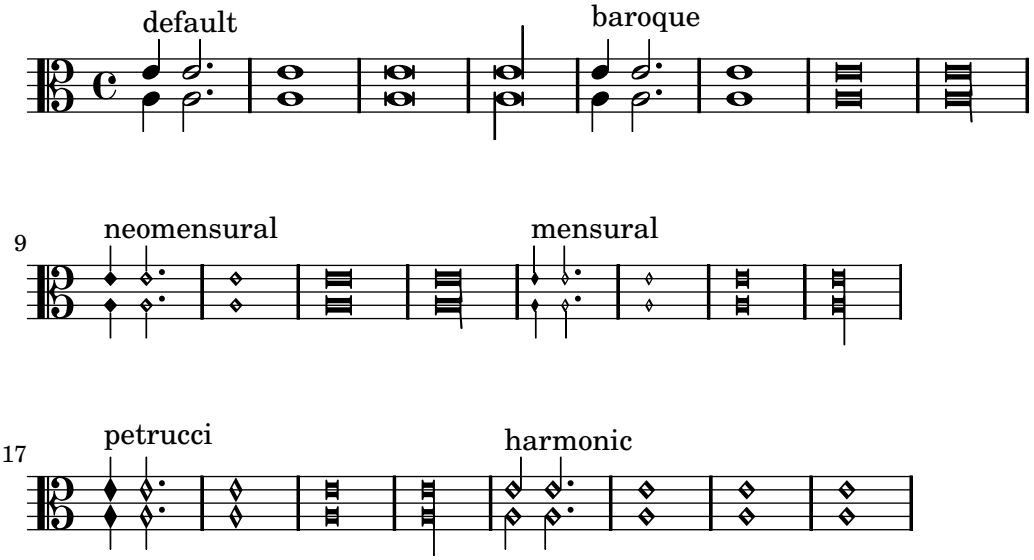
timesig.neomensural94 

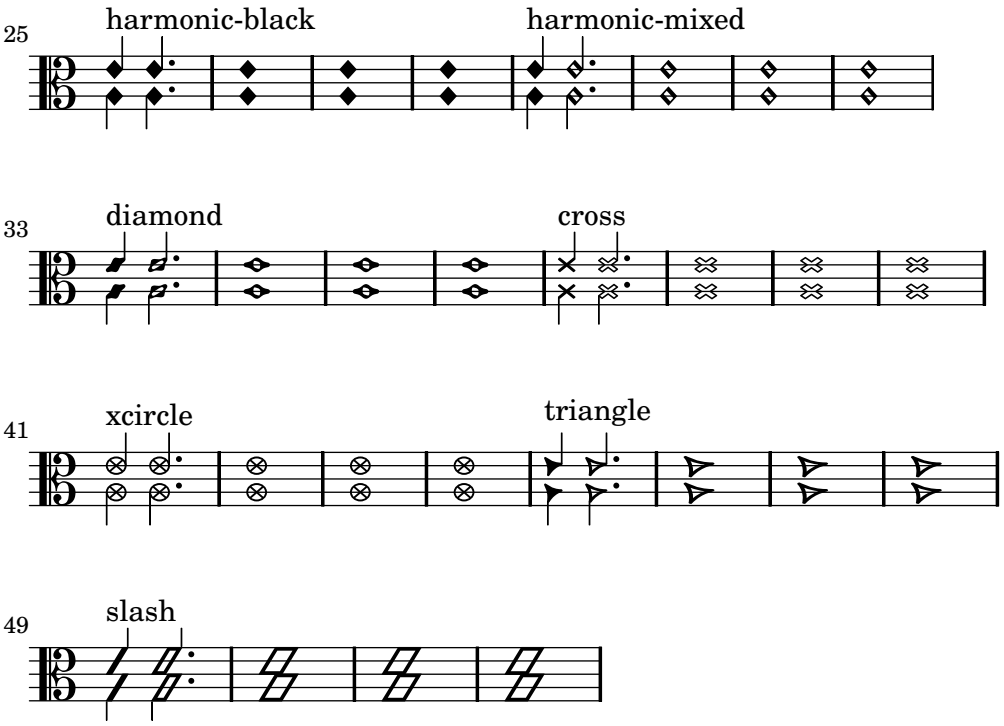
timesig.neomensural34 

timesig.neomensural68	𝁢	timesig.neomensural98	𝁤
timesig.neomensural48	𝁣	timesig.neomensural68alt	𝁥
timesig.neomensural24	𝁡	scripts.ictus	𝁩
scripts.uaccentus	𝁪	scripts.daccentus	𝁫
scripts.usemicirculus	𝁬	scripts.dsemicirculus	𝁭
scripts.circulus	𝁮	scripts.augmentum	𝁯
scripts .usignumcongruentiae	𝁰	scripts .dsignumcongruentiae	𝁱
dots.dotvaticana	𝁲		

B.5 Note head styles

The following styles may be used for note heads.





B.6 Overview of text markup commands

The following commands can all be used inside `\markup { }`.

`\arrow-head` *axis* (integer) *direction* (direction) *filled* (boolean)
: Produce an arrow head in specified direction and axis. Use the filled head if *filled* is specified.

`\beam` *width* (number) *slope* (number) *thickness* (number)
: Create a beam with the specified parameters.

`\bigger` *arg* (markup)
: Increase the font size relative to current setting.

`\bold` *arg* (markup)
: Switch to bold font-series.

`\box` *arg* (markup)
: Draw a box round *arg*. Looks at `thickness`, `box-padding` and `font-size` properties to determine line thickness and padding around the markup.

`\bracket` *arg* (markup)
: Draw vertical brackets around *arg*.

`\caps` *arg* (markup)
: Emit *arg* as small caps.

`\center-align` *args* (list of markups)
: Put `args` in a centered column.

`\char` *num* (integer)
: Produce a single character. For example, `\char #65` produces the letter 'A'.

`\circle` *arg* (markup)
: Draw a circle around *arg*. Use `thickness`, `circle-padding` and `font-size` properties to determine line thickness and padding around the markup.

`\column` *args* (list of markups)
: Stack the markups in *args* vertically. The property `baseline-skip` determines the space between each markup in *args*.

`\combine` *m1* (markup) *m2* (markup)
: Print two markups on top of each other.

`\concat` *args* (list of markups)
: Concatenate *args* in a horizontal line, without spaces inbetween. Strings and simple markups are concatenated on the input level, allowing ligatures. For example, `\concat { "f" \simple #"i" }` is equivalent to `"fi"`.

`\dir-column` *args* (list of markups)
: Make a column of args, going up or down, depending on the setting of the `#'direction` layout property.

`\doubleflat`
: Draw a double flat symbol.

`\doublesharp`
: Draw a double sharp symbol.

`\draw-circle` *radius* (number) *thickness* (number) *fill* (boolean)
: A circle of radius *radius*, thickness *thickness* and optionally filled.

`\draw-line` *dest* (pair of numbers)
: A simple line. Uses the `thickness` property.

`\dynamic` *arg* (markup)
: Use the dynamic font. This font only contains **s**, **f**, **m**, **z**, **p**, and **r**. When producing phrases, like 'più **f**', the normal words (like 'più') should be done in a different font. The recommended font for this is bold and italic.

`\epsfile` *axis* (number) *size* (number) *file-name* (string)
: Inline an EPS image. The image is scaled along *axis* to *size*.

`\fill-line` *markups* (list of markups)
: Put *markups* in a horizontal line of width *line-width*. The markups are spaced or flushed to fill the entire line. If there are no arguments, return an empty stencil.

`\filled-box` *xext* (pair of numbers) *yext* (pair of numbers) *blot* (number)
: Draw a box with rounded corners of dimensions *xext* and *yext*. For example,

  ```
  \filled-box #'(-.3 . 1.8) #'(-.3 . 1.8) #0
  ```

  creates a box extending horizontally from -0.3 to 1.8 and vertically from -0.3 up to 1.8, with corners formed from a circle of diameter 0 (i.e. sharp corners).

`\finger` *arg* (markup)
: Set the argument as small numbers.

`\flat`
: Draw a flat symbol.

`\fontCaps` *arg* (markup)
: Set `font-shape` to `caps`.

`\fontsize` *increment* (number) *arg* (markup)
: Add *increment* to the font-size. Adjust baseline skip accordingly.

`\fraction` *arg1* (markup) *arg2* (markup)
: Make a fraction of two markups.

`\fret-diagram` *definition-string* (string)
: Make a (guitar) fret diagram. For example, say

```
\markup \fret-diagram #"s:0.75;6-x;5-x;4-o;3-2;2-3;1-2;"
```

for fret spacing 3/4 of staff space, D chord diagram

Syntax rules for *definition-string*:

– Diagram items are separated by semicolons.

– Possible items:

- `s:`*number* – Set the fret spacing of the diagram (in staff spaces). Default: 1.
- `t:`*number* – Set the line thickness (in staff spaces). Default: 0.05.
- `h:`*number* – Set the height of the diagram in frets. Default: 4.
- `w:`*number* – Set the width of the diagram in strings. Default: 6.
- `f:`*number* – Set fingering label type (0 = none, 1 = in circle on string, 2 = below string). Default: 0.
- `d:`*number* – Set radius of dot, in terms of fret spacing. Default: 0.25.
- `p:`*number* – Set the position of the dot in the fret space. 0.5 is centered; 1 is on lower fret bar, 0 is on upper fret bar. Default: 0.6.
- `c:`*string1*`-`*string2*`-`*fret* – Include a barre mark from *string1* to *string2* on *fret*.
- *string*`-`*fret* – Place a dot on *string* at *fret*. If *fret* is '`o`', *string* is identified as open. If *fret* is '`x`', *string* is identified as muted.
- *string*`-`*fret*`-`*fingering* – Place a dot on *string* at *fret*, and label with *fingering* as defined by the `f:` code.

– Note: There is no limit to the number of fret indications per string.

`\fret-diagram-terse` *definition-string* (string)

Make a fret diagram markup using terse string-based syntax.

Here an example

```
\markup \fret-diagram-terse #"x;x;o;2;3;2;"
```

for a D chord diagram.

Syntax rules for *definition-string*:

- Strings are terminated by semicolons; the number of semicolons is the number of strings in the diagram.
- Mute strings are indicated by '`x`'.
- Open strings are indicated by '`o`'.
- A number indicates a fret indication at that fret.
- If there are multiple fret indicators desired on a string, they should be separated by spaces.
- Fingerings are given by following the fret number with a `-`, followed by the finger indicator, e.g. '`3-2`' for playing the third fret with the second finger.
- Where a barre indicator is desired, follow the fret (or fingering) symbol with `-(` to start a barre and `-)` to end the barre.

`\fret-diagram-verbose` *marking-list* (list)

Make a fret diagram containing the symbols indicated in *marking-list*.

For example,

```
\markup \fret-diagram-verbose
  #'((mute 6) (mute 5) (open 4)
     (place-fret 3 2) (place-fret 2 3) (place-fret 1 2))
```

produces a standard D chord diagram without fingering indications.

Possible elements in *marking-list*:

`(mute` *`string-number`*`)`
: Place a small 'x' at the top of string *string-number*.

`(open` *`string-number`*`)`
: Place a small 'o' at the top of string *string-number*.

`(barre` *`start-string end-string fret-number`*`)`
: Place a barre indicator (much like a tie) from string *start-string* to string *end-string* at fret *fret-number*.

`(place-fret` *`string-number fret-number finger-value`*`)`
: Place a fret playing indication on string *string-number* at fret *fret-number* with an optional fingering label *finger-value*. By default, the fret playing indicator is a solid dot. This can be changed by setting the value of the variable *dot-color*. If the *finger* part of the `place-fret` element is present, *finger-value* will be displayed according to the setting of the variable *finger-code*. There is no limit to the number of fret indications per string.

`\fromproperty` *symbol* (symbol)
: Read the *symbol* from property settings, and produce a stencil from the markup contained within. If *symbol* is not defined, it returns an empty markup.

`\general-align` *axis* (integer) *dir* (number) *arg* (markup)
: Align *arg* in *axis* direction to the *dir* side.

`\halign` *dir* (number) *arg* (markup)
: Set horizontal alignment. If *dir* is `-1`, then it is left-aligned, while `+1` is right. Values inbetween interpolate alignment accordingly.

`\hbracket` *arg* (markup)
: Draw horizontal brackets around *arg*.

`\hcenter-in` *length* (number) *arg* (markup)
: Center *arg* horizontally within a box of extending *length*/2 to the left and right.

`\hcenter` *arg* (markup)
: Align `arg` to its X center.

`\hspace` *amount* (number)
: This produces a invisible object taking horizontal space. For example,

```
\markup { A \hspace #2.0 B }
```

puts extra space between A and B, on top of the space that is normally inserted before elements on a line.

`\huge` *arg* (markup)
: Set font size to +2.

`\italic` *arg* (markup)
: Use italic `font-shape` for *arg*.

`\justify-field` *symbol* (symbol)
: Justify the data which has been assigned to *symbol*.

`\justify` *args* (list of markups)
: Like wordwrap, but with lines stretched to justify the margins. Use `\override #'(line-width . X)` to set the line width; *X* is the number of staff spaces.

`\justify-string` *arg* (string)
: Justify a string. Paragraphs may be separated with double newlines

`\large` *arg* (markup)
: Set font size to +1.

`\larger` *arg* (markup)
: Copy of the bigger-markup command.

`\left-align` *arg* (markup)
: Align *arg* on its left edge.

`\line` *args* (list of markups)
: Put *args* in a horizontal line. The property `word-space` determines the space between each markup in *args*.

`\lookup` *glyph-name* (string)
: Lookup a glyph by name.

`\lower` *amount* (number) *arg* (markup)
: Lower *arg* by the distance *amount*. A negative *amount* indicates raising; see also `\raise`.

`\magnify` *sz* (number) *arg* (markup)
: Set the font magnification for its argument. In the following example, the middle A is 10% larger:

```
A \magnify #1.1 { A } A
```

Note: Magnification only works if a font name is explicitly selected. Use `\fontsize` otherwise.

`\markalphabet` *num* (integer)
: Make a markup letter for *num*. The letters start with A to Z and continue with double letters.

`\markletter` *num* (integer)
: Make a markup letter for *num*. The letters start with A to Z (skipping letter I), and continue with double letters.

`\medium` *arg* (markup)
: Switch to medium font series (in contrast to bold).

`\musicglyph` *glyph-name* (string)
: *glyph-name* is converted to a musical symbol; for example, `\musicglyph #"accidentals.natural"` selects the natural sign from the music font. See notation reference, Section B.4 [The Feta font], page 290 for a complete listing of the possible glyphs.

`\natural`
: Draw a natural symbol.

`\normal-size-sub` *arg* (markup)
: Set *arg* in subscript, in a normal font size.

`\normal-size-super` *arg* (markup)
: Set *arg* in superscript with a normal font size.

`\normal-text` *arg* (markup)
: Set all font related properties (except the size) to get the default normal text font, no matter what font was used earlier.

`\normalsize` *arg* (markup)
: Set font size to default.

`\note-by-number` *log* (number) *dot-count* (number) *dir* (number)
: Construct a note symbol, with stem. By using fractional values for *dir*, you can obtain longer or shorter stems.

`\note` *duration* (string) *dir* (number)
: This produces a note with a stem pointing in *dir* direction, with the *duration* for the note head type and augmentation dots. For example, `\note #"4." #-0.75` creates a dotted quarter note, with a shortened down stem.

`\null`
: An empty markup with extents of a single point.

`\number` *arg* (markup)
: Set font family to `number`, which yields the font used for time signatures and fingerings. This font only contains numbers and some punctuation. It doesn't have any letters.

`\on-the-fly` *procedure* (symbol) *arg* (markup)
: Apply the *procedure* markup command to *arg*. *procedure* should take a single argument.

`\override` *new-prop* (pair) *arg* (markup)
: Add the first argument in to the property list. Properties may be any sort of property supported by `font-interface` and `text-interface`, for example

```
\override #'(font-family . married) "bla"
```

`\pad-around` *amount* (number) *arg* (markup)
: Add padding *amount* all around *arg*.

`\pad-markup` *padding* (number) *arg* (markup)
: Add space around a markup object.

`\pad-to-box` *x-ext* (pair of numbers) *y-ext* (pair of numbers) *arg* (markup)
: Make *arg* take at least *x-ext*, *y-ext* space.

`\pad-x` *amount* (number) *arg* (markup)
: Add padding *amount* around *arg* in the X direction.

`\page-ref` *label* (symbol) *gauge* (markup) *default* (markup)
: Reference to a page number. *label* is the label set on the referenced page (using the `\label` command), *gauge* a markup used to estimate the maximum width of the page number, and *default* the value to display when *label* is not found.

`\postscript` *str* (string)
: This inserts *str* directly into the output as a PostScript command string. Due to technicalities of the output backends, different scales should be used for the TEX and PostScript backend, selected with `-f`.

  For the TEX backend, the following string prints a rotated text

```
0 0 moveto /ecrm10 findfont
1.75 scalefont setfont 90 rotate (hello) show
```

  The magical constant 1.75 scales from LilyPond units (staff spaces) to TEX dimensions.

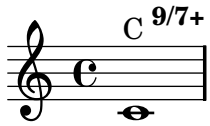
  For the postscript backend, use the following

```
gsave /ecrm10 findfont
 10.0 output-scale div
 scalefont setfont 90 rotate (hello) show grestore
```

`\put-adjacent` *arg1* (markup) *axis* (integer) *dir* (direction) *arg2* (markup)
Put *arg2* next to *arg1*, without moving *arg1*.

`\raise` *amount* (number) *arg* (markup)
Raise *arg* by the distance *amount*. A negative *amount* indicates lowering, see also `\lower`.

```
c1^\markup { C \small \raise #1.0 \bold { "9/7+" } }
```



The argument to `\raise` is the vertical displacement amount, measured in (global) staff spaces. `\raise` and `\super` raise objects in relation to their surrounding markups.

If the text object itself is positioned above or below the staff, then `\raise` cannot be used to move it, since the mechanism that positions it next to the staff cancels any shift made with `\raise`. For vertical positioning, use the `padding` and/or `extra-offset` properties.

`\right-align` *arg* (markup)
Align *arg* on its right edge.

`\roman` *arg* (markup)
Set font family to `roman`.

`\rotate` *ang* (number) *arg* (markup)
Rotate object with *ang* degrees around its center.

`\sans` *arg* (markup)
Switch to the sans serif family.

`\score` *score* (unknown)
Inline an image of music.

`\semiflat`
Draw a semiflat.

`\semisharp`
Draw a semi sharp symbol.

`\sesquiflat`
Draw a 3/2 flat symbol.

`\sesquisharp`
Draw a 3/2 sharp symbol.

`\sharp`
Draw a sharp symbol.

`\simple` *str* (string)
A simple text string; `\markup { foo }` is equivalent with `\markup { \simple #"foo" }`.

`\slashed-digit` *num* (integer)
A feta number, with slash. This is for use in the context of figured bass notation.

`\small` *arg* (markup)
Set font size to -1.

`\smallCaps` *text* (markup)
Turn `text`, which should be a string, to small caps.

```
\markup \smallCaps "Text between double quotes"
```

Note: `\smallCaps` does not support accented characters.

`\smaller` *arg* (markup)
Decrease the font size relative to current setting.

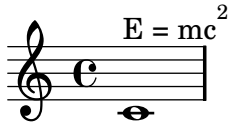
`\stencil` *stil* (unknown)
Use a stencil as markup.

`\strut`
Create a box of the same height as the space in the current font.

`\sub` *arg* (markup)
Set *arg* in subscript.

`\super` *arg* (markup)
Raising and lowering texts can be done with `\super` and `\sub`:

```
c1^\markup { E "=" \concat { "mc" \super "2" } }
```



`\teeny` *arg* (markup)
Set font size to -3.

`\text` *arg* (markup)
Use a text font instead of music symbol or music alphabet font.

`\tied-lyric` *str* (string)
Like simple-markup, but use tie characters for '~' tilde symbols.

`\tiny` *arg* (markup)
Set font size to -2.

`\translate` *offset* (pair of numbers) *arg* (markup)
This translates an object. Its first argument is a cons of numbers.

```
A \translate #(cons 2 -3) { B C } D
```

This moves 'B C' 2 spaces to the right, and 3 down, relative to its surroundings. This command cannot be used to move isolated scripts vertically, for the same reason that `\raise` cannot be used for that.

`\translate-scaled` *offset* (pair of numbers) *arg* (markup)
Translate *arg* by *offset*, scaling the offset by the `font-size`.

`\transparent` *arg* (markup)
Make the argument transparent.

`\triangle` *filled* (boolean)
A triangle, either filled or empty.

`\typewriter` *arg* (markup)
Use `font-family` typewriter for *arg*.

`\underline` *arg* (markup)
: Underline *arg*. Looks at `thickness` to determine line thickness and y offset.

`\upright` *arg* (markup)
: Set font shape to `upright`. This is the opposite of `italic`.

`\vcenter` *arg* (markup)
: Align `arg` to its Y center.

`\verbatim-file` *name* (string)
: Read the contents of a file, and include it verbatim.

`\whiteout` *arg* (markup)
: Provide a white underground for *arg*.

`\with-color` *color* (list) *arg* (markup)
: Draw *arg* in color specified by *color*.

`\with-dimensions` *x* (pair of numbers) *y* (pair of numbers) *arg* (markup)
: Set the dimensions of *arg* to *x* and *y*.

`\with-url` *url* (string) *arg* (markup)
: Add a link to URL *url* around *arg*. This only works in the PDF backend.

`\wordwrap-field` *symbol* (symbol)
: Wordwrap the data which has been assigned to *symbol*.

`\wordwrap` *args* (list of markups)
: Simple wordwrap. Use `\override #'(line-width . X)` to set the line width, where *X* is the number of staff spaces.

`\wordwrap-string` *arg* (string)
: Wordwrap a string. Paragraphs may be separated with double newlines.

B.7 Overview of text markup list commands

The following commands can all be used with `\markuplines`.

`\column-lines` *args* (list of markups)
: Like `\column`, but return a list of lines instead of a single markup. `baseline-skip` determines the space between each markup in *args*.

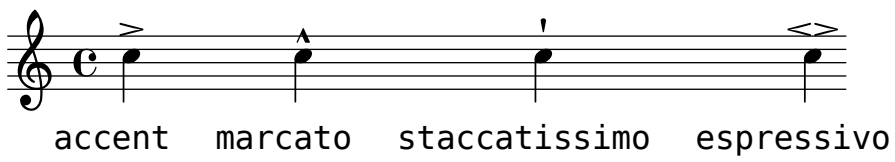
`\justified-lines` *args* (list of markups)
: Like `\justify`, but return a list of lines instead of a single markup. Use `\override-lines #'(line-width . X)` to set the line width; *X* is the number of staff spaces.

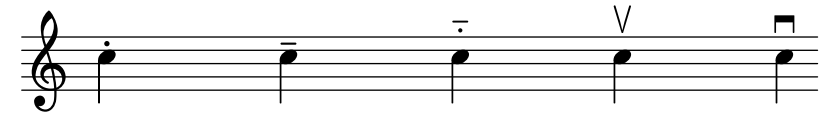
`\override-lines` *new-prop* (pair) *args* (list of markups)
: Like `\override`, for markup lists.

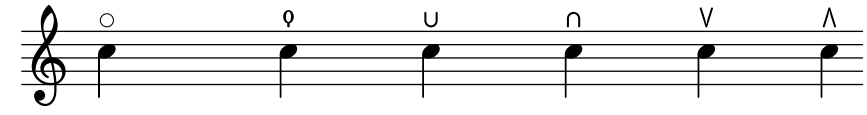
`\wordwrap-lines` *args* (list of markups)
: Like `\wordwrap`, but return a list of lines instead of a single markup. Use `\override-lines #'(line-width . X)` to set the line width, where *X* is the number of staff spaces.

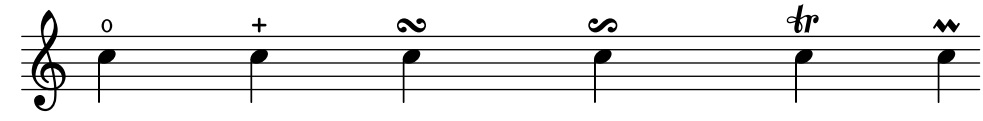
B.8 List of articulations

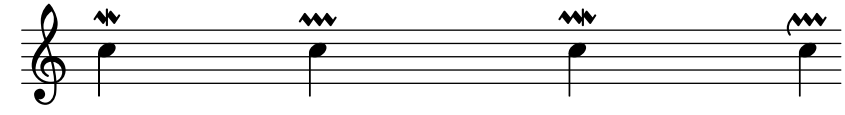
Here is a chart showing all scripts available,

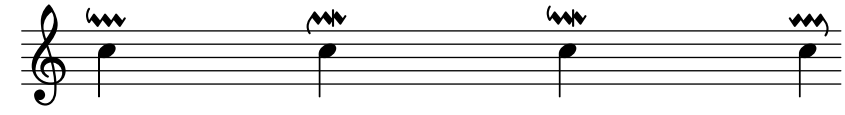


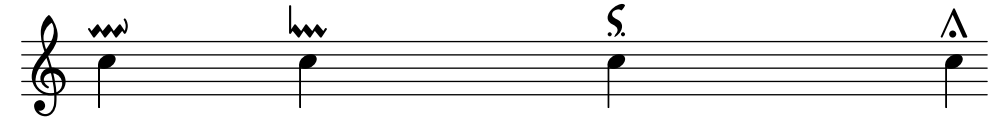

staccato tenuto portato upbow downbow

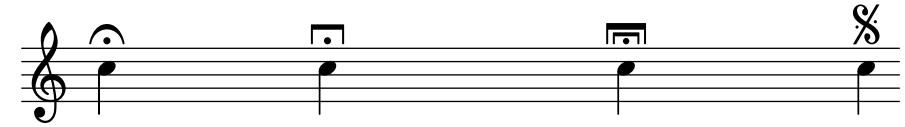

flageolet thumb lheel rheel ltoe rtoe

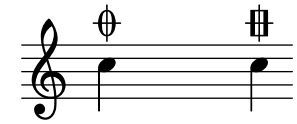

open stopped turn reverseturn trill prall


mordent prallprall prallmordent upprall


downprall upmordent downmordent pralldown


prallup lineprall signumcongruentiae shortfermata


fermata longfermata verylongfermata segno


coda varcoda

B.9 All context properties

`aDueText` (string)
: Text to print at a unisono passage.

`alignAboveContext` (string)
: Where to insert newly created context in vertiical alignment.

`alignBassFigureAccidentals` (boolean)
: If true, then the accidentals are aligned in bass figure context.

`alignBelowContext` (string)
: Where to insert newly created context in vertiical alignment.

`associatedVoice` (string)
: Name of the `Voice` that has the melody for this `Lyrics` line.

`autoAccidentals` (list)
: List of different ways to typeset an accidental.

  For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used. Each rule consists of

  context In which context is the rule applied. For example, if *context* is `Score` then all staves share accidentals, and if *context* is `Staff` then all voices in the same staff share accidentals, but staves do not.

  octavation Whether the accidental changes all octaves or only the current octave. Valid choices are

  `same-octave`
  : This is the default algorithm. Accidentals are typeset if the note changes the accidental of that note in that octave. Accidentals lasts to the end of the measure and then as many measures as specified in the value. This is, `1` means to the end of next measure, `-1` means to the end of previous measure (that is: no duration at all), etc. `#t` means forever.

  `any-octave`
  : Accidentals are typeset if the note is different from the previous note on the same pitch in any octave. The value has same meaning as in `same-octave`.

  laziness Over how many bar lines the accidental lasts. If *laziness* is `-1` then the accidental is forgotten immediately, and if *laziness* is `#t` then the accidental lasts forever.

`autoBeamCheck` (procedure)
: A procedure taking three arguments, *context*, *dir* [start/stop (-1 or 1)], and *test* [shortest note in the beam]. A non-`#f` return value starts or stops the auto beam.

`autoBeamSettings` (list)
: Specifies when automatically generated beams should begin and end. See notation reference, Section 1.2.4.2 [Setting automatic beam behavior], page 51 for more information.

`autoBeaming` (boolean)
: If set to true then beams are generated automatically.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`automaticBars` (boolean)
: If set to true then bar lines will not be printed automatically; they must be explicitly created with a `\bar` command. Unlike the `\cadenza` keyword, measures are still counted. Bar generation will resume according to that count if this property is unset.

`barAlways` (boolean)
: If set to true a bar line is drawn after each note.

`barCheckSynchronize` (boolean)
: If true then reset `measurePosition` when finding a bar check.

`barNumberVisibility` (procedure)
: A Procedure that takes an integer and returns whether the corresponding bar number should be printed.

`bassFigureFormatFunction` (procedure)
: A procedure that is called to produce the formatting for a `BassFigure` grob. It takes a list of `BassFigureEvent`s, a context, and the grob to format.

`bassStaffProperties` (list)
: An alist of property settings to apply for the down staff of `PianoStaff`. Used by `\autochange`.

`beatGrouping` (list)
: A list of beatgroups, e.g., in 5/8 time `'(2 3)`.

`beatLength` (moment)
: The length of one beat in this time signature.

`chordChanges` (boolean)
: Only show changes in chords scheme?

`chordNameExceptions` (list)
: An alist of chord exceptions. Contains `(`*`chord . markup`*`)` entries.

`chordNameExceptionsFull` (list)
: An alist of full chord exceptions. Contains `(`*`chord . markup`*`)` entries.

`chordNameExceptionsPartial` (list)
: An alist of partial chord exceptions. Contains `(`*`chord . (prefix-markup suffix-markup`*`))` entries.

`chordNameFunction` (procedure)
: The function that converts lists of pitches to chord names.

`chordNameSeparator` (markup)
: The markup object used to separate parts of a chord name.

`chordNoteNamer` (procedure)
: A function that converts from a pitch object to a text markup. Used for single pitches.

`chordPrefixSpacer` (number)
: The space added between the root symbol and the prefix of a chord name.

`chordRootNamer` (procedure)
: A function that converts from a pitch object to a text markup. Used for chords.

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefOctavation` (integer)
: Add this much extra octavation. Values of 7 and -7 are common.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`connectArpeggios` (boolean)
: If set, connect arpeggios across piano staff.

`countPercentRepeats` (boolean)
: If set, produce counters for percent repeats.

`createKeyOnClefChange` (boolean)
: Print a key signature whenever the clef is changed.

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`', '`line`', '`dashed-line`', '`dotted-line`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`decrescendoSpanner` (symbol)
: See `crescendoSpanner`.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

`defaultBarType` (string)
: Set the default type of bar line. See `whichBar` for information on available bar types.

  This variable is read by `Timing_translator` at `Score` level.

`doubleSlurs` (boolean)
: If set, two slurs are created for every slurred note, one above and one below the chord.

`drumPitchTable` (hash table)
: A table mapping percussion instruments (symbols) to pitches.

`drumStyleTable` (hash table)
: A hash table which maps drums to layout settings. Predefined values: '`drums-style`', '`timbales-style`', '`congas-style`', '`bongos-style`', and '`percussion-style`'.

  The layout style is a hash table, containing the drum-pitches (e.g., the symbol '`hihat`') as keys, and a list `(`*`notehead-style script vertical-position`*`)` as values.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`explicitKeySignatureVisibility` (vector)
: '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extendersOverRests` (boolean)
: Whether to continue extenders as they cross a rest.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals changing from a non-natural to another non-natural.

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`figuredBassPlusDirection` (direction)
: Where to put plus signs relative to the main figure.

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`firstClef` (boolean)
: If true, create a new clef when starting a staff.

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

`fontSize` (number)
: The relative size of all grobs in a context.

`forbidBreak` (boolean)
: If set to `##t`, prevent a line break at this point.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

`gridInterval` (moment)
: Interval for which to generate `GridPoints`.

`hairpinToBarline` (boolean)
: If set, end a hairpin at the barline before the ending note.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`highStringOne` (boolean)
: Whether the first string is the string with highest pitch on the instrument. This used by the automatic string selector for tablature notation.

`ignoreBarChecks` (boolean)
: Ignore bar checks.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`ignoreMelismata` (boolean)
: Ignore melismata for this `Lyrics` line.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`implicitTimeSignatureVisibility` (vector)
: break visibility for the default time signature.

`instrumentCueName` (markup)
: The name to print if another instrument is to be taken.

`instrumentEqualizer` (procedure)
: A function taking a string (instrument name), and returning a `(`*`min . max`*`)` pair of numbers for the loudness range of the instrument.

`instrumentName` (markup)
: The name to print left of a staff. The `instrument` property labels the staff in the first system, and the `instr` property labels following lines.

`instrumentTransposition` (pitch)
: Define the transposition of the instrument. Its value is the pitch that sounds like middle C. This is used to transpose the MIDI output, and `\quote`s.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

`keyAlterationOrder` (list)
: An alist that defines in what order alterations should be printed. The format is `(`*`step . alter`*`)`, where *step* is a number from 0 to 6 and *alter* from -2 (sharp) to 2 (flat).

`keySignature` (list)
: The current key signature. This is an alist containing `(`*`step . alter`*`)` or `((`*`octave . step`*`) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keySignature = #`((6 . ,FLAT))`.

`lyricMelismaAlignment` (direction)
: Alignment to use for a melisma syllable.

`majorSevenSymbol` (markup)
: How should the major 7th be formatted in a chord name?

`markFormatter` (procedure)
: A procedure taking as arguments the context and the rehearsal mark. It should return the formatted mark as a markup object.

`maximumFretStretch` (number)
: Don't allocate frets further than this from specified frets.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`melismaBusyProperties` (list)
: A list of properties (symbols) to determine whether a melisma is playing. Setting this property will influence how lyrics are aligned to notes. For example, if set to `#'(melismaBusy beamMelismaBusy)`, only manual melismata and manual beams are considered. Possible values include `melismaBusy`, `slurMelismaBusy`, `tieMelismaBusy`, and `beamMelismaBusy`.

`metronomeMarkFormatter` (procedure)
: How to produce a metronome markup. Called with two arguments, event and context.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`middleCOffset` (number)
: The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`midiInstrument` (string)
: Name of the MIDI instrument to use.

`midiMaximumVolume` (number)
: Analogous to `midiMinimumVolume`.

`midiMinimumVolume` (number)
: Set the minimum loudness for MIDI. Ranges from 0 to 1.

`minimumFret` (number)
: The tablature auto string-selecting mechanism selects the highest string with a fret at least `minimumFret`.

`minimumPageTurnLength` (moment)
: Minimum length of a rest for a page turn to be allowed.

`minimumRepeatLengthForPageTurn` (moment)
: Minimum length of a repeated section for a page turn to be allowed within that section.

`noteToFretFunction` (procedure)
: How to produce a fret diagram. Parameters: A list of note events and a list of tabstring events.

`ottavation` (string)
: If set, the text for an ottava spanner. Changing this creates a new text spanner.

`output` (unknown)
: The output produced by a score-level translator during music interpretation.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

`printOctaveNames` (boolean)
: Print octave marks for the `NoteNames` context.

`printPartCombineTexts` (boolean)
: Set 'Solo' and 'A due' texts in the part combiner?

`proportionalNotationDuration` (moment)
: Global override for shortest-playing duration. This is used for switching on proportional notation.

`recordEventSequence` (procedure)
: When `Recording_group_engraver` is in this context, then upon termination of the context, this function is called with current context and a list of music objects. The list of contains entries with start times, music objects and whether they are processed in this context.

`rehearsalMark` (integer)
: The last rehearsal mark printed.

`repeatCommands` (list)
: This property is read to find any command of the form `(volta . x)`, where x is a string or `#f`.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

`shapeNoteStyles` (vector)
: Vector of symbols, listing style for each note head relative to the tonic (qv.) of the scale.

`shortInstrumentName` (markup)
: See `instrument`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`skipBars` (boolean)
: If set to true, then skip the empty bars that are produced by multimeasure notes and rests. These bars will not appear on the printed output. If not set (the default), multimeasure notes and rests expand into their full length, printing the appropriate number of empty bars so that synchronization with other voices is preserved.

```
{
  r1 r1*3 R1*3
  \set Score.skipBars= ##t
  r1*3 R1*3
}
```

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`soloIIText` (string)
: The text for the start of a solo for voice ‘two’ when part-combining.

`soloText` (string)
: The text for the start of a solo when part-combining.

`squashedPosition` (integer)
: Vertical position of squashing for `Pitch_squash_engraver`.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

`stanza` (markup)
: Stanza ‘number’ to print before the start of a verse. Use in `Lyrics` context.

`stemLeftBeamCount` (integer)
: Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
: See `stemLeftBeamCount`.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`stringOneTopmost` (boolean)
: Whether the first string is printed on the top line of the tablature.

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitch (in semitones) of each string (starting with the lower one).

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at beat positions by only drawing one beam over the beat.

`suggestAccidentals` (boolean)
: If set, accidentals are typeset as cautionary suggestions over the note.

`systemStartDelimiter` (symbol)
: Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

`systemStartDelimiterHierarchy` (pair)
: A nested list, indicating the nesting of a start delimiters.

`tablatureFormat` (procedure)
: A function formatting a tablature note head; it takes a string number, a list of string tunings and a `Pitch` object. It returns the text as a string.

`tempoUnitCount` (number)
: Count for specifying tempo.

`tempoUnitDuration` (duration)
: Unit for specifying tempo.

`tempoWholesPerMinute` (moment)
: The tempo in whole notes per minute.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

`timeSignatureFraction` (pair of numbers)
: A pair of numbers, signifying the time signature. For example, `#'(4 . 4)` is a 4/4 time signature.

`timing` (boolean)
: Keep administration of measure length, position, bar number, etc.? Switch off for cadenzas.

`tonic` (pitch)
: The tonic of the current scale.

`trebleStaffProperties` (list)
: An alist of property settings to apply for the up staff of `PianoStaff`. Used by `\autochange`.

`tremoloFlags` (integer)
: The number of tremolo flags to add if no number is specified.

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

`tupletSpannerDuration` (moment)
: Normally, a tuplet bracket is as wide as the `\times` expression that gave rise to it. By setting this property, you can make brackets last shorter.

```
{
  \set tupletSpannerDuration = #(ly:make-moment 1 4)
  \times 2/3 { c8 c c c c c }
}
```

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

`verticallySpacedContexts` (list)
: List of symbols, containing context names whose vertical axis groups should be taken into account for vertical spacing of systems.

`vocalName` (markup)
: Name of a vocal line.

`voltaSpannerDuration` (moment)
: This specifies the maximum duration to use for the brackets printed for `\alternative`. This can be used to shrink the length of brackets in the situation where one alternative is very large.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

```
\set Staff.whichBar = "|:"
```

  This will create a start-repeat bar in this staff only. Valid values are described in `bar-line-interface`.

B.10 Layout properties

`X-extent` (pair of numbers)
: Hard coded extent in X direction.

`X-offset` (number)
: The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers)
: See `X-extent`.

`Y-offset` (number)
: The vertical amount that this object is moved relative to its Y-parent.

`add-stem-support` (boolean)
: If set, the `Stem` object is included in this script's support.

`after-line-breaking` (boolean)
: Dummy property, used to trigger callback for `after-line-breaking`.

`align-dir` (direction)
: Which side to align? `-1`: left side, `0`: around center of width, `1`: right side.

`allow-loose-spacing` (boolean)
: If set, column can be detached from main spacing.

`allow-span-bar` (boolean)
: If false, no inter-staff barline will be created below this barline.

`alteration` (number)
: Alteration numbers for accidental.

`alteration-alist` (list)
: List of (*`pitch . accidental`*) pairs for key signature.

`arpeggio-direction` (direction)
: If set, put an arrow on the arpeggio squiggly line.

`arrow-length` (number)
: Arrow length.

`arrow-width` (number)
: Arrow width.

`auto-knee-gap` (dimension, in staff space)
: If a gap is found between note heads where a horizontal beam fits that is larger than this number, make a kneed beam.

`average-spacing-wishes` (boolean)
: If set, the spacing wishes are averaged over staves.

`avoid-note-head` (boolean)
: If set, the stem of a chord does not pass through all note heads, but starts at the last note head.

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `around`, `inside`, `outside`. If unset, scripts and slurs ignore each other. `around` only moves the script if there is a collision; `outside` always moves the script.

`axes` (list) List of axis numbers. In the case of alignment grobs, this should contain only one number.

`bar-size` (dimension, in staff space)
: The size of a bar line.

`barre-type` (symbol)
: Type of barre indication used in a fret diagram. Choices include `curved` and `straight`.

`base-shortest-duration` (moment)
: Spacing is based on the shortest notes in a piece. Normally, pieces are spaced as if notes at least as short as this are present.

`baseline-skip` (dimension, in staff space)
: Distance between base lines of multiple lines of text.

`beam-thickness` (dimension, in staff space)
: Beam thickness, measured in `staff-space` units.

`beam-width` (dimension, in staff space)
: Width of the tremolo sign.

`beamed-stem-shorten` (list)
: How much to shorten beamed stems, when their direction is forced. It is a list, since the value is different depending on the number of flags and beams.

`beaming` (pair)
: Pair of number lists. Each number list specifies which beams to make. `0` is the central beam, `1` is the next beam toward the note, etc. This

information is used to determine how to connect the beaming patterns from stem to stem inside a beam.

`before-line-breaking` (boolean)
: Dummy property, used to trigger a callback function.

`between-cols` (pair)
: Where to attach a loose column to.

`bound-details` (list)
: An alist of properties for determining attachments of spanners to edges.

`bound-padding` (number)
: The amount of padding to insert around spanner bounds.

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`bracket-visibility` (boolean or symbol)
: This controls the visibility of the tuplet bracket. Setting it to false prevents printing of the bracket. Setting the property to `if-no-beam` makes it print only if there is no beam associated with this tuplet bracket.

`break-align-anchor` (number)
: Grobs aligned to this break-align grob will have their X-offsets shifted by this number. In barlines, for example, this is used to position grobs relative to the (visual) center of the barline.

`break-align-anchor-alignment` (number)
: Read by ly:break-aligned-interface::calc-extent-aligned-anchor for aligning an anchor to a grobs extent

`break-align-orders` (vector)
: Defines the order in which prefatory matter (clefs, key signatures) appears. The format is a vector of length 3, where each element is one order for end-of-line, middle of line, and start-of-line, respectively. An order is a list of symbols.

  For example, clefs are put after key signatures by setting

```
\override Score.BreakAlignment #'break-align-orders =
  #(make-vector 3 '(span-bar
                    breathing-sign
                    staff-bar
                    key
                    clef
                    time-signature))
```

`break-align-symbol` (symbol)
: This key is used for aligning and spacing breakable items.

`break-align-symbols` (list)
: A list of symbols that determine which break-aligned grobs to align this to. If the grob selected by the first symbol in the list is invisible due to break-visibility, we will align to the next grob (and so on).

`break-overshoot` (pair of numbers)
: How much does a broken spanner stick out of its bounds?

`break-visibility` (vector)
: A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`breakable` (boolean)
: Allow breaks here.

`c0-position` (integer)
: An integer indicating the position of middle C.

`clip-edges` (boolean)
: Allow outward pointing beamlets at the edges of beams?

`collapse-height` (dimension, in staff space)
: Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

`color` (list)
: The color of this grob.

`common-shortest-duration` (moment)
: The most common shortest note length. This is used in spacing. Enlarging this sets the score tighter.

`concaveness` (number)
: A beam is concave if its inner stems are closer to the beam than the two outside stems. This number is a measure of the closeness of the inner stems. It is used for damping the slope of the beam.

`connect-to-neighbor` (pair)
: Pair of booleans, indicating whether this grob looks as a continued break.

`control-points` (list)
: List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`damping` (number)
: Amount of beam slope damping.

`dash-fraction` (number)
: Size of the dashes, relative to `dash-period`. Should be between `0.0` (no line) and `1.0` (continuous line).

`dash-period` (number)
: The length of one dash together with whitespace. If negative, no line is drawn at all.

`default-direction` (direction)
: Direction determined by note head positions.

`digit-names` (unknown)
: Names for string finger digits.

`direction` (direction)
: If `side-axis` is `1` (or `#X`), then this property determines whether the object is placed `#LEFT`, `#CENTER` or `#RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `#UP`, `#CENTER` or `#DOWN`. Numerical values may also be used: `#UP`=`1`, `#DOWN`=`-1`, `#LEFT`=`-1`, `#RIGHT`=`1`, `CENTER`=`0` but also other numerical values are permitted.

`dot-color` (symbol)
: Color of dots. Options include `black` and `white`.

`dot-count` (integer)
: The number of dots.

`dot-radius` (number)
: Radius of dots.

`duration-log` (integer)
: The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`eccentricity` (number)
: How asymmetrical to make a slur. Positive means move the center to the right.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`edge-text` (pair)
: A pair specifying the texts to be set at the edges: `(`*`left-text . right-text`*`)`.

`expand-limit` (integer)
: Maximum number of measures expanded in church rests.

`extra-X-extent` (pair of numbers)
: A grob is enlarged in X dimension by this much.

`extra-Y-extent` (pair of numbers)
: See `extra-X-extent`.

`extra-dy` (number)
: Slope glissandi this much extra.

`extra-offset` (pair of numbers)
: A pair representing an offset. This offset is added just before outputting the symbol, so the typesetting engine is completely oblivious to it. The values are measured in `staff-space` units of the staff's `StaffSymbol`.

`extra-spacing-width` (pair of numbers)
: In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`finger-code` (symbol)
: Code for the type of fingering indication in a fret diagram. Options include `none`, `in-dot`, and `below-string`.

`flag-count` (number)
: The number of tremolo beams.

`flag-style` (symbol)
: A string determining what style of flag glyph is typeset on a `Stem`. Valid options include `()` and `mensural`. Additionally, `no-flag` switches off the flag.

`font-encoding` (symbol)
: The font encoding is the broadest category for selecting a font. Options include: `fetaMusic`, `fetaNumber`, `TeX-text`, `TeX-math`, `fetaBraces`, `fetaDynamic`.

`font-family` (symbol)
: The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`font-name` (string)
: Specifies a file name (without extension) of the font to load. This setting overrides selection using `font-family`, `font-series` and `font-shape`.

`font-series` (symbol)
: Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-shape` (symbol)
: Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number)
: The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger, 6 steps are exactly a factor 2 larger. Fractional values are allowed.

`force-hshift` (number)
: This specifies a manual shift for notes in collisions. The unit is the note head width of the first voice note. This is used by `note-collision-interface`.

`forced` (boolean)
: Manually forced accidental.

`fraction` (pair of numbers)
: Numerator and denominator of a time signature object.

`french-beaming` (boolean)
: Use French beaming style for this stem. The stem stops at the innermost beams.

`fret-count` (integer)
: The number of frets in a fret diagram.

`full-size-change` (boolean)
: Don't make a change clef smaller.

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`gap-count` (integer)
: Number of gapped beams for tremolo.

`glyph` (string)
: A string determining what 'style' of glyph is typeset. Valid choices depend on the function that is reading this property.

`glyph-name-alist` (list)
: An alist of key-string pairs.

`grow-direction` (direction)
: Crescendo or decrescendo?

`hair-thickness` (number)
: Thickness of the thin line in a bar line.

`head-direction` (direction)
: Are the note heads left or right in a semitie?

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`height-limit` (dimension, in staff space)
: Maximum slur height: The longer the slur, the closer it is to this height.

`horizontal-shift` (integer)
: An integer that identifies ranking of `NoteColumn`s for horizontal shifting. This is used by `note-collision-interface`.

`horizontal-skylines` (unknown)
: Two skylines, one to the left and one to the right of this grob.

`ignore-collision` (boolean)
: If set, don't do note collision resolution on this `NoteColumn`.

`implicit` (boolean)
: Is this an implicit bass figure?

`infinite-spacing-height` (boolean)
: If true, then for the purposes of horizontal spacing, treat this item as though it were infinitely tall. That is, no object from another column is allowed to stick above or below this item.

`inspect-index` (integer)
: If debugging is set, set beam and slur configuration to this index, and print the respective scores.

`inspect-quants` (pair of numbers)
: If debugging is set, set beam and slur quants to this position, and print the respective scores.

`keep-fixed-while-stretching` (boolean)
: A grob with this property set to true is fixed relative to the staff above it when systems are stretched.

`keep-inside-line` (boolean)
: If set, this column cannot have things sticking into the margin.

`kern` (dimension, in staff space)
: Amount of extra white space to add. For bar lines, this is the amount of space after a thick line.

`knee` (boolean)
: Is this beam kneed?

`knee-spacing-correction` (number)
: Factor for the optical correction amount for kneed beams. Set between `0` for no correction and `1` for full correction.

`label-dir` (direction)
: Side to which a label is attached. `-1` for left, `1` for right.

`labels` (list)
: List of labels (symbols) placed on a column

`layer` (number)
: The output layer (a value between 0 and 2: Layers define the order of printing objects. Objects in lower layers are overprinted by objects in higher layers.

`ledger-line-thickness` (pair of numbers)
: The thickness of ledger lines. It is the sum of 2 numbers: The first is the factor for line thickness, and the second for staff space. Both contributions are added.

`left-bound-info` (list)
: An alist of properties for determining attachments of spanners to edges.

`left-padding` (dimension, in staff space)
: The amount of space that is put left to an object (e.g., a group of accidentals).

`length` (dimension, in staff space)
: User override for the stem length of unbeamed stems.

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`line-break-penalty` (number)
: Penalty for a line break at this column. This affects the choices of the line breaker; it avoids a line break at a column with a positive penalty and prefer a line break at a column with a negative penalty.

`line-break-permission` (symbol)
: Instructs the line breaker on whether to put a line break at this column. Can be `force` or `allow`.

`line-break-system-details` (list)
: An alist of properties to use if this column is the start of a system.

`line-count` (integer)
: The number of staff lines.

`line-positions` (list)
: Vertical positions of staff lines.

`line-thickness` (number)
: The thickness of the tie or slur contour.

`long-text` (markup)
: Text markup. See notation reference, Section 1.8.2 [Text markup], page 119 .

`max-beam-connect` (integer)
: Maximum number of beams to connect to beams from this stem. Further beams are typeset as beamlets.

`max-stretch` (number)
: The maximum amount that this `VerticalAxisGroup` can be vertically stretched (for example, in order to better fill a page).

`measure-count` (integer)
: The number of measures for a multi-measure rest.

`measure-length` (moment)
: Length of a measure. Used in some spacing situations.

`merge-differently-dotted` (boolean)
: Merge note heads in collisions, even if they have a different number of dots. This is normal notation for some types of polyphonic music.

  `merge-differently-dotted` only applies to opposing stem directions (i.e., voice 1 & 2).

`merge-differently-headed` (boolean)
: Merge note heads in collisions, even if they have different note heads. The smaller of the two heads is rendered invisible. This is used in polyphonic guitar notation. The value of this setting is used by `note-collision-interface`.

  `merge-differently-headed` only applies to opposing stem directions (i.e., voice 1 & 2).

`minimum-X-extent` (pair of numbers)
: Minimum size of an object in X dimension, measured in `staff-space` units.

`minimum-Y-extent` (pair of numbers)
: See `minimum-X-extent`.

`minimum-distance` (dimension, in staff space)
: Minimum distance between rest and notes or beam.

`minimum-length` (dimension, in staff space)
: Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`minimum-length-fraction` (number)
: Minimum length of ledger line as fraction of note head size.

`minimum-space` (dimension, in staff space)
: Minimum distance that the victim should move (after padding).

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

`neutral-position` (number)
: Position (in half staff spaces) where to flip the direction of custos stem.

`next` (layout object)
: Object that is next relation (e.g., the lyric syllable following an extender.

`no-alignment` (boolean)
: If set, don't place this grob in a VerticalAlignment; rather, place it using its own Y-offset callback

`no-ledgers` (boolean)
: If set, don't draw ledger lines on this object.

`no-stem-extend` (boolean)
: If set, notes with ledger lines do not get stems extending to the middle staff line.

`non-default` (boolean)
: Set for manually specified clefs.

`non-musical` (boolean)
: True if the grob belongs to a `NonMusicalPaperColumn`.

`note-names` (vector)
: Vector of strings containing names for easy-notation note heads.

`number-type` (symbol)
: Type of numbers to use in label. Choices include `roman-lower`, `roman-upper`, and `arabic`.

`outside-staff-horizontal-padding` (number)
: By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbour.

`outside-staff-padding` (number)
: The padding to place between this grob and the staff when spacing according to `outside-staff-priority`.

`outside-staff-priority` (number)
: If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`packed-spacing` (boolean)
: If set, the notes are spaced as tightly as possible.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`page-break-penalty` (number)
: Penalty for page break at this column. This affects the choices of the page breaker; it avoids a page break at a column with a positive penalty and prefer a page break at a column with a negative penalty.

`page-break-permission` (symbol)
: Instructs the page breaker on whether to put a page break at this column. Can be `force` or `allow`.

`page-turn-penalty` (number)
: Penalty for a page turn at this column. This affects the choices of the page breaker; it avoids a page turn at a column with a positive penalty and prefer a page turn at a column with a negative penalty.

`page-turn-permission` (symbol)
: Instructs the page breaker on whether to put a page turn at this column. Can be `force` or `allow`.

`parenthesized` (boolean)
: Parenthesize this grob.

`positions` (pair of numbers)
: Pair of staff coordinates (*`left . right`*), where both *left* and *right* are in `staff-space` units of the current staff.

  For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`ratio` (number)
: Parameter for slur shape. The higher this number, the quicker the slur attains its `height-limit`.

`remove-empty` (boolean)
: If set, remove group if it contains no interesting items.

`remove-first` (boolean)
: Remove the first staff of a orchestral score?

`restore-first` (boolean)
: Print a natural before the accidental.

`rhythmic-location` (rhythmic location)
: Where (bar number, measure position) in the score.

`right-bound-info` (list)
: An alist of properties for determining attachments of spanners to edges.

`right-padding` (dimension, in staff space)
: Space to insert on the right side of an object (e.g., between note and its accidentals).

`rotation` (list)
: Number of degrees to rotate this object, and what point to rotate around. For example, `#'(45 0 0)` rotates by 45 degrees around the center of this object.

`same-direction-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. This amount is used for stems with the same direction to compensate for note-head to stem distance.

`script-priority` (number)
: A sorting key that determines in what order a script is within a stack of scripts.

`self-alignment-X` (number)
: Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Values in-between may also be specified.

`self-alignment-Y` (number)
: Like `self-alignment-X` but for the Y axis.

`shorten-pair` (pair of numbers)
: The lengths to shorten a text-spanner on both sides, for example a pedal bracket. Positive values shorten the text-spanner, while negative values lengthen it.

`shortest-duration-space` (dimension, in staff space)
: Start with this much space for the shortest duration. This is expressed in `spacing-increment` as unit. See also `spacing-spanner-interface`.

`shortest-playing-duration` (moment)
: The duration of the shortest playing here.

`shortest-starter-duration` (moment)
: The duration of the shortest note that starts here.

`side-axis` (number)
: If the value is `#X` (or equivalently `1`), the object is placed horizontally next to the other object. If the value is `#Y` or `0`, it is placed vertically.

`side-relative-direction` (direction)
: Multiply direction of `direction-source` with this to get the direction of this object.

`size` (number)
: Size of object, relative to standard size.

`slope` (number)
: The slope of this object.

`slur-padding` (number)
: Extra distance between slur and script.

`space-alist` (list)
: A table that specifies distances between prefatory items, like clef and time-signature. The format is an alist of spacing tuples: `(`*`break-align-symbol type . distance`*`)`, where *type* can be the symbols `minimum-space` or `extra-space`.

`space-to-barline` (boolean)
: If set, the distance between a note and the following non-musical column will be measured to the barline instead of to the beginning of the non-musical column. If there is a clef change followed by a barline, for example, this means that we will try to space the non-musical column as though the clef is not there.

`spacing-increment` (number)
: Add this much space for a doubled duration. Typically, the width of a note head. See also `spacing-spanner-interface`.

`springs-and-rods` (boolean)
: Dummy variable for triggering spacing routines.

`stacking-dir` (direction)
: Stack objects in which direction?

`staff-padding` (dimension, in staff space)
: Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamic **p** and **f**) on their baselines.

`staff-position` (number)
: Vertical position, measured in half staff spaces, counted from the middle line.

`staff-space` (dimension, in staff space)
: Amount of space between staff lines, expressed in global `staff-space`.

`stem-attachment` (pair of numbers)
: A `(`*`x . y`*`)` pair where the stem attaches to the notehead.

`stem-end-position` (number)
: Where does the stem end (the end is opposite to the support-head)?

`stem-spacing-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. For opposite directions, this amount is the correction for two normal sized stems that overlap completely.

`stemlet-length` (number)
: How long should a stem over a rest be?

`stencil` (unknown)
: The symbol to print.

`stencils` (list)
: Multiple stencils, used as intermediate value.

`strict-grace-spacing` (boolean)
: If set, grace notes are not spaced separately, but put before musical columns.

`strict-note-spacing` (boolean)
: If set, unbroken columns with non-musical material (clefs, barlines, etc.) are not spaced separately, but put before musical columns.

`string-count` (integer)
: The number of strings in a fret diagram.

`string-fret-finger-combinations` (list)
: List consisting of `(`*`string-number fret-number finger-number`*`)` entries.

`stroke-style` (string)
: Set to `"grace"` to turn stroke through flag on.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`text` (markup)
: Text markup. See notation reference, Section 1.8.2 [Text markup], page 119 .

`text-direction` (direction)
: This controls the ordering of the words. The default `RIGHT` is for roman text. Arabic or Hebrew should use `LEFT`.

`thick-thickness` (number)
: Bar line thickness, measured in `line-thickness`.

`thickness` (number)
: Line thickness, generally measured in `line-thickness`.

`thin-kern` (number)
: The space after a hair-line in a bar line.

`threshold` (pair of numbers)
: `(`*`min . max`*`)`, where *min* and *max* are dimensions in staff space.

`tie-configuration` (list)
: List of `(`*`position . dir`*`)` pairs, indicating the desired tie configuration, where *position* is the offset from the center of the staff in staff space and *dir* indicates the direction of the tie (`1`=>up, `-1`=>down, `0`=>center). A non-pair entry in the list causes the corresponding tie to be formatted automatically.

`to-barline` (boolean)
: If true, the spanner will stop at that barline just before it would otherwise stop.

`transparent` (boolean)
: This makes the grob invisible.

`uniform-stretching` (boolean)
: If set, items stretch proportionally to their durations. This looks better in complex polyphonic patterns.

`used` (boolean)
: If set, this spacing column is kept in the spacing problem

`vertical-skylines` (unknown)
: Two skylines, one above and one below this grob.

`when` (moment)
: Global time step associated with this column happen?

`width` (dimension, in staff space)
: The width of a grob measured in staff space.

`word-space` (dimension, in staff space)
: Space to insert between words in texts.

`zigzag-length` (dimension, in staff space)
: The length of the lines of a zigzag, relative to `zigzag-width`. A value of `1` gives 60-degree zigzags.

`zigzag-width` (dimension, in staff space)
: The width of one zigzag squiggle. This number is adjusted slightly so that the glissando line can be constructed from a whole number of squiggles.

B.11 Identifiers

`autochange` - *music* (music)
: (undocumented; fixme)

`endSpanners` - *music* (music)
: (undocumented; fixme)

`musicMap` - *proc* (procedure) *mus* (music)
: (undocumented; fixme)

`scoreTweak` - *name* (string)
: (undocumented; fixme)

`killCues` - *music* (music)
: (undocumented; fixme)

`appoggiatura` - *music* (music)
: (undocumented; fixme)

`cueDuring` - *what* (string) *dir* (direction) *main-music* (music)
: (undocumented; fixme)

`breathe` -
: (undocumented; fixme)

`acciaccatura` - *music* (music)
: (undocumented; fixme)

`removeWithTag` - *tag* (symbol) *music* (music)
: (undocumented; fixme)

`tocItem` - *text* (markup)
: Add a line to the table of content, using the `tocItemMarkup` paper variable markup

`applyContext` - *proc* (procedure)
: (undocumented; fixme)

`pitchedTrill` - *main-note* (music) *secondary-note* (music)
(undocumented; fixme)

`assertBeamQuant` - *l* (pair) *r* (pair)
(undocumented; fixme)

`oldaddlyrics` - *music* (music) *lyrics* (music)
(undocumented; fixme)

`addQuote` - *name* (string) *music* (music)
(undocumented; fixme)

`featherDurations` - *factor* (moment) *argument* (music)
(undocumented; fixme)

`transposition` - *pitch-note* (music)
(undocumented; fixme)

`bendAfter` - *delta* (integer)
(undocumented; fixme)

`noPageTurn` -
(undocumented; fixme)

`noPageBreak` -
(undocumented; fixme)

`balloonText` - *offset* (pair of numbers) *text* (markup)
(undocumented; fixme)

`barNumberCheck` - *n* (integer)
(undocumented; fixme)

`addInstrumentDefinition` - *name* (string) *lst* (list)
(undocumented; fixme)

`shiftDurations` - *dur* (integer) *dots* (integer) *arg* (music)
(undocumented; fixme)

`unfoldRepeats` - *music* (music)
(undocumented; fixme)

`applyMusic` - *func* (procedure) *music* (music)
(undocumented; fixme)

`quoteDuring` - *what* (string) *main-music* (music)
(undocumented; fixme)

`keepWithTag` - *tag* (symbol) *music* (music)
(undocumented; fixme)

`displayLilyMusic` - *music* (music)
(undocumented; fixme)

`rightHandFinger` - *finger* (number or string)
(undocumented; fixme)

`bar` - *type* (string)
(undocumented; fixme)

`tag` - *tag* (symbol) *arg* (music)
(undocumented; fixme)

`assertBeamSlope` - *comp* (procedure)
(undocumented; fixme)

`tweak` - *sym* (symbol) *val* (any type) *arg* (music)
(undocumented; fixme)

`transposedCueDuring` - *what* (string) *dir* (direction) *pitch-note* (music) *main-music* (music)
(undocumented; fixme)

`overrideProperty` - *name* (string) *property* (symbol) *value* (any type)
(undocumented; fixme)

`withMusicProperty` - *sym* (symbol) *val* (any type) *music* (music)
(undocumented; fixme)

`octave` - *pitch-note* (music)
(undocumented; fixme)

`applyOutput` - *ctx* (symbol) *proc* (procedure)
(undocumented; fixme)

`afterGrace` - *main* (music) *grace* (music)
(undocumented; fixme)

`allowPageTurn` -
(undocumented; fixme)

`compressMusic` - *fraction* (pair of numbers) *music* (music)
(undocumented; fixme)

`displayMusic` - *music* (music)
(undocumented; fixme)

`pageTurn` -
(undocumented; fixme)

`balloonGrobText` - *grob-name* (symbol) *offset* (pair of numbers) *text* (markup)
(undocumented; fixme)

`includePageLayoutFile` -
(undocumented; fixme)

`instrumentSwitch` - *name* (string)
(undocumented; fixme)

`makeClusters` - *arg* (music)
(undocumented; fixme)

`parallelMusic` - *voice-ids* (list) *music* (music)
(undocumented; fixme)

`resetRelativeOctave` - *reference-note* (music)
(undocumented; fixme)

`label` - *label* (symbol)
(undocumented; fixme)

`grace` - *music* (music)
(undocumented; fixme)

`spacingTweaks` - *parameters* (list)
(undocumented; fixme)

`clef` - *type* (string)
(undocumented; fixme)

`partcombine` - *part1* (music) *part2* (music)
(undocumented; fixme)

`parenthesize` - *arg* (music)
(undocumented; fixme)

`pageBreak` -
(undocumented; fixme)

B.12 Scheme functions

`dispatcher` *x* [Function]
Is *x* a `Dispatcher` object?

`listener` *x* [Function]
Is *x* a `Listener` object?

`ly:add-file-name-alist` *alist* [Function]
Add mappings for error messages from *alist*.

`ly:add-interface` *a b c* [Function]
Add an interface description.

`ly:add-listener` *list disp cl* [Function]
Add the listener *list* to the dispatcher *disp*. Whenever *disp* hears an event of class *cl*, it is forwarded to *list*.

`ly:add-option` *sym val description* [Function]
Add a program option *sym* with default *val*.

`ly:all-grob-interfaces` [Function]
Get a hash table with all interface descriptions.

`ly:all-options` [Function]
Get all option settings in an alist.

`ly:all-stencil-expressions` [Function]
Return all symbols recognized as stencil expressions.

`ly:assoc-get` *key alist default-value* [Function]
Return value if *key* in *alist*, else `default-value` (or `#f` if not specified).

`ly:book-add-score!` *book-smob score* [Function]
Add *score* to *book-smob* score list.

`ly:book-process` *book-smob default-paper default-layout output* [Function]
Print book. *output* is passed to the backend unchanged. For example, it may be a string (for file based outputs) or a socket (for network based output).

`ly:book-process-to-systems` *book-smob default-paper default-layout output* [Function]
Print book. *output* is passed to the backend unchanged. For example, it may be a string (for file based outputs) or a socket (for network based output).

`ly:box?` *x* [Function]
Is *x* a `Box` object?

`ly:bp` *num* [Function]
num bigpoints (1/72th inch).

`ly:bracket` *a iv t p* [Function]
Make a bracket in direction *a*. The extent of the bracket is given by *iv*. The wings protude by an amount of *p*, which may be negative. The thickness is given by *t*.

`ly:broadcast` *disp ev* [Function]
Send the stream event *ev* to the dispatcher *disp*.

`ly:camel-case->lisp-identifier` *name-sym* [Function]
Convert `FooBar_Bla` to `foo-bar-bla` style symbol.

`ly:chain-assoc-get` *key achain dfault* [Function]
Return value for *key* from a list of alists *achain*. If no entry is found, return *dfault* or `#f` if no *dfault* is specified.

`ly:clear-anonymous-modules` [Function]
Plug a GUILE 1.6 and 1.7 memory leak by breaking a weak reference pointer cycle explicitly.

`ly:cm` *num* [Function]
num cm.

`ly:command-line-code` [Function]
The Scheme code specified on command-line with '`-e`'.

`ly:command-line-options` [Function]
The Scheme options specified on command-line with '`-d`'.

`ly:command-line-verbose?` [Function]
Was `be_verbose_global` set?

`ly:connect-dispatchers` *to from* [Function]
Make the dispatcher *to* listen to events from *from*.

`ly:context-event-source` *context* [Function]
Return `event-source` of context *context*.

`ly:context-events-below` *context* [Function]
Return a `stream-distributor` that distributes all events from *context* and all its subcontexts.

`ly:context-find` *context name* [Function]
Find a parent of *context* that has name or alias *name*. Return `#f` if not found.

`ly:context-grob-definition` *context name* [Function]
Return the definition of *name* (a symbol) within *context* as an alist.

`ly:context-id` *context* [Function]
Return the ID string of *context*, i.e., for `\context Voice = one ...` return the string `one`.

`ly:context-name` *context* [Function]
Return the name of *context*, i.e., for `\context Voice = one ...` return the symbol `Voice`.

`ly:context-now` *context* [Function]
Return `now-moment` of context *context*.

`ly:context-parent` *context* [Function]
Return the parent of *context*, `#f` if none.

`ly:context-property` *c name* [Function]
Return the value of *name* from context *c*.

`ly:context-property-where-defined` *context name* [Function]
Return the context above *context* where *name* is defined.

`ly:context-pushpop-property` *context grob eltprop val* [Function]
Do a single `\override` or `\revert` operation in *context*. The grob definition *grob* is extended with *eltprop* (if *val* is specified) or reverted (if unspecified).

`ly:context-set-property!` *context name val* [Function]
Set value of property *name* in context *context* to *val*.

`ly:context-unset-property` *context name* [Function]
Unset value of property *name* in context *context*.

`ly:context?` *x* [Function]
Is *x* a `Context` object?

`ly:default-scale` [Function]
Get the global default scale.

`ly:dimension?` *d* [Function]
Return *d* as a number. Used to distinguish length variables from normal numbers.

`ly:dir?` *s* [Function]
A type predicate. The direction `s` is `-1`, `0` or `1`, where `-1` represents left or down and `1` represents right or up.

`ly:duration->string` *dur* [Function]
Convert *dur* to a string.

`ly:duration-dot-count` *dur* [Function]
Extract the dot count from *dur*.

`ly:duration-factor` *dur* [Function]
Extract the compression factor from *dur*. Return it as a pair.

`ly:duration-length` *dur* [Function]
The length of the duration as a `moment`.

`ly:duration-log` *dur* [Function]
Extract the duration log from *dur*.

`ly:duration<?` *p1 p2* [Function]
Is *p1* shorter than *p2*?

`ly:duration?` *x* [Function]
Is *x* a `Duration` object?

`ly:effective-prefix` [Function]
Return effective prefix.

`ly:error` *str rest* [Function]
A Scheme callable function to issue the error *str*. The error is formatted with `format` and *rest*.

`ly:eval-simple-closure` *delayed closure scm-start scm-end* [Function]
Evaluate a simple *closure* with the given *delayed* argument. If *scm-start* and *scm-end* are defined, evaluate it purely with those start and end points.

`ly:event-deep-copy` *m* [Function]
Copy *m* and all sub expressions of *m*.

`ly:event-property` *sev sym* [Function]
Get the property *sym* of stream event *mus*. If *sym* is undefined, return `'()`.

`ly:event-set-property!` *ev sym val* [Function]
Set property *sym* in event *ev* to *val*.

`ly:expand-environment` *str* [Function]
Expand `$VAR` and `${VAR}` in *str*.

`ly:export` *arg* [Function]
Export a Scheme object to the parser so it is treated as an identifier.

`ly:find-file` *name* [Function]
Return the absolute file name of *name*, or `#f` if not found.

`ly:font-config-display-fonts` [Function]
Dump a list of all fonts visible to FontConfig.

`ly:font-config-get-font-file` *name* [Function]
Get the file for font *name*.

`ly:font-design-size` *font* [Function]
Given the font metric *font*, return the design size, relative to the current output-scale.

`ly:font-file-name` *font* [Function]
Given the font metric *font*, return the corresponding file name.

`ly:font-get-glyph` *font name* [Function]
Return a stencil from *font* for the glyph named *name*. *font* must be available as an AFM file. If the glyph is not available, return `#f`.

`ly:font-glyph-name-to-charcode` *font name* [Function]
Return the character code for glyph *name* in *font*.

`ly:font-glyph-name-to-index` *font name* [Function]
Return the index for *name* in *font*.

`ly:font-index-to-charcode` *font index* [Function]
Return the character code for *index* in *font*.

`ly:font-load` *name* [Function]
Load the font *name*.

`ly:font-magnification` *font* [Function]
Given the font metric *font*, return the magnification, relative to the current output-scale.

`ly:font-metric?` *x* [Function]
Is *x* a `Font_metric` object?

`ly:font-name` *font* [Function]
Given the font metric *font*, return the corresponding name.

`ly:font-sub-fonts` *font* [Function]
Given the font metric *font* of an OpenType font, return the names of the subfonts within *font*.

`ly:format` *str rest* [Function]
LilyPond specific format, supporting `~a` and `~[0-9]f`.

`ly:format-output` *context* [Function]
Given a global context in its final state, process it and return the `Music_output` object in its final state.

`ly:get-all-function-documentation` [Function]
Get a hash table with all LilyPond Scheme extension functions.

`ly:get-all-translators` [Function]
Return a list of all translator objects that may be instantiated.

`ly:get-glyph` *font index* [Function]
Retrieve a stencil for the glyph numbered *index* in *font*.

`ly:get-listened-event-classes` [Function]
Return a list of all event classes that some translator listens to.

`ly:get-option` *var* [Function]
Get a global option setting.

`ly:gettext` *string* [Function]
A Scheme wrapper function for `gettext`.

`ly:grob-alist-chain` *grob global* [Function]
Get an alist chain for grob *grob*, with *global* as the global default. If unspecified, `font-defaults` from the layout block is taken.

`ly:grob-array-length` *grob-arr* [Function]
Return the length of *grob-arr*.

`ly:grob-array-ref` *grob-arr index* [Function]
Retrieve the *index*th element of *grob-arr*.

`ly:grob-array?` *x* [Function]
Is *x* a `Grob_array` object?

`ly:grob-basic-properties` *grob* [Function]
Get the immutable properties of *grob*.

`ly:grob-common-refpoint` *grob other axis* [Function]
Find the common refpoint of *grob* and *other* for *axis*.

`ly:grob-common-refpoint-of-array` *grob others axis* [Function]
Find the common refpoint of *grob* and *others* (a grob-array) for *axis*.

`ly:grob-default-font` *grob* [Function]
Return the default font for grob *gr*.

`ly:grob-extent` *grob refp axis* [Function]
Get the extent in *axis* direction of *grob* relative to the grob *refp*.

`ly:grob-interfaces` *grob* [Function]
Return the interfaces list of grob *grob*.

`ly:grob-layout` *grob* [Function]
Get `\layout` definition from grob *grob*.

`ly:grob-object` *grob sym* [Function]
Return the value of a pointer in grob *g* of property *sym*. It returns `'()` (end-of-list) if *sym* is undefined in *g*.

`ly:grob-original` *grob* [Function]
Return the unbroken original grob of *grob*.

`ly:grob-parent` *grob axis* [Function]
Get the parent of *grob*. *axis* is 0 for the X-axis, 1 for the Y-axis.

`ly:grob-pq<?` *a b* [Function]
Compare two grob priority queue entries. This is an internal function.

`ly:grob-properties` *grob* [Function]
Get the mutable properties of *grob*.

`ly:grob-property` *grob sym deflt* [Function]
Return the value of a value in grob *g* of property *sym*. It returns `'()` (end-of-list) or *deflt* (if specified) if *sym* is undefined in *g*.

`ly:grob-property-data` *grob sym* [Function]
Retrieve *sym* for *grob* but don't process callbacks.

`ly:grob-relative-coordinate` *grob refp axis* [Function]
Get the coordinate in *axis* direction of *grob* relative to the grob *refp*.

`ly:grob-robust-relative-extent` *grob refp axis* [Function]
Get the extent in *axis* direction of *grob* relative to the grob *refp*, or `(0,0)` if empty.

`ly:grob-script-priority-less` *a b* [Function]
Compare two grobs by script priority. For internal use.

`ly:grob-set-property!` *grob sym val* [Function]
Set *sym* in grob *grob* to value *val*.

`ly:grob-staff-position` *sg* [Function]
Return the Y-position of *sg* relative to the staff.

`ly:grob-suicide!` *grob* [Function]
Kill *grob*.

`ly:grob-system` *grob* [Function]
Return the system grob of *grob*.

`ly:grob-translate-axis!` *grob d a* [Function]
Translate *g* on axis *a* over distance *d*.

`ly:grob?` *x* [Function]
Is *x* a `Grob` object?

`ly:gulp-file` *name size* [Function]
Read the file *name*, and return its contents in a string. The file is looked up using the search path.

`ly:hash-table-keys` *tab* [Function]
Return a list of keys in *tab*.

`ly:inch` *num* [Function]
num inches.

`ly:input-both-locations` *sip* [Function]
Return input location in *sip* as `(file-name first-line first-column last-line last-column)`.

`ly:input-file-line-char-column` *sip* [Function]
Return input location in *sip* as `(file-name line char column)`.

`ly:input-location?` *x* [Function]
Is *x* an `input-location`?

`ly:input-message` *sip msg rest* [Function]
Print *msg* as a GNU compliant error message, pointing to the location in *sip*. *msg* is interpreted similar to `format`'s argument, using *rest*.

`ly:interpret-music-expression` *mus ctx* [Function]
Interpret the music expression *mus* in the global context *ctx*. The context is returned in its final state.

`ly:interpret-stencil-expression` *expr func arg1 offset* [Function]
Parse *expr*, feed bits to *func* with first arg *arg1* having offset *offset*.

`ly:intlog2` *d* [Function]
The 2-logarithm of $1/d$.

`ly:is-listened-event-class` *sym* [Function]
Is *sym* a listened event class?

`ly:item-break-dir` *it* [Function]
The break status direction of item *it*. `-1` means end of line, `0` unbroken, and `1` beginning of line.

`ly:item?` *g* [Function]
Is *g* an `Item` object?

`ly:iterator?` *x* [Function]
Is *x* a `Music_iterator` object?

`ly:lexer-keywords` *lexer* [Function]
Return a list of `(KEY . CODE)` pairs, signifying the LilyPond reserved words list.

`ly:lily-lexer?` *x* [Function]
Is *x* a `Lily_lexer` object?

`ly:lily-parser?` *x* [Function]
Is *x* a `Lily_parser` object?

`ly:load-text-dimensions` *dimension-alist* [Function]
Load dimensions from TEX in a `(KEY . (W H D))` format alist.

`ly:make-book` *paper header scores* [Function]
Make a `\book` of *paper* and *header* (which may be `#f` as well) containing `\scores`.

`ly:make-dispatcher` [Function]
Return a newly created dispatcher.

`ly:make-duration` *length dotcount num den* [Function]
length is the negative logarithm (base 2) of the duration: 1 is a half note, 2 is a quarter note, 3 is an eighth note, etc. The number of dots after the note is given by the optional argument *dotcount*.

The duration factor is optionally given by *num* and *den*.

A duration is a musical duration, i.e., a length of time described by a power of two (whole, half, quarter, etc.) and a number of augmentation dots.

`ly:make-global-context` *output-def* [Function]
Set up a global interpretation context, using the output block *output_def*. The context is returned.

`ly:make-global-translator` *global* [Function]
Create a translator group and connect it to the global context *global*. The translator group is returned.

`ly:make-listener` *callback* [Function]
Create a listener. Any time the listener hears an object, it will call *callback* with that object.
callback should take exactly one argument.

`ly:make-moment` *n d gn gd* [Function]
Create the rational number with main timing n/d, and optional grace timing gn/gd.

A *moment* is a point in musical time. It consists of a pair of rationals (m, g), where m is the timing for the main notes, and g the timing for grace notes. In absence of grace notes, g is zero.

`ly:make-music` *props* [Function]
Make a C++ `Music` object and initialize it with *props*.

This function is for internal use and is only called by `make-music`, which is the preferred interface for creating music objects.

`ly:make-music-function` *signature func* [Function]
Make a function to process music, to be used for the parser. `func` is the function, and `signature` describes its arguments. `signature` is a list containing either `ly:music?` predicates or other type predicates.

`ly:make-output-def` [Function]
Make an output definition.

`ly:make-page-label-marker` *label* [Function]
Return page marker with label.

`ly:make-page-permission-marker` *symbol permission* [Function]
Return page marker with page breaking and turning permissions.

`ly:make-pango-description-string` *chain size* [Function]
Make a `PangoFontDescription` string for the property alist *chain* at size *size*.

`ly:make-paper-outputter` *port format* [Function]
Create an outputter that evaluates within `output-`*format*, writing to *port*.

`ly:make-pitch` *octave note alter* [Function]
octave is specified by an integer, zero for the octave containing middle C. *note* is a number from 0 to 6, with 0 corresponding to pitch C and 6 corresponding to pitch B. *alter* is a rational number of whole tones for alteration.

`ly:make-prob` *type init rest* [Function]
Create a `Prob` object.

`ly:make-scale` *steps* [Function]
Create a scale. Takes a vector of integers as argument.

`ly:make-score` *music* [Function]
Return score with *music* encapsulated in *score*.

`ly:make-simple-closure` *expr* [Function]
Make a simple closure. *expr* should be form of `(func a1 A2 ...)`, and will be invoked as `(func delayed-arg a1 a2 ...)`.

`ly:make-stencil` *expr xext yext* [Function]
Stencils are device independent output expressions. They carry two pieces of information:

1. A specification of how to print this object. This specification is processed by the output backends, for example '`scm/output-ps.scm`'.
2. The vertical and horizontal extents of the object, given as pairs. If an extent is unspecified (or if you use `(1000 . -1000)` as its value), it is taken to be empty.

`ly:make-stream-event` *cl proplist* [Function]
Create a stream event of class *cl* with the given mutable property list.

`ly:message` *str rest* [Function]
A Scheme callable function to issue the message *str*. The message is formatted with `format` and *rest*.

`ly:minimal-breaking` *pb* [Function]
Break (pages and lines) the `Paper_book` object *pb*without looking for optimal spacing: stack as many lines ona page before moving to the next one.

`ly:mm` *num* [Function]
num mm.

`ly:module->alist` *mod* [Function]
Dump the contents of module *mod* as an alist.

`ly:module-copy` *dest src* [Function]
Copy all bindings from module *src* into *dest*.

`ly:modules-lookup` *modules sym def* [Function]
Look up *sym* in the list *modules*, returning the first occurence. If not found, return *def* or `#f` if *def* isn't specified.

`ly:moment-add` *a b* [Function]
Add two moments.

`ly:moment-div` *a b* [Function]
Divide two moments.

`ly:moment-grace-denominator` *mom* [Function]
Extract denominator from grace timing.

`ly:moment-grace-numerator` *mom* [Function]
Extract numerator from grace timing.

`ly:moment-main-denominator` *mom* [Function]
Extract denominator from main timing.

`ly:moment-main-numerator` *mom* [Function]
Extract numerator from main timing.

`ly:moment-mod` *a b* [Function]
Modulo of two moments.

`ly:moment-mul` *a b* [Function]
Multiply two moments.

`ly:moment-sub` *a b* [Function]
Subtract two moments.

`ly:moment<?` *a b* [Function]
Compare two moments.

`ly:moment?` *x* [Function]
Is *x* a `Moment` object?

`ly:music-compress` *m factor* [Function]
Compress music object *m* by moment *factor*.

`ly:music-deep-copy` *m* [Function]
Copy *m* and all sub expressions of *m*.

`ly:music-duration-compress` *mus fact* [Function]
Compress *mus* by factor *fact*, which is a `Moment`.

`ly:music-duration-length` *mus* [Function]
Extract the duration field from *mus* and return the length.

`ly:music-function-extract` *x* [Function]
Return the Scheme function inside *x*.

`ly:music-function?` *x* [Function]
Is *x* a `music-function`?

`ly:music-length` *mus* [Function]
Get the length of music expression *mus* and return it as a `Moment` object.

`ly:music-list?` *lst* [Function]
Type predicate: Return true if *lst* is a list of music objects.

`ly:music-mutable-properties` *mus* [Function]
Return an alist containing the mutable properties of *mus*. The immutable properties are not available, since they are constant and initialized by the `make-music` function.

`ly:music-output?` *x* [Function]
Is *x* a `Music_output` object?

`ly:music-property` *mus sym dfault* [Function]
Get the property *sym* of music expression *mus*. If *sym* is undefined, return `'()`.

`ly:music-set-property!` *mus sym val* [Function]
Set property *sym* in music expression *mus* to *val*.

`ly:music-transpose` *m p* [Function]
Transpose *m* such that central C is mapped to *p*. Return *m*.

`ly:music?` *obj* [Function]
Type predicate.

`ly:note-head::stem-attachment` *font-metric glyph-name* [Function]
Get attachment in *font-metric* for attaching a stem to notehead *glyph-name*.

`ly:number->string` *s* [Function]
Convert *num* to a string without generating many decimals.

`ly:optimal-breaking` *pb* [Function]
Optimally break (pages and lines) the `Paper_book` object *pb* to minimize badness in bother vertical and horizontal spacing.

`ly:option-usage` [Function]
Print `ly:set-option` usage.

`ly:otf->cff` *otf-file-name* [Function]
Convert the contents of an OTF file to a CFF file, returning it as a string.

`ly:otf-font-glyph-info` *font glyph* [Function]
Given the font metric *font* of an OpenType font, return the information about named glyph *glyph* (a string).

`ly:otf-font-table-data` *font tag* [Function]
Extract a table *tag* from *font*. Return empty string for non-existent *tag*.

`ly:otf-font?` *font* [Function]
Is *font* an OpenType font?

`ly:otf-glyph-list` *font* [Function]
Return a list of glyph names for *font*.

`ly:output-def-clone` *def* [Function]
Clone output definition *def*.

`ly:output-def-lookup` *pap sym def* [Function]
Look up *sym* in the *pap* output definition (e.g., `\paper`). Return the value or *def* (which defaults to `'()`) if undefined.

`ly:output-def-parent` *def* [Function]
Get the parent output definition of *def*.

`ly:output-def-scope` *def* [Function]
Get the variable scope inside *def*.

`ly:output-def-set-variable!` *def sym val* [Function]
Set an output definition *def* variable *sym* to *val*.

`ly:output-def?` *def* [Function]
Is *def* a layout definition?

`ly:output-description` *output-def* [Function]
Return the description of translators in *output-def*.

`ly:output-formats` [Function]
Formats passed to '`--format`' as a list of strings, used for the output.

`ly:outputter-close` *outputter* [Function]
Close port of *outputter*.

`ly:outputter-dump-stencil` *outputter stencil* [Function]
Dump stencil *expr* onto *outputter*.

`ly:outputter-dump-string` *outputter str* [Function]
Dump *str* onto *outputter*.

`ly:outputter-output-scheme` *outputter expr* [Function]
Eval *expr* in module of *outputter*.

`ly:outputter-port` *outputter* [Function]
Return output port for *outputter*.

`ly:page-marker?` *x* [Function]
Is *x* a `Page_marker` object?

`ly:page-turn-breaking` *pb* [Function]
Optimally break (pages and lines) the `Paper_book` object *pb* such that page turns only happen in specified places, returning its pages.

`ly:pango-font-physical-fonts` *f* [Function]
Return alist of `(PSNAME . FILENAME)` tuples.

`ly:pango-font?` *f* [Function]
Is *f* a pango font?

`ly:paper-book-pages` *pb* [Function]
Return pages in book *pb*.

`ly:paper-book-paper` *pb* [Function]
Return pages in book *pb*.

`ly:paper-book-performances` *paper-book* [Function]
Return performances in book *paper-book*.

`ly:paper-book-scopes` *book* [Function]
Return pages in layout book *book*.

`ly:paper-book-systems` *pb* [Function]
Return systems in book *pb*.

`ly:paper-book?` *x* [Function]
Is *x* a `Paper_book` object?

`ly:paper-fonts` *bp* [Function]
Return fonts from the `\paper` block *bp*.

`ly:paper-get-font` *paper-smob chain* [Function]
Return a font metric satisfying the font-qualifiers in the alist chain *chain*. (An alist chain is a list of alists, containing grob properties.)

`ly:paper-get-number` *layout-smob name* [Function]
Return the layout variable *name*.

`ly:paper-outputscale` *bp* [Function]
Get output-scale for *bp*.

`ly:paper-score-paper-systems` *paper-score* [Function]
Return vector of `paper_system` objects from *paper-score*.

`ly:paper-system-minimum-distance` *sys1 sys2* [Function]
Measure the minimum distance between these two paper-systems, using their stored skylines if possible and falling back to their extents otherwise.

`ly:paper-system?` *obj* [Function]
Type predicate.

`ly:parse-file` *name* [Function]
Parse a single `.ly` file. Upon failure, throw `ly-file-failed` key.

`ly:parser-clear-error` *parser* [Function]
Clear the error flag for the parser.

`ly:parser-clone` *parser-smob* [Function]
Return a clone of *parser-smob*.

`ly:parser-define!` *parser-smob symbol val* [Function]
Bind *symbol* to *val* in *parser-smob*'s module.

`ly:parser-error` *parser msg input* [Function]
Display an error message and make the parser fail.

`ly:parser-has-error?` *parser* [Function]
Does *parser* have an error flag?

`ly:parser-lexer` *parser-smob* [Function]
Return the lexer for *parser-smob*.

`ly:parser-lookup` *parser-smob symbol* [Function]
Look up *symbol* in *parser-smob*'s module. Return `'()` if not defined.

`ly:parser-output-name` *parser* [Function]
Return the base name of the output file.

`ly:parser-parse-string` *parser-smob ly-code* [Function]
Parse the string *ly-code* with *parser-smob*. Upon failure, throw `ly-file-failed` key.

`ly:parser-set-note-names` *parser names* [Function]
Replace current note names in *parser*. *names* is an alist of symbols. This only has effect if the current mode is notes.

`ly:performance-write` *performance filename* [Function]
Write *performance* to *filename*.

`ly:pfb->pfa` *pfb-file-name* [Function]
Convert the contents of a PFB file to PFA.

`ly:pitch-alteration` *pp* [Function]
Extract the alteration from pitch *pp*.

`ly:pitch-diff` *pitch root* [Function]
Return pitch *delta* such that *pitch* transposed by *delta* equals *root*.

`ly:pitch-negate` *p* [Function]
Negate *p*.

`ly:pitch-notename` *pp* [Function]
Extract the note name from pitch *pp*.

`ly:pitch-octave` *pp* [Function]
Extract the octave from pitch *pp*.

`ly:pitch-quartertones` *pp* [Function]
Calculate the number of quarter tones of *pp* from middle C.

`ly:pitch-semitones` *pp* [Function]
Calculate the number of semitones of *pp* from middle C.

`ly:pitch-steps` *p* [Function]
Number of steps counted from middle C of the pitch *p*.

`ly:pitch-transpose` *p delta* [Function]
Transpose *p* by the amount *delta*, where *delta* is relative to middle C.

`ly:pitch<?` *p1 p2* [Function]
Is *p1* lexicographically smaller than *p2*?

`ly:pitch?` *x* [Function]
Is *x* a `Pitch` object?

`ly:prob-property` *obj sym dfault* [Function]
Return the value for *sym*.

`ly:prob-property?` *obj sym* [Function]
Is boolean prop *sym* set?

`ly:prob-set-property!` *obj sym value* [Function]
Set property *sym* of *obj* to *value*.

`ly:prob-type?` *obj type* [Function]
Is *obj* the specified prob-type?

`ly:prob?` *x* [Function]
Is *x* a `Prob` object?

`ly:programming-error` *str rest* [Function]
A Scheme callable function to issue the internal warning *str*. The message is formatted with `format` and *rest*.

`ly:progress` *str rest* [Function]
A Scheme callable function to print progress *str*. The message is formatted with `format` and *rest*.

`ly:property-lookup-stats` *sym* [Function]
Return hash table with a property access corresponding to *sym*. Choices are `prob`, `grob`, and `context`.

`ly:protects` [Function]
Return hash of protected objects.

`ly:pt` *num* [Function]
num printer points.

`ly:register-stencil-expression` *symbol* [Function]
Add *symbol* as head of a stencil expression.

`ly:relative-group-extent` *elements common axis* [Function]
Determine the extent of *elements* relative to *common* in the *axis* direction.

`ly:reset-all-fonts` [Function]
Forget all about previously loaded fonts.

`ly:round-filled-box` *xext yext blot* [Function]
Make a `Stencil` object that prints a black box of dimensions *xext*, *yext* and roundness *blot*.

`ly:run-translator` *mus output-def* [Function]
Process *mus* according to *output-def*. An interpretation context is set up, and *mus* is interpreted with it. The context is returned in its final state.

Optionally, this routine takes an object-key to to uniquely identify the score block containing it.

`ly:score-add-output-def!` *score def* [Function]
Add an output definition *def* to *score*.

`ly:score-embedded-format` *score layout* [Function]
Run *score* through *layout* (an output definition) scaled to correct output-scale already, returning a list of layout-lines. This function takes an optional `Object_key` argument.

`ly:score-error?` *score* [Function]
Was there an error in the score?

`ly:score-header` *score* [Function]
Return score header.

`ly:score-music` *score* [Function]
Return score music.

`ly:score-output-defs` *score* [Function]
All output definitions in a score.

`ly:score?` *x* [Function]
Is *x* a `Score` object?

`ly:set-default-scale` *scale* [Function]
Set the global default scale.

`ly:set-grob-modification-callback` *cb* [Function]
Specify a procedure that will be called every time LilyPond modifies a grob property. The callback will receive as arguments the grob that is being modified, the name of the C++ file in which the modification was requested, the line number in the C++ file in which the modification was requested, the name of the function in which the modification was requested, the property to be changed, and the new value for the property.

`ly:set-middle-C!` *context* [Function]
Set the `middleCPosition` variable in *context* based on the variables `middleCClefPosition` and middleCOffset.

`ly:set-option` *var val* [Function]
Set a program option.

`ly:set-point-and-click` *what* [Function]
Deprecated.

`ly:set-property-cache-callback` *cb* [Function]
Specify a procedure that will be called whenever lilypond calculates a callback function and caches the result. The callback will receive as arguments the grob whose property it is, the name of the property, the name of the callback that calculated the property, and the new (cached) value of the property.

`ly:simple-closure?` *clos* [Function]
Type predicate.

`ly:skyline-pair?` *x* [Function]
Is *x* a `Skyline_pair` object?

`ly:skyline?` *x* [Function]
Is *x* a `Skyline` object?

`ly:smob-protects` [Function]
Return LilyPond's internal smob protection list.

`ly:solve-spring-rod-problem` *springs rods length ragged* [Function]
Solve a spring and rod problem for *count* objects, that are connected by *count*-1 *springs*, and an arbitrary number of *rods*. *count* is implicitly given by *springs* and *rods*. The *springs* argument has the format `(ideal, inverse_hook)` and *rods* is of the form `(idx1, idx2, distance)`.

length is a number, *ragged* a boolean.

The function returns a list containing the force (positive for stretching, negative for compressing and `#f` for non-satisfied constraints) followed by *spring-count*+1 positions of the objects.

`ly:source-file?` *x* [Function]
Is *x* a `Source_file` object?

`ly:spanner-bound` *slur dir* [Function]
Get one of the bounds of *slur*. *dir* is `-1` for left, and `1` for right.

`ly:spanner-broken-into` *spanner* [Function]
Return broken-into list for *spanner*.

`ly:spanner?` *g* [Function]
Is *g* a spanner object?

`ly:start-environment` [Function]
Return the environment (a list of strings) that was in effect at program start.

`ly:stderr-redirect` *file-name mode* [Function]
Redirect stderr to *file-name*, opened with *mode*.

`ly:stencil-add` *args* [Function]
Combine stencils. Takes any number of arguments.

`ly:stencil-aligned-to` *stil axis dir* [Function]
Align *stil* using its own extents. *dir* is a number. `-1` and `1` are left and right, respectively. Other values are interpolated (so `0` means the center).

`ly:stencil-combine-at-edge` *first axis direction second padding minimum* [Function]
Construct a stencil by putting *second* next to *first*. *axis* can be 0 (x-axis) or 1 (y-axis). *direction* can be -1 (left or down) or 1 (right or up). The stencils are juxtaposed with *padding* as extra space. If this puts the reference points closer than *minimum*, they are moved by the latter amount. *first* and *second* may also be `'()` or `#f`.

`ly:stencil-empty?` *stil* [Function]
Return whether *stil* is empty.

`ly:stencil-expr` *stil* [Function]
Return the expression of *stil*.

`ly:stencil-extent` *stil axis* [Function]
Return a pair of numbers signifying the extent of *stil* in *axis* direction (`0` or `1` for x and y axis, respectively).

`ly:stencil-fonts` *s* [Function]
Analyze *s*, and return a list of fonts used in *s*.

`ly:stencil-in-color` *stc r g b* [Function]
Put *stc* in a different color.

`ly:stencil-rotate` *stil angle x y* [Function]
Return a stencil *stil* rotated *angle* degrees around point (*x*, *y*).

`ly:stencil-translate` *stil offset* [Function]
Return a *stil*, but translated by *offset* (a pair of numbers).

`ly:stencil-translate-axis` *stil amount axis* [Function]
Return a copy of *stil* but translated by *amount* in *axis* direction.

`ly:stencil?` *x* [Function]
Is *x* a `Stencil` object?

`ly:stream-event?` *x* [Function]
Is *x* a `Stream_event` object?

`ly:string-substitute` *a b s* [Function]
Replace string *a* by string *b* in string *s*.

`ly:system-print` *system* [Function]
Draw the system and return the prob containing its stencil.

`ly:system-stretch` *system amount-scm* [Function]
Stretch the system vertically by the given amount. This must be called before the system is drawn (for example with `ly:system-print`).

`ly:text-dimension` *font text* [Function]
Given the font metric in *font* and the string *text*, compute the extents of that text in that font. The return value is a pair of number-pairs.

`ly:text-interface::interpret-markup` [Function]
Convert a text markup into a stencil. Takes three arguments, *layout*, *props*, and *markup*.

layout is a `\layout` block; it may be obtained from a grob with `ly:grob-layout`. *props* is a alist chain, ie. a list of alists. This is typically obtained with `(ly:grob-alist-chain (ly:layout-lookup layout 'text-font-defaults))`. *markup* is the markup text to be processed.

`ly:translator-description` *me* [Function]
Return an alist of properties of translator *me*.

`ly:translator-group?` *x* [Function]
Is *x* a `Translator_group` object?

`ly:translator-name` *trans* [Function]
Return the type name of the translator object *trans*. The name is a symbol.

`ly:translator?` *x* [Function]
Is *x* a `Translator` object?

`ly:transpose-key-alist` *l pit* [Function]
Make a new key alist of *l* transposed by pitch *pit*.

`ly:truncate-list!` *lst i* [Function]
Take at most the first *i* of list *lst*.

`ly:ttf->pfa` *ttf-file-name* [Function]
Convert the contents of a TTF file to Type42 PFA, returning it as a string.

`ly:ttf-ps-name` *ttf-file-name* [Function]
Extract the PostScript name from a TrueType font.

`ly:unit` [Function]
Return the unit used for lengths as a string.

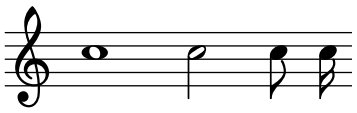
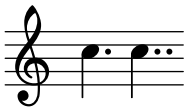
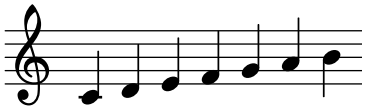
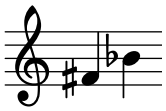
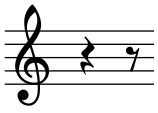
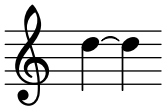
`ly:usage` [Function]
Print usage message.

`ly:version` [Function]
Return the current lilypond version as a list, e.g., `(1 3 127 uu1)`.

`ly:warning` *str rest* [Function]
A Scheme callable function to issue the warning `str`. The message is formatted with `format` and `rest`.

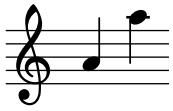
`ly:wide-char->utf-8` *wc* [Function]
Encode the Unicode codepoint *wc*, an integer, as UTF-8.

Appendix C Cheat sheet

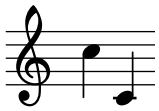
Syntax	Description	Example
`1 2 8 16`	durations	
`c4. c4..`	augmentation dots	
`c d e f g a b`	scale	
`fis bes`	alteration	
`\clef treble \clef bass`	clefs	
`\time 3/4 \time 4/4`	time signature	
`r4 r8`	rest	
`d ~ d`	tie	

`\key es \major` — key signature

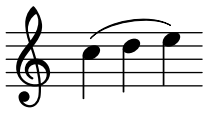
note`'` — raise octave



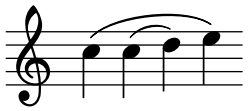
note`,` — lower octave



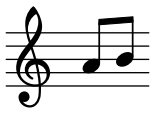
`c( d e)` — slur



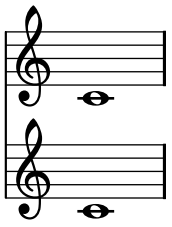
`c\( c( d) e\)` — phrasing slur



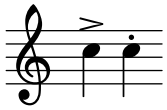
`a8[ b]` — beam



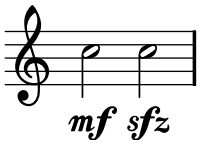
`<< \new Staff ... >>` — more staves



`c-> c-.` — articulations

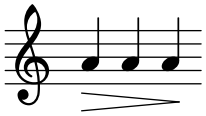


`c2\mf c\sfz` dynamics

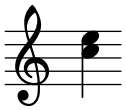


`a\< a a\!` crescendo

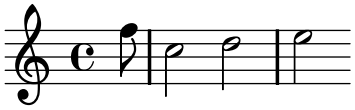
`a\> a a\!` decrescendo



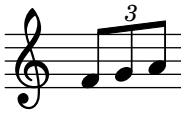
`< >` chord



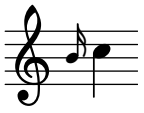
`\partial 8` upstep



`\times 2/3 {f g a}` triplets



`\grace` grace notes

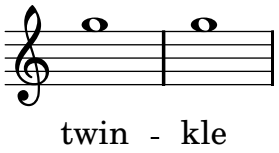


`\lyricmode { twinkle }` entering lyrics twinkle

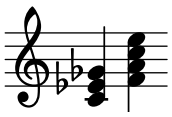
`\new Lyrics` printing lyrics

twinkle

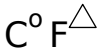
`twin -- kle` lyric hyphen



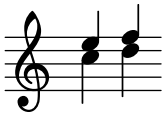
`\chordmode { c:dim f:maj7 }` chords



`\context ChordNames` printing chord names



`<<{e f} \\ {c d}>>` polyphony



`s4 s8 s16` spacer rests

Appendix D GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of 'copyleft', which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The 'Document', below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as 'you'.

A 'Modified Version' of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A 'Secondary Section' is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The 'Invariant Sections' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The 'Cover Texts' are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A 'Transparent' copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file

format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not 'Transparent' is called 'Opaque'.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The 'Title Page' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, 'Title Page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled 'History', and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled 'History' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the 'History' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled 'Acknowledgments' or 'Dedications', preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled 'Endorsements'. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as 'Endorsements' or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to

the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled 'Endorsements', provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled 'History' in the various original documents, forming one section entitled 'History'; likewise combine any sections entitled 'Acknowledgments', and any sections entitled 'Dedications'. You must delete all sections entitled 'Endorsements.'

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an 'aggregate', and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License 'or any later version' applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

D.0.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
  Copyright (C)  year  your name.
  Permission is granted to copy, distribute and/or modify this document
  under the terms of the GNU Free Documentation License, Version 1.1
  or any later version published by the Free Software Foundation;
  with the Invariant Sections being list their titles, with the
  Front-Cover Texts being list, and with the Back-Cover Texts being list.
  A copy of the license is included in the section entitled ‘GNU
Free Documentation License’
.
```

If you have no Invariant Sections, write ‘with no Invariant Sections’ instead of saying which ones are invariant. If you have no Front-Cover Texts, write ‘no Front-Cover Texts’ instead of ‘Front-Cover Texts being *list*’; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix E LilyPond command index

This index does blah blah blah.

!

! 5

#

#(set-accidental-style 'piano-cautionary) ... 22

'

' 1

(

(begin * * * *) 51
(end * * * *) 51

,

, ... 1

.

. 30

/

/ 148
/+ 148

=

= 7

?

? 5

[

[..... 54

]

] 54

_

_ 130, 133

\

\! 70
\< 70
\> 70
\accepts 258
\addlyrics 132
\aeolian 15
\afterGrace 64
\aikenHeads 27
\allowPageTurn 218
\alternative 80
\applyContext 282
\applyOutput 283
\arpeggio 76
\arpeggioBracket 76
\arpeggioDown 76
\arpeggioNeutral 76
\arpeggioUp 76
\ascendens 183
\auctum 183
\augmentum 183
\autoBeamOff 53
\autoBeamOn 53
\bar 55
\book 191
\break 216
\breve 30
\breve 37
\cadenzaOff 45
\cadenzaOn 45
\caesura 175
\cavum 183
\chordmode 4, 11
\clef 11
\compressMusic 34
\compressMusic 47
\context 251
\deminutum 183
\denies 258
\descendens 183
\displayLilyMusic 195, 274
\displayMusic 274
\divisioMaior 175
\divisioMaxima 175
\divisioMinima 175
\dorian 15
\dotsDown 31
\dotsNeutral 31
\dotsUp 31
\dynamicDown 72
\dynamicNeutral 72
\dynamicUp 72
\easyHeadsOff 27
\easyHeadsOn 27
\f 70
\featherDurations 54
\ff 70
\fff 70
\ffff 70
\finalis 175
\flexa 183
\fp 70
\frenchChords 151
\germanChords 151
\glissando 75
\grace 63
\hideNotes 107
\hideStaffSwitch 155

\improvisationOff ... 29
\improvisationOn ... 29
\inclinatum ... 183
\include ... 192
\ionian ... 15
\italianChords ... 151
\key ... 14, 27
\label ... 203
\laissezVibrer ... 36
\layout ... 214
\linea ... 183
\locrian ... 15
\longa ... 30
\longa ... 37
\lydian ... 15
\lyricmode ... 130, 132
\lyricsto ... 132
\major ... 15
\mark ... 62, 116
\markuplines ... 123
\maxima ... 30
\maxima ... 37
\melisma ... 135
\melismaEnd ... 135
\mf ... 70
\minor ... 15
\mixolydian ... 15
\mp ... 70
\new ... 250
\noBreak ... 216
\noPageBreak ... 217
\noPageTurn ... 218
\normalsize ... 105
\octave ... 7
\oneVoice ... 88
\oriscus ... 183
\override ... 259
\p ... 70
\page-ref ... 203
\pageBreak ... 217
\pageTurn ... 218
\paper ... 208
\partial ... 44
\pes ... 183
\phrasingSlurDown ... 74
\phrasingSlurNeutral ... 74
\phrasingSlurUp ... 74
\phrygian ... 15
\pp ... 70
\ppp ... 70
\pppp ... 70
\property in \lyricmode ... 130
\quilisma ... 183
\relative ... 2, 4, 11
\repeat ... 80
\repeatTie ... 35, 81
\rest ... 37
\rfz ... 70
\sacredHarpHeads ... 27
\semiGermanChords ... 151
\set ... 251
\sf ... 70
\sff ... 70
\sfz ... 70
\shiftOff ... 88
\shiftOn ... 88
\shiftOnn ... 88
\shiftOnnn ... 88
\showStaffSwitch ... 155
\skip ... 38
\slurDashed ... 73
\slurDotted ... 73
\slurDown ... 73
\slurNeutral ... 73
\slurSolid ... 73
\slurUp ... 73
\small ... 105
\sp ... 70
\spp ... 70
\startTrillSpan ... 79
\stemDown ... 109
\stemNeutral ... 109
\stemUp ... 109
\stopTrillSpan ... 79
\stropha ... 183
\table-of-contents ... 204
\tag ... 193
\tempo ... 98
\textLengthOff ... 115
\textLengthOn ... 115
\tieDashed ... 36
\tieDotted ... 36
\tieDown ... 36
\tieNeutral ... 36
\tieSolid ... 36
\tieUp ... 36
\time ... 42
\times ... 31
\tiny ... 105
\tocItem ... 204
\transpose ... 4, 8, 11
\transposition ... 17
\tupletDown ... 32
\tupletNeutral ... 32
\tupletUp ... 32
\tweak ... 262
\unfoldRepeats ... 206
\unHideNotes ... 107
\unset ... 252
\virga ... 183
\virgula ... 175
\voiceFour ... 88
\voiceFourStyle ... 88
\voiceNeutralStyle ... 88
\voiceOne ... 88
\voiceOneStyle ... 88
\voiceThree ... 88
\voiceThreeStyle ... 88
\voiceTwo ... 88
\voiceTwoStyle ... 88
\with ... 253

|

| ... 61

~

~ ... 35

A

after-title-space ... 210
annotate-spacing ... 214
arranger ... 198
aug ... 148
auto-first-page-number ... 211
autoBeaming ... 53
autoBeamSettings ... 51

B

barCheckSynchronize ... 61
barNumberVisibility ... 58
base-shortest-duration ... 235
before-title-space ... 210
between-system-padding ... 209
between-system-space ... 209
between-title-space ... 210
blank-last-page-force ... 211
blank-page-force ... 211
bookTitleMarkup ... 201
bottom-margin ... 209
breakable ... 50
breakbefore ... 198

C

chordNameExceptions ... 150
chordNameSeparator ... 150
chordNoteNamer ... 150
chordPrefixSpacer ... 150
chordRootNamer ... 150
common-shortest-duration ... 235
composer ... 198
copyright ... 198
currentBarNumber ... 58, 67

D

dedication ... 198
defaultBarType ... 57
dim ... 148

E

evenFooterMarkup ... 201
evenHeaderMarkup ... 201

F

first-page-number ... 208
followVoice ... 154
font-interface ... 123
foot-separation ... 209
forget accidental style ... 23

H

head-separation ... 209
horizontal-shift ... 210

I

indent ... 238
instrument ... 198

L

layout file ... 213
left-margin ... 209
line-width ... 209, 238
ly:minimal-breaking ... 218
ly:optimal-breaking ... 217
ly:page-turn-breaking ... 217

M

m ... 148
maj ... 148
majorSevenSymbol ... 150
measureLength ... 67
measurePosition ... 67
meter ... 198
minimumFret ... 160
minimumPageTurnLength ... 217
minimumRepeatLengthForPageTurn ... 218
modern style accidentals ... 21
modern-cautionary ... 21
modern-voice ... 22
modern-voice-cautionary ... 22

N

no-reset accidental style ... 23

O

oddFooterMarkup ... 201
oddHeaderMarkup ... 201
opus ... 198
outside-staff-horizontal-padding ... 233
outside-staff-padding ... 233
outside-staff-priority ... 233

P

page-breaking-between-system-padding ... 210
page-spacing-weight ... 211
page-top-space ... 209
paper-height ... 208
paper-width ... 208
papersize ... 208
piano accidentals ... 22
piece ... 198
pipeSymbol ... 61
poet ... 198
print-first-page-number ... 208
print-page-number ... 208
printallheaders ... 200, 210

R

r ... 37
R ... 39
ragged-bottom ... 209

ragged-last 238
ragged-last-bottom 209
ragged-right 238
repeatCommands 82

S

s 38
scoreTitleMarkup 201
set-accidental-style 19
shapeNoteStyles 27
showLastLength 195
skipTypesetting 195
spacing 235
Staff.midiInstrument 206
stem-spacing-correction 235
stemLeftBeamCount 54
stemRightBeamCount 54
subdivideBeams 50
subsubtitle 198
subtitle 198
suggestAccidentals 185
sus 148
system-count 209
systemSeparatorMarkup 210

T

tagline 198
textSpannerDown 116
textSpannerNeutral 116
textSpannerUp 116
title 198
top-margin 209
tremoloFlags 84
TupletBracket 32
tupletFullLength 33
tupletFullLengthNote 33
TupletNumber 32
tupletNumberFormatFunction 32
tupletSpannerDuration 32

W

whichBar 57

Appendix F LilyPond index

In contrast, this index does blaz blaz blaz.

!

! 5

#

#(set-accidental-style 'piano-cautionary) ... 22

'

' 1

(

(begin * * * *) 51
(end * * * *) 51

,

, ... 1

.

. 30

/

/ 148
/+ 148

=

= 7

?

? 5

[

[..... 54

]

] 54

_

_ 130, 133

\

\! 70
\< 70
\> 70
\accepts 258
\addlyrics 129
\addlyrics 132
\aeolian 15
\afterGrace 64
\aikenHeads 27
\allowPageTurn 218
\alternative 80
\applyContext 282
\applyOutput 283
\arpeggio 76
\arpeggioBracket 76
\arpeggioDown 76
\arpeggioNeutral 76
\arpeggioUp 76
\ascendens 183
\auctum 183
\augmentum 183
\autoBeamOff 53
\autoBeamOn 53
\bar 55
\book 191
\break 216
\breve 30
\breve 37
\cadenzaOff 45
\cadenzaOn 45
\caesura 175
\cavum 183
\chordmode 4, 11
\clef 11
\compressMusic 34
\compressMusic 47
\context 251
\deminutum 183
\denies 258
\descendens 183
\displayLilyMusic 195, 274
\displayMusic 274
\divisioMaior 175
\divisioMaxima 175
\divisioMinima 175
\dorian 15
\dotsDown 31
\dotsNeutral 31
\dotsUp 31
\dynamicDown 72
\dynamicNeutral 72
\dynamicUp 72
\easyHeadsOff 27
\easyHeadsOn 27
\f 70
\featherDurations 54
\ff 70
\fff 70
\ffff 70
\finalis 175
\flexa 183
\fp 70
\frenchChords 151
\germanChords 151
\glissando 75
\grace 63

\hideNotes ... 107
\hideStaffSwitch ... 155
\improvisationOff ... 29
\improvisationOn ... 29
\inclinatum ... 183
\include ... 192
\ionian ... 15
\italianChords ... 151
\key ... 14, 27
\label ... 203
\laissezVibrer ... 36
\layout ... 214
\linea ... 183
\locrian ... 15
\longa ... 30
\longa ... 37
\lydian ... 15
\lyricmode ... 130, 132
\lyricsto ... 132
\major ... 15
\mark ... 62, 116
\markuplines ... 123
\maxima ... 30
\maxima ... 37
\melisma ... 135
\melismaEnd ... 135
\mf ... 70
\minor ... 15
\mixolydian ... 15
\mp ... 70
\new ... 250
\noBreak ... 216
\noPageBreak ... 217
\noPageTurn ... 218
\normalsize ... 105
\octave ... 7
\once ... 253
\oneVoice ... 88
\oriscus ... 183
\override ... 259
\p ... 70
\page-ref ... 203
\pageBreak ... 217
\pageTurn ... 218
\paper ... 201
\paper ... 208
\partial ... 44
\pes ... 183
\phrasingSlurDown ... 74
\phrasingSlurNeutral ... 74
\phrasingSlurUp ... 74
\phrygian ... 15
\pp ... 70
\ppp ... 70
\pppp ... 70
\property in \lyricmode ... 130
\quilisma ... 183
\relative ... 2, 4, 11
\repeat ... 80
\repeatTie ... 35, 81
\rest ... 37
\rfz ... 70
\sacredHarpHeads ... 27
\semiGermanChords ... 151
\set ... 251
\sf ... 70
\sff ... 70
\sfz ... 70
\shiftOff ... 88
\shiftOn ... 88
\shiftOnn ... 88
\shiftOnnn ... 88
\showStaffSwitch ... 155
\skip ... 38
\slurDashed ... 73
\slurDotted ... 73
\slurDown ... 73
\slurNeutral ... 73
\slurSolid ... 73
\slurUp ... 73
\small ... 105
\sp ... 70
\spp ... 70
\startTrillSpan ... 79
\stemDown ... 109
\stemNeutral ... 109
\stemUp ... 109
\stopTrillSpan ... 79
\stropha ... 183
\super ... 313
\table-of-contents ... 204
\tag ... 193
\tempo ... 98
\textLengthOff ... 115
\textLengthOn ... 115
\tieDashed ... 36
\tieDotted ... 36
\tieDown ... 36
\tieNeutral ... 36
\tieSolid ... 36
\tieUp ... 36
\time ... 42
\times ... 31
\tiny ... 105
\tocItem ... 204
\transpose ... 4, 8, 11
\transposition ... 17
\tupletDown ... 32
\tupletNeutral ... 32
\tupletUp ... 32
\tweak ... 262
\unfoldRepeats ... 206
\unHideNotes ... 107
\unset ... 252
\virga ... 183
\virgula ... 175
\voiceFour ... 88
\voiceFourStyle ... 88
\voiceNeutralStyle ... 88
\voiceOne ... 88
\voiceOneStyle ... 88
\voiceThree ... 88
\voiceThreeStyle ... 88
\voiceTwo ... 88
\voiceTwoStyle ... 88
\with ... 253

|

| 61

~

~ 35

1

15ma 16

8

8va 16
8ve 16

A

a due 90
absolute 1
absolute octave specification 1
accent 69, 314
acciaccatura 63
acciaccatura 66
acciaccatura 270, 338
accidental 4
Accidental 24
Accidental 168
accidental style 19
accidental style, cautionary, modern voice 22
accidental style, default 19
accidental style, forget 23
accidental style, modern 22
accidental style, modern voice cautionary 22
accidental style, no reset 23
accidental style, piano 22
accidental style, piano cautionary 22
accidental style, voice, modern cautionary 22
accidental, cautionary 5
Accidental, musica ficta 185
accidental, parenthesized 5
accidental, reminder 5
Accidental_engraver 24
Accidental_engraver 185
AccidentalPlacement 24
accidentals 19, 168
accidentals and simultaneous notes 24
accidentals in chords 24
accidentals, automatic 19
accidentals, modern 22
accidentals, modern cautionary style 21
accidentals, modern style 21
accidentals, multivoice 22
accidentals, piano 22
accidentals, piano cautionary 22
AccidentalSuggestion 24
AccidentalSuggestion 185
addInstrumentDefinition 271, 339
additions, in chords 147
addQuote 270, 339
adjusting staff symbol 95
after-title-space 210
afterGrace 271, 340
Aiken shape note heads 27
al niente 71
alignAboveContext 259
alignBelowContext 259
aligning to cadenza 66
All layout objects 256
allowPageTurn 271, 340
alto clef 11
ambit 24
ambitus 24
ambitus 26
Ambitus 26
Ambitus_engraver 26
AmbitusAccidental 26
AmbitusLine 26
AmbitusNoteHead 26
anacrusis 44
ancient clef 11
annotate-spacing 214
applyContext 270, 338
applyMusic 271, 339
applyOutput 271, 340
appoggiatura 63
appoggiatura 66
appoggiatura 270, 338
arpeggio 76
Arpeggio 78
arranger 198
arrow-head 306
arrow-head-markup 306
articulations 68, 173
artificial harmonics 165
assertBeamQuant 270, 339
assertBeamSlope 271, 340
aug 148
auto-first-page-number 211
auto-knee-gap 50
autobeam 53
autoBeaming 53
autoBeamOff 49
autoBeamOn 49
autoBeamSettings 51
autochange 269, 338
AutoChangeMusic 152
automatic accidentals 19
automatic beam generation 53
automatic beams, tuning 51
automatic part combining 89
Automatic staff changes 152
automatic syllable durations 132
Axis_group_engraver 224

B

Backend 256, 260
balloon 110
balloonGrobText 272, 340
balloonText 270, 339
Banjo tablatures 162
Banter 151
bar 271, 339
bar check 61
bar lines 55
bar lines, invisible 56
bar lines, symbols on 116

bar number ... 67
bar number alignment ... 60
bar number, format ... 59
bar numbers ... 58
bar numbers, regular spacing ... 58
bar-line-interface ... 325
Bar_engraver ... 149
barCheckSynchronize ... 61
baritone clef ... 11
BarLine ... 58
BarNumber ... 60
BarNumber ... 61
barNumberCheck ... 270, 339
barNumberVisibility ... 58
base-shortest-duration ... 235
bass clef ... 11
BassFigure ... 186
BassFigureAlignment ... 188
BassFigureBracket ... 188
BassFigureContinuation ... 188
BassFigureLine ... 188
Basso continuo ... 185
beam ... 306
Beam ... 51, 84
beam-markup ... 306
beams and line breaks ... 50
beams, feathered ... 54
beams, kneed ... 50
beams, manual ... 49, 54
beats per minute ... 98
before-title-space ... 210
beginners' music ... 26
bendAfter ... 270, 339
between-system-padding ... 209
between-system-space ... 209
between-title-space ... 210
bigger ... 306
bigger-markup ... 306
blank staff paper ... 111
blank-last-page-force ... 211
blank-page-force ... 211
bold ... 306
bold-markup ... 306
bookTitleMarkup ... 201
bottom-margin ... 209
box ... 306
box-markup ... 306
brace ... 95
brace, vertical ... 93
bracket ... 95
bracket ... 306
bracket length, tuplets ... 33
bracket, tuplet ... 32
bracket, vertical ... 93
bracket-markup ... 306
brackets ... 112
break, line ... 50
breakable ... 50
breakbefore ... 198
breaking lines ... 215
breaking pages ... 238
breathe ... 270, 338
BreathingSign ... 75, 175
breve ... 31
breve ... 37
broken chord ... 76

C

C clef ... 11
cadenza ... 45, 66
cadenza, aligning to ... 66
calling code during interpreting ... 282
calling code on layout objects ... 283
caps ... 306
caps-markup ... 306
cautionary accidental ... 5
cautionary accidental style, piano ... 22
cautionary accidentals, piano ... 22
center-align ... 306
center-align-markup ... 306
changing properties ... 251
char ... 306
char-markup ... 306
ChoirStaff ... 95
choral score ... 135
choral tenor clef ... 12
chord ... 86
chord diagrams ... 163
chord entry ... 146
chord mode ... 146
chord names ... 145, 146, 148
chordNameExceptions ... 150
ChordNames ... 98
ChordNames ... 148, 149
chordNameSeparator ... 150
chordNoteNamer ... 150
chordPrefixSpacer ... 150
chordRootNamer ... 150
chords ... 145, 148
Chords ... 86
Chords mode ... 146
chords, accidentals in ... 24
chords, fingering ... 106
chords, jazz ... 151
church mode ... 16
church modes ... 15
church rest ... 40
circle ... 306
circle-markup ... 306
clef ... 4, 11
clef ... 272, 341
Clef ... 14
clef, alto ... 11
clef, ancient ... 11
clef, baritone ... 11
clef, bass ... 11
clef, C ... 11
clef, F ... 11
clef, french ... 11
clef, G ... 11
clef, mezzosoprano ... 11
clef, soprano ... 11
clef, tenor ... 11
clef, transposing ... 12
clef, treble ... 11
clef, varbaritone ... 11
clef, violin ... 11
clefs ... 169
cluster ... 86

Cluster_spanner_engraver ... 86
clusters ... 148
ClusterSpanner ... 86
ClusterSpannerBeacon ... 86
coda ... 63, 69, 314
coda on bar line ... 116
colored notes ... 107
colored objects ... 107
coloring notes ... 107
coloring objects ... 107
colors ... 107
Colors, list of ... 288
column ... 306
column-lines ... 314
column-lines-markup-list ... 314
column-markup ... 306
combine ... 307
combine-markup ... 307
combining parts ... 89
common-shortest-duration ... 235
Completion_heads_engraver ... 48
composer ... 198
compressing music ... 34
compressMusic ... 271, 340
concat ... 307
concat-markup ... 307
concert pitch ... 18
condensing rests ... 42
context definition ... 206
Context, creating ... 250
copyright ... 198
copyright ... 201
creating contexts ... 251
crescendo ... 71
crescendo ... 73
cross note heads ... 26
cross staff ... 154
cross staff stem ... 151
cue notes ... 101, 102
cue notes, formatting ... 102
cueDuring ... 270, 338
cues ... 101, 102
cues, formatting ... 102
currentBarNumber ... 58, 67
custodes ... 174
custos ... 174
Custos ... 175
Custos_engraver ... 174

D

D.S al Fine ... 63
decrescendo ... 71
decrescendo ... 73
dedication ... 198
default accidental style ... 19
defaultBarType ... 57
defining markup commands ... 279
diamond note heads ... 26
dim ... 148
diminuendo ... 71
dir-column ... 307
dir-column-markup ... 307
dispatcher ... 341
displayLilyMusic ... 271, 339
displayMusic ... 271, 340
distance between staves ... 222
distance between staves in piano music ... 151
divisio ... 175
divisiones ... 175
doits ... 76
DotColumn ... 31
Dots ... 31
dotted notes ... 30
double flat ... 4
double flat ... 6
double sharp ... 4
double sharp ... 6
double time signatures ... 45
doubleflat ... 307
doubleflat-markup ... 307
DoublePercentRepeat ... 85
DoublePercentRepeatCounter ... 85
doublesharp ... 307
doublesharp-markup ... 307
downbow ... 69, 314
draw-circle ... 307
draw-circle-markup ... 307
draw-line ... 307
draw-line-markup ... 307
drums ... 156
DrumStaff ... 96, 156, 159
DrumVoice ... 156, 157, 159
Duration names notes and rests ... 31
durations, of notes ... 30
dynamic ... 307
dynamic-markup ... 307
DynamicLineSpanner ... 72, 73
dynamics ... 70
Dynamics, editorial ... 125
Dynamics, parenthesis ... 125
DynamicText ... 73

E

easy notation ... 26
easy play note heads ... 26
endSpanners ... 269, 338
Engraver_group ... 258
epsfile ... 307
epsfile-markup ... 307
espressivo ... 69, 314
espressivo, articulation ... 71
evenFooterMarkup ... 201
evenHeaderMarkup ... 201
Event ... 263
exceptions, chord names ... 150
expanding repeats ... 206
extender ... 135

F

F clef ... 11
falls ... 76
FDL, GNU Free Documentation License ... 363
featherDurations ... 270, 339
fermata ... 69, 314
fermata on bar line ... 116

fermata on multi-measure rest ... 40
Feta font ... 290
fifth ... 4
FiguredBass ... 98
FiguredBass ... 186
FiguredBass ... 187
FiguredBass ... 188
fill-line ... 307
fill-line-markup ... 307
filled-box ... 307
filled-box-markup ... 307
finalis ... 175
finding graphical objects ... 259
finger ... 307
finger change ... 106
finger-interface ... 261
finger-markup ... 307
FingerEvent ... 260
fingering ... 106
Fingering ... 107, 260, 261
fingering chords ... 106
fingering-event ... 260
Fingering_engraver ... 260, 262
fingerings, right hand, for guitar ... 164
first-page-number ... 208
flageolet ... 69, 314
flags ... 171
flat ... 4
flat ... 6
flat ... 307
flat, double ... 4
flat-markup ... 307
follow voice ... 154
followVoice ... 154
font families, setting ... 124
font magnification ... 123, 124
font selection ... 123
font size ... 105, 124
font size scaling ... 105
font size, selecting ... 105
font size, setting ... 213
font size, standard ... 105
font switching ... 120
Font, Feta ... 290
font-interface ... 105, 123, 261, 311
fontCaps ... 307
fontCaps-markup ... 307
fontsize ... 307
fontsize-markup ... 307
foot marks ... 69, 314
foot-separation ... 209
footer ... 201
footer, page ... 208
Forbid_line_break_engraver ... 48
forget accidental style ... 23
format, rehearsal mark ... 62
four bar music ... 215
fraction ... 307
fraction-markup ... 307
fragments ... 101, 102
french clef ... 11
Frenched scores ... 97
Frenched staff ... 96
Frenched staves ... 96
fret ... 160
fret diagrams ... 163
fret-diagram ... 307
fret-diagram-interface ... 163
fret-diagram-markup ... 307
fret-diagram-terse ... 308
fret-diagram-terse-markup ... 308
fret-diagram-verbose ... 308
fret-diagram-verbose-markup ... 308
fromproperty ... 309
fromproperty-markup ... 309
full measure rests ... 39

G

G clef ... 11
general-align ... 309
general-align-markup ... 309
ghost notes ... 109
glissando ... 75
Glissando ... 76, 128
grace ... 272, 340
grace notes ... 63
grace notes ... 66
grace notes, following ... 64
GraceMusic ... 66, 274
grand staff ... 93
GrandStaff ... 24
GrandStaff ... 95
graphical object descriptions ... 259
Gregorian square neumes ligatures ... 177
Gregorian_ligature_engraver ... 168
grob ... 260
grob-interface ... 260, 261
guitar note heads ... 26
guitar tablature ... 160

H

hairpin ... 71
hairpin ... 73
Hairpin ... 71, 73
Hal Leonard ... 26
halign ... 309
halign-markup ... 309
harmonic note heads ... 26
hbracket ... 309
hbracket-markup ... 309
hcenter ... 309
hcenter-in ... 309
hcenter-in-markup ... 309
hcenter-markup ... 309
head-separation ... 209
header ... 201
header, page ... 208
Hidden notes ... 107
horizontal spacing ... 234
horizontal-shift ... 210
Horizontal_bracket_engraver ... 112
HorizontalBracket ... 112
hspace ... 309
hspace-markup ... 309
hufnagel ... 167
huge ... 309
huge-markup ... 309

hyphens ... 135

I

improvisation ... 28
includePageLayoutFile ... 272, 340
including files ... 192
indent ... 238
instrument ... 198
instrument names ... 99, 206
instrument names, centering ... 100
instrument names, changing ... 100
instrument names, short ... 99
InstrumentName ... 101
instrumentSwitch ... 272, 340
interface, layout ... 260
Interleaved music ... 91
internal documentation ... 259
internal storage ... 274
Internals Reference ... 249
interval ... 4
Invisible notes ... 107
invisible rest ... 38
italic ... 309
italic-markup ... 309
item-interface ... 261

J

jazz chords ... 151
justified-lines ... 314
justified-lines-markup-list ... 314
justify ... 309
justify-field ... 309
justify-field-markup ... 309
justify-markup ... 309
justify-string ... 309
justify-string-markup ... 309

K

keepWithTag ... 194
keepWithTag ... 271, 339
key signature ... 4, 14
Key_engraver ... 16
KeyCancellation ... 16
KeySignature ... 16, 168, 169
killCues ... 270, 338
kirchenpausen ... 40
kneed beams ... 50

L

label ... 272, 340
laissez vibrer ... 36
LaissezVibrerTie ... 37
LaissezVibrerTieColumn ... 37
landscape ... 208
large ... 310
large-markup ... 310
larger ... 310
larger-markup ... 310
layout block ... 205
layout file ... 213
layout interface ... 260
lead sheet ... 145
Lead sheets ... 145
LedgerLineSpanner ... 26
left-align ... 310
left-align-markup ... 310
left-margin ... 209
Ligature_bracket_engraver ... 176, 177
LigatureBracket ... 176
Ligatures ... 176
line ... 310
line breaks ... 50, 56, 215
line-markup ... 310
line-spanner-interface ... 128
line-width ... 209, 238
LineBreakEvent ... 216
List of colors ... 288
listener ... 341
longa ... 31
longa ... 37
lookup ... 310
lookup-markup ... 310
lower ... 310
lower-markup ... 310
lowering text ... 313
ly:add-file-name-alist ... 341
ly:add-interface ... 341
ly:add-listener ... 341
ly:add-option ... 341
ly:all-grob-interfaces ... 341
ly:all-options ... 341
ly:all-stencil-expressions ... 341
ly:assoc-get ... 341
ly:book-add-score! ... 341
ly:book-process ... 341
ly:book-process-to-systems ... 341
ly:box? ... 342
ly:bp ... 342
ly:bracket ... 342
ly:broadcast ... 342
ly:camel-case->lisp-identifier ... 342
ly:chain-assoc-get ... 342
ly:clear-anonymous-modules ... 342
ly:cm ... 342
ly:command-line-code ... 342
ly:command-line-options ... 342
ly:command-line-verbose? ... 342
ly:connect-dispatchers ... 342
ly:context-event-source ... 342
ly:context-events-below ... 342
ly:context-find ... 342
ly:context-grob-definition ... 342
ly:context-id ... 342
ly:context-name ... 342
ly:context-now ... 343
ly:context-parent ... 343
ly:context-property ... 343
ly:context-property-where-defined ... 343
ly:context-pushpop-property ... 343
ly:context-set-property! ... 343
ly:context-unset-property ... 343
ly:context? ... 343
ly:default-scale ... 343
ly:dimension? ... 343
ly:dir? ... 343

ly:duration->string ... 343
ly:duration-dot-count ... 343
ly:duration-factor ... 343
ly:duration-length ... 343
ly:duration-log ... 343
ly:duration<? ... 343
ly:duration? ... 343
ly:effective-prefix ... 343
ly:error ... 344
ly:eval-simple-closure ... 344
ly:event-deep-copy ... 344
ly:event-property ... 344
ly:event-set-property! ... 344
ly:expand-environment ... 344
ly:export ... 344
ly:find-file ... 344
ly:font-config-display-fonts ... 344
ly:font-config-get-font-file ... 344
ly:font-design-size ... 344
ly:font-file-name ... 344
ly:font-get-glyph ... 344
ly:font-glyph-name-to-charcode ... 344
ly:font-glyph-name-to-index ... 344
ly:font-index-to-charcode ... 344
ly:font-load ... 344
ly:font-magnification ... 344
ly:font-metric? ... 345
ly:font-name ... 345
ly:font-sub-fonts ... 345
ly:format ... 345
ly:format-output ... 345
ly:get-all-function-documentation ... 345
ly:get-all-translators ... 345
ly:get-glyph ... 345
ly:get-listened-event-classes ... 345
ly:get-option ... 345
ly:gettext ... 345
ly:grob-alist-chain ... 345
ly:grob-array-length ... 345
ly:grob-array-ref ... 345
ly:grob-array? ... 345
ly:grob-basic-properties ... 345
ly:grob-common-refpoint ... 345
ly:grob-common-refpoint-of-array ... 345
ly:grob-default-font ... 346
ly:grob-extent ... 346
ly:grob-interfaces ... 346
ly:grob-layout ... 346
ly:grob-object ... 346
ly:grob-original ... 346
ly:grob-parent ... 346
ly:grob-pq<? ... 346
ly:grob-properties ... 346
ly:grob-property ... 346
ly:grob-property-data ... 346
ly:grob-relative-coordinate ... 346
ly:grob-robust-relative-extent ... 346
ly:grob-script-priority-less ... 346
ly:grob-set-property! ... 346
ly:grob-staff-position ... 346
ly:grob-suicide! ... 346
ly:grob-system ... 346
ly:grob-translate-axis! ... 346
ly:grob? ... 347
ly:gulp-file ... 347
ly:hash-table-keys ... 347
ly:inch ... 347
ly:input-both-locations ... 347
ly:input-file-line-char-column ... 347
ly:input-location? ... 347
ly:input-message ... 347
ly:interpret-music-expression ... 347
ly:interpret-stencil-expression ... 347
ly:intlog2 ... 347
ly:is-listened-event-class ... 347
ly:item-break-dir ... 347
ly:item? ... 347
ly:iterator? ... 347
ly:lexer-keywords ... 347
ly:lily-lexer? ... 347
ly:lily-parser? ... 347
ly:load-text-dimensions ... 348
ly:make-book ... 348
ly:make-dispatcher ... 348
ly:make-duration ... 348
ly:make-global-context ... 348
ly:make-global-translator ... 348
ly:make-listener ... 348
ly:make-moment ... 348
ly:make-music ... 348
ly:make-music-function ... 348
ly:make-output-def ... 348
ly:make-page-label-marker ... 348
ly:make-page-permission-marker ... 348
ly:make-pango-description-string ... 348
ly:make-paper-outputter ... 349
ly:make-pitch ... 349
ly:make-prob ... 349
ly:make-scale ... 349
ly:make-score ... 349
ly:make-simple-closure ... 349
ly:make-stencil ... 349
ly:make-stream-event ... 349
ly:message ... 349
ly:minimal-breaking ... 218
ly:minimal-breaking ... 349
ly:mm ... 349
ly:module->alist ... 349
ly:module-copy ... 349
ly:modules-lookup ... 349
ly:moment-add ... 349
ly:moment-div ... 349
ly:moment-grace-denominator ... 350
ly:moment-grace-numerator ... 350
ly:moment-main-denominator ... 350
ly:moment-main-numerator ... 350
ly:moment-mod ... 350
ly:moment-mul ... 350
ly:moment-sub ... 350
ly:moment<? ... 350
ly:moment? ... 350
ly:music-compress ... 350
ly:music-deep-copy ... 350
ly:music-duration-compress ... 350
ly:music-duration-length ... 350
ly:music-function-extract ... 350
ly:music-function? ... 350
ly:music-length ... 350
ly:music-list? ... 350
ly:music-mutable-properties ... 350

ly:music-output? ... 350
ly:music-property ... 351
ly:music-set-property! ... 351
ly:music-transpose ... 351
ly:music? ... 351
ly:note-head::stem-attachment ... 351
ly:number->string ... 351
ly:optimal-breaking ... 217
ly:optimal-breaking ... 351
ly:option-usage ... 351
ly:otf->cff ... 351
ly:otf-font-glyph-info ... 351
ly:otf-font-table-data ... 351
ly:otf-font? ... 351
ly:otf-glyph-list ... 351
ly:output-def-clone ... 351
ly:output-def-lookup ... 351
ly:output-def-parent ... 351
ly:output-def-scope ... 351
ly:output-def-set-variable! ... 351
ly:output-def? ... 352
ly:output-description ... 352
ly:output-formats ... 352
ly:outputter-close ... 352
ly:outputter-dump-stencil ... 352
ly:outputter-dump-string ... 352
ly:outputter-output-scheme ... 352
ly:outputter-port ... 352
ly:page-marker? ... 352
ly:page-turn-breaking ... 217
ly:page-turn-breaking ... 352
ly:pango-font-physical-fonts ... 352
ly:pango-font? ... 352
ly:paper-book-pages ... 352
ly:paper-book-paper ... 352
ly:paper-book-performances ... 352
ly:paper-book-scopes ... 352
ly:paper-book-systems ... 352
ly:paper-book? ... 352
ly:paper-fonts ... 352
ly:paper-get-font ... 353
ly:paper-get-number ... 353
ly:paper-outputscale ... 353
ly:paper-score-paper-systems ... 353
ly:paper-system-minimum-distance ... 353
ly:paper-system? ... 353
ly:parse-file ... 353
ly:parser-clear-error ... 353
ly:parser-clone ... 353
ly:parser-define! ... 353
ly:parser-error ... 353
ly:parser-has-error? ... 353
ly:parser-lexer ... 353
ly:parser-lookup ... 353
ly:parser-output-name ... 353
ly:parser-parse-string ... 353
ly:parser-set-note-names ... 353
ly:performance-write ... 353
ly:pfb->pfa ... 354
ly:pitch-alteration ... 354
ly:pitch-diff ... 354
ly:pitch-negate ... 354
ly:pitch-notename ... 354
ly:pitch-octave ... 354
ly:pitch-quartertones ... 354
ly:pitch-semitones ... 354
ly:pitch-steps ... 354
ly:pitch-transpose ... 354
ly:pitch<? ... 354
ly:pitch? ... 354
ly:prob-property ... 354
ly:prob-property? ... 354
ly:prob-set-property! ... 354
ly:prob-type? ... 354
ly:prob? ... 354
ly:programming-error ... 354
ly:progress ... 354
ly:property-lookup-stats ... 355
ly:protects ... 355
ly:pt ... 355
ly:register-stencil-expression ... 355
ly:relative-group-extent ... 355
ly:reset-all-fonts ... 355
ly:round-filled-box ... 355
ly:run-translator ... 355
ly:score-add-output-def! ... 355
ly:score-embedded-format ... 355
ly:score-error? ... 355
ly:score-header ... 355
ly:score-music ... 355
ly:score-output-defs ... 355
ly:score? ... 355
ly:set-default-scale ... 355
ly:set-grob-modification-callback ... 355
ly:set-middle-C! ... 356
ly:set-option ... 356
ly:set-point-and-click ... 356
ly:set-property-cache-callback ... 356
ly:simple-closure? ... 356
ly:skyline-pair? ... 356
ly:skyline? ... 356
ly:smob-protects ... 356
ly:solve-spring-rod-problem ... 356
ly:source-file? ... 356
ly:spanner-bound ... 356
ly:spanner-broken-into ... 356
ly:spanner? ... 356
ly:start-environment ... 356
ly:stderr-redirect ... 356
ly:stencil-add ... 357
ly:stencil-aligned-to ... 357
ly:stencil-combine-at-edge ... 357
ly:stencil-empty? ... 357
ly:stencil-expr ... 357
ly:stencil-extent ... 357
ly:stencil-fonts ... 357
ly:stencil-in-color ... 357
ly:stencil-rotate ... 357
ly:stencil-translate ... 357
ly:stencil-translate-axis ... 357
ly:stencil? ... 357
ly:stream-event? ... 357
ly:string-substitute ... 357
ly:system-print ... 357
ly:system-stretch ... 357
ly:text-dimension ... 357
ly:text-interface::interpret-markup ... 358
ly:translator-description ... 358
ly:translator-group? ... 358
ly:translator-name ... 358

ly:translator? ... 358
ly:transpose-key-alist ... 358
ly:truncate-list! ... 358
ly:ttf->pfa ... 358
ly:ttf-ps-name ... 358
ly:unit ... 358
ly:usage ... 358
ly:version ... 358
ly:warning ... 358
ly:wide-char->utf-8 ... 358
LyricCombineMusic ... 134, 136
LyricExtender ... 135
LyricHyphen ... 135
lyrics ... 53, 130
Lyrics ... 98
Lyrics ... 132, 133, 136, 320
lyrics and melodies ... 132
Lyrics, increasing space between ... 139
lyrics, skip ... 38
lyrics, variables ... 135
LyricSpace ... 132
LyricText ... 132, 144

M

m ... 148
magnify ... 310
magnify-markup ... 310
maj ... 148
majorSevenSymbol ... 150
make-dynamic-script ... 124
makeClusters ... 272, 340
manual beams ... 49
manual staff switches ... 152
marcato ... 69, 314
margins ... 208
mark, rehearsal ... 62
mark, rehearsal, format ... 62
mark, rehearsal, style ... 62
markalphabet ... 310
markalphabet-markup ... 310
markletter ... 310
markletter-markup ... 310
markup ... 119
markup text ... 119
maxima ... 37
measure groupings ... 43
measure lines ... 55
measure lines, invisible ... 56
measure number ... 67
measure number, format ... 59
measure numbers ... 58
measure repeats ... 84
measure, partial ... 44
Measure_grouping_engraver ... 43
MeasureGrouping ... 43
measureLength ... 67
measurePosition ... 67
Medicaea, Editio ... 167
medium ... 310
medium-markup ... 310
melisma ... 134, 135
Melisma_translator ... 135
melismata ... 134
mensural ... 167
Mensural ligatures ... 176
Mensural_ligature_engraver ... 168, 176, 177
MensuralStaffContext ... 184
MensuralVoiceContext ... 184
Mensurstriche layout ... 94
merging notes ... 87
meter ... 42
meter ... 45, 198
meter, polymetric ... 45
metronome ... 99
metronome mark ... 99
metronome marking ... 98
MetronomeMark ... 99
metronomic indication ... 99
mezzosoprano clef ... 11
MIDI ... 17, 204
MIDI block ... 206
minimumFret ... 160
minimumPageTurnLength ... 217
minimumRepeatLengthForPageTurn ... 218
modern accidental style ... 21, 22
modern accidentals ... 22
modern cautionary accidental style ... 21
modern style accidentals ... 21
modern style cautionary accidentals ... 21
modern-cautionary ... 21
modern-voice ... 22
modern-voice-cautionary ... 22
modes ... 15
modifiers, in chords ... 148
mordent ... 69, 314
movements, multiple ... 191
moving text ... 313
multi measure rests ... 39
multi-measure rest ... 39
MultiMeasureRest ... 42
MultiMeasureRestNumber ... 42
MultiMeasureRestText ... 42
multiple voices ... 88
multivoice accidentals ... 22
Music classes ... 274
Music expressions ... 274
Music properties ... 274
music, beginners' ... 26
music, unmetered ... 67
Musica ficta ... 185
musicglyph ... 310
musicglyph-markup ... 310
musicMap ... 270, 338
musicological analysis ... 112

N

name of singer ... 141
natural ... 310
natural sign ... 4
natural-markup ... 310
new contexts ... 250
New_fingering_engraver ... 260
NewBassFigure ... 188
niente, al ... 71
no reset accidental style ... 23
no-reset accidental style ... 23

non-empty texts ... 114
Non-guitar tablatures ... 161
noPageBreak ... 270, 339
noPageTurn ... 270, 339
normal-size-sub ... 310
normal-size-sub-markup ... 310
normal-size-super ... 310
normal-size-super-markup ... 310
normal-text ... 310
normal-text-markup ... 310
normalsize ... 310
normalsize-markup ... 310
notation, explaining ... 110
note ... 311
note collisions ... 87
note durations ... 30
note grouping bracket ... 112
note head styles ... 305
note heads, Aiken ... 27
note heads, ancient ... 168
note heads, cross ... 26
note heads, diamond ... 26
note heads, easy notation ... 26
note heads, easy play ... 26
note heads, guitar ... 26
note heads, harmonic ... 26
note heads, improvisation ... 28
note heads, parlato ... 26
note heads, practice ... 26
note heads, sacred harp ... 27
note heads, shape ... 27
note heads, slashed ... 28
note heads, special ... 26
note names, default ... 4
note names, Dutch ... 4
note names, other languages ... 6
note value ... 31
note-by-number ... 311
note-by-number-markup ... 311
note-collision-interface ... 330, 331, 333
note-event ... 156
note-markup ... 311
Note_heads_engraver ... 48, 258
NoteCollision ... 87, 89
NoteColumn ... 88
NoteEvent ... 274
NoteHead ... 26, 27, 168, 283
notes, colored ... 107
notes, dotted ... 30
notes, ghost ... 109
notes, parenthesized ... 109
NoteSpacing ... 235
null ... 311
null-markup ... 311
number ... 311
number of staff lines, setting ... 96
number-markup ... 311

O

octavation ... 16
octavation ... 17
octave ... 271, 340
octave check ... 7
octave correction ... 7
octave transposition ... 12
oddFooterMarkup ... 201
oddHeaderMarkup ... 201
oldaddlyrics ... 270, 339
on-the-fly ... 311
on-the-fly-markup ... 311
open ... 69, 314
opus ... 198
organ pedal marks ... 69, 314
orientation ... 208
ornaments ... 63, 68
ossia ... 96
ossia ... 97, 259
ottava ... 16
OttavaBracket ... 17
outside-staff-horizontal-padding ... 233
outside-staff-padding ... 233
outside-staff-priority ... 233
override ... 311
override-lines ... 314
override-lines-markup-list ... 314
override-markup ... 311
overrideProperty ... 271, 340
OverrideProperty ... 256

P

pad-around ... 311
pad-around-markup ... 311
pad-markup ... 311
pad-markup-markup ... 311
pad-to-box ... 311
pad-to-box-markup ... 311
pad-x ... 311
pad-x-markup ... 311
padding ... 262
page breaks ... 238
page breaks, forcing ... 198
page formatting ... 208
page layout ... 201, 238
page size ... 208
page-breaking-between-system-padding ... 210
page-ref ... 311
page-ref-markup ... 311
page-spacing-weight ... 211
page-top-space ... 209
pageBreak ... 272, 341
pageTurn ... 272, 340
Pango ... 124
paper size ... 208
paper-height ... 208
paper-width ... 208
papersize ... 208
parallelMusic ... 272, 340
parentheses ... 109
parenthesize ... 272, 341
parenthesized accidental ... 5
parlato note heads ... 26
part combiner ... 89
partcombine ... 272, 341
PartCombineMusic ... 90
partial measure ... 44
Pedals ... 153

percent repeats ... 84
PercentRepeat ... 85
PercentRepeatCounter ... 85
PercentRepeatedMusic ... 85
percussion ... 156
Petrucci ... 167
phrasing brackets ... 112
phrasing marks ... 74
phrasing slurs ... 74
phrasing, in lyrics ... 134
PhrasingSlur ... 74
piano accidental style ... 22
piano accidentals ... 22
piano cautionary accidental style ... 22
piano cautionary accidentals ... 22
PianoPedalBracket ... 154
PianoStaff ... 24
PianoStaff ... 78, 101, 151
pickup measure ... 44
piece ... 198
pipeSymbol ... 61
pitch names ... 1
Pitch names ... 2, 4, 6, 7
pitch names, other languages ... 6
pitch range ... 24
Pitch_squash_engraver ... 29, 258, 323
pitched trills ... 78
pitchedTrill ... 270, 339
pitches ... 1
poet ... 198
polymetric ... 34, 45
polymetric scores ... 254
polymetric signatures ... 45
polymetric time signature ... 45
polyphonic music ... 88
portato ... 69, 314
postscript ... 311
postscript-markup ... 311
prall ... 69, 314
prall, down ... 69, 314
prall, up ... 69, 314
prallmordent ... 69, 314
prallprall ... 69, 314
prima volta ... 80
print-first-page-number ... 208
print-page-number ... 208
printallheaders ... 200, 210
printing chord names ... 148
properties ... 251
PropertySet ... 256
punctuation ... 130
put-adjacent ... 312
put-adjacent-markup ... 312

Q

quarter tones ... 5
quarter-tone ... 6
quoteDuring ... 271, 339
QuoteMusic ... 102
quotes, in lyrics ... 130
quoting other voices ... 101, 102

R

r ... 37
R ... 39
ragged-bottom ... 209
ragged-last ... 238
ragged-last-bottom ... 209
ragged-right ... 238
raise ... 312
raise-markup ... 312
raising text ... 313
range of pitches ... 24
regular line breaks ... 215
rehearsal mark format ... 62
rehearsal mark style ... 62
rehearsal marks ... 62
RehearsalMark ... 63, 119
RehearsalMark ... 121
relative ... 2
relative octave specification ... 2
reminder accidental ... 5
removals, in chords ... 147
removeWithTag ... 194
removeWithTag ... 270, 338
Renaissance music ... 94
repeat bars ... 55
repeat, ambiguous ... 82
repeatCommands ... 82
RepeatedMusic ... 82, 83, 274
repeating ties ... 35
repeats ... 56, 80
RepeatSlash ... 85
resetRelativeOctave ... 272, 340
rest ... 37
Rest ... 38, 169
rest, church ... 40
rest, full measure ... 39
rest, multi measure ... 39
RestCollision ... 89
rests, ancient ... 169
reverseturn ... 69, 314
RevertProperty ... 256
RhythmicStaff ... 156
right hand fingerings for guitar ... 164
right-align ... 312
right-align-markup ... 312
rightHandFinger ... 271, 339
roman ... 312
roman-markup ... 312
root of chord ... 147
rotate ... 312
rotate-markup ... 312
rotated text ... 311

S

s ... 38
sacred harp note heads ... 27
sans ... 312
sans-markup ... 312
SATB ... 135
scordatura ... 16
score ... 312
Score ... 43, 68, 316, 318
score-markup ... 312

`scoreTitleMarkup` ... 201
`scoreTweak` ... 270, 338
`Script` ... 70
script on multi-measure rest ... 40
scripts ... 68
seconda volta ... 80
segno ... 63, 69, 314
segno on bar line ... 116
selecting font size ... 105
`self-alignment-interface` ... 261
semi-flats, semi-sharps ... 5
`semiflat` ... 312
`semiflat-markup` ... 312
`semisharp` ... 312
`semisharp-markup` ... 312
`SeparatingGroupSpanner` ... 235
`SeparationItem` ... 235
`SequentialMusic` ... 274
`sesquiflat` ... 312
`sesquiflat-markup` ... 312
`sesquisharp` ... 312
`sesquisharp-markup` ... 312
`set-accidental-style` ... 19
shape notes ... 27
`shapeNoteStyles` ... 27
sharp ... 4
`sharp` ... 6
`sharp` ... 312
sharp, double ... 4
`sharp-markup` ... 312
sheet music, empty ... 111
shift note ... 87
`shiftDurations` ... 271, 339
shifting voices ... 88
shorten measures ... 44
`showLastLength` ... 195
`side-position-interface` ... 261
signatures, polymetric ... 45
`simple` ... 312
`simple-markup` ... 312
simultaneous notes and accidentals ... 24
`SimultaneousMusic` ... 274
singer name ... 141
skip ... 38
`SkipMusic` ... 39
`skipTypesetting` ... 195
slashed note heads ... 28
`slashed-digit` ... 312
`slashed-digit-markup` ... 312
`Slur` ... 74
slurs ... 73
`small` ... 313
`small-markup` ... 313
`smallCaps` ... 313
`smallCaps-markup` ... 313
`smaller` ... 313
`smaller-markup` ... 313
soprano clef ... 11
Sound ... 204
space between staves ... 222
space inside systems ... 222
space note ... 38
spaces, in lyrics ... 130
`spacing` ... 235
Spacing lyrics ... 139
Spacing, display of properties ... 214
spacing, horizontal ... 234
spacing, vertical ... 222
`spacing-spanner-interface` ... 335, 336
`SpacingSpanner` ... 234, 235, 236
`spacingTweaks` ... 272, 340
`SpanBar` ... 58
special note heads ... 26
Square neumes ligatures ... 177
staccatissimo ... 69, 314
staccato ... 69, 314
`staff` ... 95, 96
`Staff` ... 24
`Staff` ... 26, 39, 46, 48, 58
`Staff` ... 98
`Staff` ... 101, 112, 174, 235, 316
staff distance ... 222
staff group ... 93
staff lines, setting number of ... 96
staff lines, setting thickness of ... 96
staff paper, blank ... 111
staff size, setting ... 213
staff switch, manual ... 152
staff switching ... 154
staff, choir ... 93
staff, multiple ... 93
staff, nested ... 94
`Staff.midiInstrument` ... 206
`StaffGroup` ... 61
`StaffGroup` ... 95
`StaffSpacing` ... 235
`StaffSymbol` ... 96, 213
standard font size ... 105
stanza number ... 141
`StanzaNumber` ... 144
start of system ... 93
`staves` ... 95
staves, blank sheet ... 111
staves, empty ... 97
staves, Frenched ... 96
staves, hiding ... 97
stem ... 109
`Stem` ... 171, 283
stem, cross staff ... 151
stem, direction ... 109
stem, down ... 109
stem, neutral ... 109
stem, up ... 109
stem, with slash ... 65
`stem-spacing-correction` ... 235
`stemLeftBeamCount` ... 54
`stemRightBeamCount` ... 54
`StemTremolo` ... 84
`stencil` ... 313
`stencil-markup` ... 313
stopped ... 69, 314
String numbers ... 160
`StringNumber` ... 160
`StrokeFinger` ... 164
`strut` ... 313
`strut-markup` ... 313
style, rehearsal mark ... 62
`sub` ... 313
`sub-markup` ... 313
subbass clef ... 11

subbass clef, subbass 11
subdivideBeams 50
subsubtitle 198
subtitle 198
suggestAccidentals 185
super 313
super-markup 313
sus 148
SustainPedal 153
system 93
system-count 209
systemSeparatorMarkup 210
SystemStartBar 95
SystemStartBrace 95
SystemStartBracket 95
systemStartDelimiterHierarchy 95

T

Tab_note_heads_engraver 162
tablature 160
Tablatures basic 160
TabNoteHead 161
TabStaff 160
TabStaff 161
TabVoice 160
TabVoice 161
tag 193
tag 271, 339
tagline 198
tagline 201
teeny 313
teeny-markup 313
Tempo 98
tempo indication 99
tenor clef 11
tenuto 69, 314
text 313
text items, non-empty 114
text markup 119
text on multi-measure rest 40
Text scripts 114
Text spanners 115
Text, other languages 113
text-balloon-interface 110
text-interface 261, 311
text-markup 313
text-script-interface 261
TextScript 70, 115, 119, 122
TextSpanner 116, 128
textSpannerDown 116
textSpannerNeutral 116
textSpannerUp 116
thickness of staff lines, setting 96
thumb marking 69, 314
thumb-script 106
tie 35
tie 37
Tie 37, 264
tied-lyric 313
tied-lyric-markup 313
ties, in lyrics 130, 133
ties, laissez vibrer 36
time administration 67
time signature 42
time signatures 172
Time signatures, multiple 254
TimeScaledMusic 34
TimeSignature 44, 45, 48, 173
timing (within the score) 67
Timing-translator 48
Timing_translator 44, 46, 58, 68, 318
tiny 313
tiny-markup 313
title 198
titles 201
tocItem 270, 338
top-margin 209
translate 313
translate-markup 313
translate-scaled 313
translate-scaled-markup 313
translating text 313
Translation 260
transparent 313
Transparent notes 107
transparent-markup 313
transpose 8
transposedCueDuring 271, 340
TransposedMusic 11
transposing clefs 12
transposing instrument 17
transposing instrument 18
transposing instruments 9
transposition 8
transposition 270, 339
transposition of pitches 8
transposition, instrument 17
transposition, MIDI 17
treble clef 11
tremolo beams 83
tremolo marks 84
tremoloFlags 84
triangle 313
triangle-markup 313
trill 69, 314
TrillSpanner 79, 128
triplet 34
triplet bracket 32
triplet bracket length 33
triplet formatting 32
triplets 31
triplets, nested 32
tuning automatic beaming 51
tuplet 34
tuplet bracket 32
tuplet bracket length 33
tuplet formatting 32
TupletBracket 32, 34
tupletFullLength 33
tupletFullLengthNote 33
TupletNumber 32, 34
tupletNumberFormatFunction 32
tuplets 31
tuplets, nested 32
tupletSpannerDuration 32
turn 69, 314
tweak 271, 340
tweaking 259

typeset text 119
typewriter 313
typewriter-markup 313

U

underline 314
underline-markup 314
UnfoldedRepeatedMusic 82, 83
unfoldRepeats 271, 339
unmetered music 67
upbeat 44
upbow 69, 314
upright 314
upright-markup 314

V

varbaritone clef 11
varcoda 69, 314
variables 190
Vaticana, Editio 167
VaticanaStaffContext 183
VaticanaVoiceContext 183
vcenter 314
vcenter-markup 314
verbatim-file 314
verbatim-file-markup 314
vertical spacing 222, 238
VerticalAlignment 222, 224
VerticalAxisGroup 98
VerticalAxisGroup 222
violin clef 11
VocalName 144
Voice 26, 29, 39, 71, 90, 102, 104, 132, 133, 176, 205, 235, 252, 262
VoiceFollower 128, 155
volta 80
volta brackets and ties 35
Volta_engraver 149
VoltaBracket 82, 83
VoltaRepeatedMusic 82, 83

W

whichBar 57
White mensural ligatures 176
whiteout 314
whiteout-markup 314
whole rest for a full measure 39
with-color 314
with-color-markup 314
with-dimensions 314
with-dimensions-markup 314
with-url 314
with-url-markup 314
withMusicProperty 271, 340
wordwrap 314
wordwrap-field 314
wordwrap-field-markup 314
wordwrap-lines 314
wordwrap-lines-markup-list 314
wordwrap-markup 314
wordwrap-string 314
wordwrap-string-markup 314
Writing music in parallel 91

X

x11-color 108