

LilyPond

The music typesetter

Notation Reference

The LilyPond development team

This manual provides a reference for all notation that can be produced with LilyPond version 2.13.27. It assumes that the reader is familiar with the material in the [Section “Learning Manual”](#) in *Learning Manual*.

For more information about how this manual fits with the other documentation, or to read this manual in other formats, see [Section “Manuals”](#) in *General Information*.

If you are missing any manuals, the complete documentation can be found at <http://www.lilypond.org/>.

Copyright © 1999–2010 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 2.13.27

Table of Contents

1	Musical notation	1
1.1	Pitches	1
1.1.1	Writing pitches	1
	Absolute octave entry	1
	Relative octave entry	2
	Accidentals	4
	Note names in other languages	7
	Non-Western note names and accidentals	8
1.1.2	Changing multiple pitches	9
	Octave checks	9
	Transpose	10
1.1.3	Displaying pitches	13
	Clef	13
	Key signature	16
	Ottava brackets	18
	Instrument transpositions	19
	Automatic accidentals	20
	Ambitus	27
1.1.4	Note heads	29
	Special note heads	29
	Easy notation note heads	30
	Shape note heads	32
	Improvisation	34
1.2	Rhythms	35
1.2.1	Writing rhythms	35
	Durations	35
	Tuplets	37
	Scaling durations	41
	Ties	42
1.2.2	Writing rests	45
	Rests	45
	Invisible rests	47
	Full measure rests	48
1.2.3	Displaying rhythms	52
	Time signature	52
	Upbeats	55
	Unmetered music	55
	Polymetric notation	57
	Automatic note splitting	59
	Showing melody rhythms	60
1.2.4	Beams	62
	Automatic beams	63
	Setting automatic beam behavior	64
	Manual beams	69
	Feathered beams	72
1.2.5	Bars	72
	Bar lines	72
	Bar numbers	77

Bar and bar number checks	81
Rehearsal marks	81
1.2.6 Special rhythmic concerns	83
Grace notes	84
Aligning to cadenzas	88
Time administration	89
1.3 Expressive marks	90
1.3.1 Attached to notes	90
Articulations and ornamentations	90
Dynamics	93
New dynamic marks	97
1.3.2 Curves	98
Slurs	99
Phrasing slurs	101
Breath marks	102
Falls and doits	104
1.3.3 Lines	104
Glissando	104
Arpeggio	105
Trills	108
1.4 Repeats	110
1.4.1 Long repeats	110
Normal repeats	110
Manual repeat marks	114
Written-out repeats	116
1.4.2 Short repeats	117
Percent repeats	117
Tremolo repeats	119
1.5 Simultaneous notes	120
1.5.1 Single voice	121
Chorded notes	121
Chord repetition	121
Simultaneous expressions	122
Clusters	123
1.5.2 Multiple voices	123
Single-staff polyphony	123
Voice styles	126
Collision resolution	126
Automatic part combining	130
Writing music in parallel	133
1.6 Staff notation	136
1.6.1 Displaying staves	136
Instantiating new staves	136
Grouping staves	137
Nested staff groups	141
1.6.2 Modifying single staves	142
Staff symbol	142
Ossia staves	145
Hiding staves	148
1.6.3 Writing parts	151
Metronome marks	151
Instrument names	154
Quoting other voices	157
Formatting cue notes	161

1.7	Editorial annotations	164
1.7.1	Inside the staff	164
	Selecting notation font size	164
	Fingering instructions	165
	Hidden notes	168
	Coloring objects	168
	Parentheses	170
	Stems	170
1.7.2	Outside the staff	171
	Balloon help	171
	Grid lines	172
	Analysis brackets	174
1.8	Text	175
1.8.1	Writing text	175
	Text scripts	176
	Text spanners	177
	Text marks	177
	Separate text	181
1.8.2	Formatting text	182
	Text markup introduction	182
	Selecting font and font size	184
	Text alignment	186
	Graphic notation inside markup	189
	Music notation inside markup	192
	Multi-page markup	194
1.8.3	Fonts	195
	Fonts explained	195
	Single entry fonts	197
	Entire document fonts	197
2	Specialist notation	198
2.1	Vocal music	198
2.1.1	Common notation for vocal music	198
	References for vocal music and lyrics	198
	Opera	198
	Song books	198
	Spoken music	199
	Chants	199
	Ancient vocal music	199
2.1.2	Entering lyrics	199
	Lyrics explained	199
	Setting simple songs	201
	Working with lyrics and variables	201
2.1.3	Aligning lyrics to a melody	202
	Automatic syllable durations	202
	Manual syllable durations	203
	Multiple syllables to one note	204
	Multiple notes to one syllable	204
	Skipping notes	206
	Extenders and hyphens	206
	Lyrics and repeats	207
2.1.4	Specific uses of lyrics	207
	Divisi lyrics	207
	Lyrics independent of notes	208

Spacing out syllables	208
Centering lyrics between staves	209
2.1.5 Stanzas	209
Adding stanza numbers	210
Adding dynamics marks to stanzas	210
Adding singers' names to stanzas	210
Stanzas with different rhythms	211
Printing stanzas at the end	213
Printing stanzas at the end in multiple columns	214
2.2 Keyboard and other multi-staff instruments	216
2.2.1 Common notation for keyboards	216
References for keyboards	216
Changing staff manually	217
Changing staff automatically	218
Staff-change lines	219
Cross-staff stems	220
2.2.2 Piano	221
Piano pedals	222
2.2.3 Accordion	222
Discant symbols	223
2.2.4 Harp	226
References for harps	226
Harp pedals	227
2.3 Unfretted string instruments	227
2.3.1 Common notation for unfretted strings	228
References for unfretted strings	228
Bowing indications	228
Harmonics	229
Snap (Bartók) pizzicato	230
2.4 Fretted string instruments	230
2.4.1 Common notation for fretted strings	230
References for fretted strings	231
String number indications	231
Default tablatures	233
Custom tablatures	239
Fret diagram markups	241
Predefined fret diagrams	249
Automatic fret diagrams	258
Right-hand fingerings	260
2.4.2 Guitar	262
Indicating position and barring	262
Indicating harmonics and dampened notes	262
2.4.3 Banjo	264
Banjo tablatures	264
2.5 Percussion	264
2.5.1 Common notation for percussion	264
References for percussion	265
Basic percussion notation	265
Drum rolls	266
Pitched percussion	266
Percussion staves	267
Custom percussion staves	268
Ghost notes	272
2.6 Wind instruments	273

2.6.1	Common notation for wind instruments	273
	References for wind instruments	273
	Fingerings	274
2.6.2	Bagpipes	276
	Bagpipe definitions	276
	Bagpipe example	277
2.7	Chord notation	278
2.7.1	Chord mode	278
	Chord mode overview	279
	Common chords	280
	Extended and altered chords	281
2.7.2	Displaying chords	284
	Printing chord names	284
	Customizing chord names	286
2.7.3	Figured bass	290
	Introduction to figured bass	291
	Entering figured bass	291
	Displaying figured bass	294
2.8	Contemporary music	297
2.8.1	Pitch and harmony in contemporary music	297
	References for pitch and harmony in contemporary music	297
	Microtonal notation	297
	Contemporary key signatures and harmony	297
2.8.2	Contemporary approaches to rhythm	297
	References for contemporary approaches to rhythm	297
	Tuplets in contemporary music	297
	Contemporary time signatures	298
	Extended polymetric notation	298
	Beams in contemporary music	298
	Bar lines in contemporary music	298
2.8.3	Graphical notation	298
2.8.4	Contemporary scoring techniques	298
2.8.5	New instrumental techniques	298
2.8.6	Further reading and scores of interest	298
	Books and articles on contemporary musical notation	298
	Scores and musical examples	298
2.9	Ancient notation	298
2.9.1	Overview of the supported styles	299
2.9.2	Ancient notation—common features	300
	Pre-defined contexts	300
	Ligatures	300
	Custodes	300
	Figured bass support	301
2.9.3	Typesetting mensural music	301
	Mensural contexts	301
	Mensural clefs	302
	Mensural time signatures	303
	Mensural note heads	304
	Mensural flags	304
	Mensural rests	305
	Mensural accidentals and key signatures	306
	Annotational accidentals (<i>musica ficta</i>)	306
	White mensural ligatures	307
2.9.4	Typesetting Gregorian chant	308

Gregorian chant contexts	308
Gregorian clefs	308
Gregorian accidentals and key signatures	309
Divisiones	310
Gregorian articulation signs	310
Augmentum dots (<i>morae</i>)	311
Gregorian square neume ligatures	311
2.9.5 Working with ancient music—scenarios and solutions	318
Incipits	318
Mensurstriche layout	318
Transcribing Gregorian chant	319
Ancient and modern from one source	322
Editorial markings	322
2.10 World music	322
2.10.1 Arabic music	322
References for Arabic music	322
Arabic note names	323
Arabic key signatures	323
Arabic time signatures	325
Arabic music example	326
Further reading for Arabic music	327
2.10.2 Turkish classical music	327
References for Turkish classical music	327
Turkish note names	327
3 General input and output	328
3.1 Input structure	328
3.1.1 Structure of a score	328
3.1.2 Multiple scores in a book	329
3.1.3 File structure	330
3.2 Titles and headers	332
3.2.1 Creating titles	332
3.2.2 Custom titles	335
3.2.3 Reference to page numbers	336
3.2.4 Table of contents	337
3.3 Working with input files	339
3.3.1 Including LilyPond files	339
3.3.2 Different editions from one source	340
Using variables	340
Using tags	341
3.3.3 Text encoding	344
3.3.4 Displaying LilyPond notation	345
3.4 Controlling output	345
3.4.1 Extracting fragments of music	345
3.4.2 Skipping corrected music	346
3.4.3 Alternative output formats	346
3.4.4 Replacing the notation font	347
3.5 MIDI output	347
3.5.1 Creating MIDI files	347
Instrument names	348
3.5.2 MIDI block	349
3.5.3 What goes into the MIDI output?	350
Supported in MIDI	350
Unsupported in MIDI	351

3.5.4	Repeats in MIDI	351
3.5.5	Controlling MIDI dynamics	352
	Dynamic marks	352
	Overall MIDI volume	352
	Equalizing different instruments (i)	353
	Equalizing different instruments (ii)	354
3.5.6	Percussion in MIDI	356
4	Spacing issues	357
4.1	Paper and pages	357
4.1.1	Paper size	357
4.1.2	Page formatting	358
	Vertical dimensions	358
	Horizontal dimensions	360
	Other layout variables	362
4.2	Music layout	364
4.2.1	Setting the staff size	365
4.2.2	Score layout	365
4.3	Breaks	366
4.3.1	Line breaking	366
4.3.2	Page breaking	367
4.3.3	Optimal page breaking	368
4.3.4	Optimal page turning	369
4.3.5	Minimal page breaking	370
4.3.6	Explicit breaks	370
4.3.7	Using an extra voice for breaks	371
4.4	Vertical spacing	373
4.4.1	Vertical spacing inside a system	373
	Spacing between staves	373
	Spacing of non-staff lines	375
4.4.2	Vertical spacing between systems	377
4.4.3	Explicit staff and system positioning	377
4.4.4	Vertical collision avoidance	382
4.5	Horizontal spacing	383
4.5.1	Horizontal spacing overview	383
4.5.2	New spacing area	385
4.5.3	Changing horizontal spacing	385
4.5.4	Line length	387
4.5.5	Proportional notation	388
4.6	Fitting music onto fewer pages	394
4.6.1	Displaying spacing	394
4.6.2	Changing spacing	396
5	Changing defaults	398
5.1	Interpretation contexts	398
5.1.1	Contexts explained	398
	Score - the master of all contexts	398
	Top-level contexts - staff containers	398
	Intermediate-level contexts - staves	399
	Bottom-level contexts - voices	399
5.1.2	Creating contexts	400
5.1.3	Keeping contexts alive	401
5.1.4	Modifying context plug-ins	403

5.1.5	Changing context default settings.....	405
5.1.6	Defining new contexts.....	406
5.1.7	Aligning contexts.....	408
5.2	Explaining the Internals Reference.....	409
5.2.1	Navigating the program reference.....	409
5.2.2	Layout interfaces.....	410
5.2.3	Determining the grob property.....	411
5.2.4	Naming conventions.....	412
5.3	Modifying properties.....	412
5.3.1	Overview of modifying properties.....	412
5.3.2	The <code>\set</code> command.....	412
5.3.3	The <code>\override</code> command.....	414
5.3.4	The <code>\tweak</code> command.....	416
5.3.5	<code>\set</code> vs. <code>\override</code>	418
5.4	Useful concepts and properties.....	418
5.4.1	Input modes.....	418
5.4.2	Direction and placement.....	419
5.4.3	Context layout order.....	420
5.4.4	Distances and measurements.....	421
5.4.5	Staff symbol properties.....	421
5.4.6	Spanners.....	422
	Using the <code>spanner-interface</code>	422
	Using the <code>line-spanner-interface</code>	424
5.4.7	Visibility of objects.....	426
	Removing the stencil.....	427
	Making objects transparent.....	427
	Painting objects white.....	427
	Using break-visibility.....	428
	Special considerations.....	429
5.4.8	Line styles.....	431
5.4.9	Rotating objects.....	432
	Rotating layout objects.....	432
	Rotating markup.....	432
5.5	Advanced tweaks.....	433
5.5.1	Aligning objects.....	433
	Setting <code>X-offset</code> and <code>Y-offset</code> directly.....	433
	Using the <code>side-position-interface</code>	434
	Using the <code>self-alignment-interface</code>	434
	Using the <code>break-alignable-interface</code>	435
5.5.2	Vertical grouping of grobs.....	438
5.5.3	Modifying stencils.....	438
5.5.4	Modifying shapes.....	438
	Modifying ties and slurs.....	438
5.6	Using music functions.....	440
5.6.1	Substitution function syntax.....	440
5.6.2	Substitution function examples.....	441

Appendix A	Notation manual tables	443
A.1	Chord name chart	443
A.2	Common chord modifiers	444
A.3	Predefined fretboard diagrams	447
A.4	MIDI instruments	455
A.5	List of colors	455
A.6	The Feta font	457
	Clef glyphs	457
	Time Signature glyphs	457
	Number glyphs	458
	Accidental glyphs	458
	Default Notehead glyphs	459
	Special Notehead glyphs	459
	Shape-note Notehead glyphs	460
	Rest glyphs	462
	Flag glyphs	462
	Dot glyphs	463
	Dynamic glyphs	463
	Script glyphs	463
	Arrowhead glyphs	466
	Bracket-tip glyphs	466
	Pedal glyphs	466
	Accordion glyphs	466
	Vaticana glyphs	467
	Medicaea glyphs	468
	Hufnagel glyphs	468
	Mensural glyphs	469
	Neomensural glyphs	471
	Petrucchi glyphs	472
	Solesmes glyphs	473
A.7	Note head styles	473
A.8	Text markup commands	474
	A.8.1 Font	474
	A.8.2 Align	482
	A.8.3 Graphic	496
	A.8.4 Music	501
	A.8.5 Instrument Specific Markup	504
	A.8.6 Other	508
A.9	Text markup list commands	512
A.10	List of articulations	513
	Articulation scripts	513
	Ornament scripts	514
	Fermata scripts	514
	Instrument-specific scripts	514
	Repeat sign scripts	514
	Ancient scripts	514
A.11	Percussion notes	515
A.12	Technical glossary	516
	alist	516
	callback	516
	closure	516
	glyph	516
	grob	517
	immutable	517

interface	517
lexer	517
mutable	517
output-def	518
parser	518
parser variable	518
prob	518
simple closure	518
smob	519
stencil	519
A.13 All context properties	519
A.14 Layout properties	529
A.15 Available music functions	544
A.16 Predefined type predicates	548
R5RS primary predicates	548
R5RS secondary predicates	549
Guile predicates	549
LilyPond scheme predicates	549
LilyPond exported predicates	550
A.17 Scheme functions	550
Appendix B Cheat sheet	571
Appendix C LilyPond grammar	575
Appendix D GNU Free Documentation License	594
Appendix E LilyPond command index	601
Appendix F LilyPond index	610

This chapter explains how to create musical notation.

dolce e molto legato

p *cresc.* *sf*

38

p

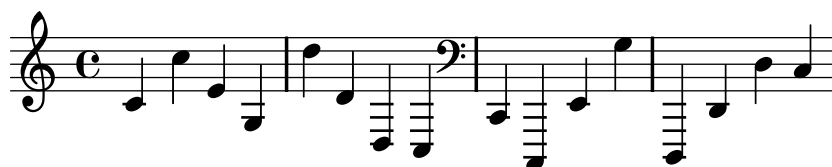
1.1.1 Writing pitches

Absolute octave entry

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



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



See also

Music Glossary: [Section “Pitch names” in *Music Glossary*](#).

Snippets: [Section “Pitches” in *Snippets*](#).

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 each pitch inside *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
}
```



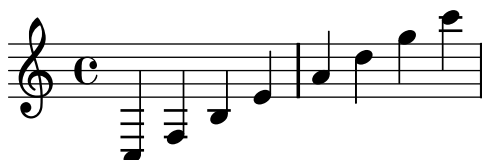
Octave changing marks are used for intervals greater than a fourth:

```
\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
}
```



When `\relative` blocks are nested, the innermost `\relative` block applies.

```
\relative c' {
  c d e f
  \relative c'' {
    c d e f
  }
}
```



`\relative` has no effect on `\chordmode` blocks.

```
\new Staff {
  \relative c''' {
    \chordmode { c1 }
  }
  \chordmode { c1 }
}
```



`\relative` is not allowed inside of `\chordmode` blocks.

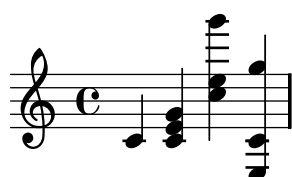
Music inside a `\transpose` block is absolute unless a `\relative` is included.

```
\relative c' {
  d e
  \transpose f g {
    d e
    \relative c' {
      d e
    }
  }
}
```



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. Examine the next example carefully, paying attention to the `c` notes.

```
\relative c' {
  c
  <c e g>
  <c' e g'>
  <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 fesfes
}
```



See also

Music Glossary: [Section “fifth”](#) in *Music Glossary*, [Section “interval”](#) in *Music Glossary*, [Section “Pitch names”](#) in *Music Glossary*.

Notation Reference: [\[Octave checks\]](#), page 9.

Snippets: [Section “Pitches”](#) in *Snippets*.

Internals Reference: [Section “RelativeOctaveMusic”](#) in *Internals Reference*.

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.

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 [Section “Accidentals and key signatures”](#) in *Learning Manual*.

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 [\[Note names in other languages\]](#), page 7.

```
ais1 aes aisis aeses
```



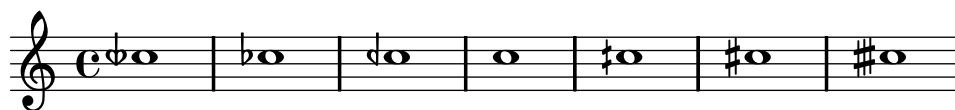
A natural will cancel the effect of an accidental or key signature. However, naturals are not encoded into the note name syntax with a suffix; a natural pitch is shown as a simple note name:

```
a4 aes a2
```



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

```
ceseh1 ces ceh c cih cis cish
```



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

Hiding accidentals on tied notes at the beginning of a new system

This shows how to hide accidentals on tied notes at the beginning of a new system.

```
\relative c'' {
  \override Accidental #'hide-tied-accidental-after-break = ##t
  cis1~ cis~
  \break
  cis
}
```



Preventing extra naturals from being automatically added

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 `f` in the `Staff` context.

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



See also

Music Glossary: Section “sharp” in *Music Glossary*, Section “flat” in *Music Glossary*, Section “double sharp” in *Music Glossary*, Section “double flat” in *Music Glossary*, Section “Pitch names” in *Music Glossary*, Section “quarter tone” in *Music Glossary*.

Learning Manual: Section “Accidentals and key signatures” in *Learning Manual*.

Notation Reference: [Automatic accidentals], page 20, [Annotational accidentals (musica ficta)], page 306, [Note names in other languages], page 7.

Snippets: Section “Pitches” in *Snippets*.

Internals Reference: Section “Accidental-engraver” in *Internals Reference*, Section “Accidental” in *Internals Reference*, Section “AccidentalCautionary” in *Internals Reference*, Section “accidental-interface” in *Internals Reference*.

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.

Note names in other languages

There are predefined sets of note and accidental names for various other languages. To use them, include the language-specific init file listed below. For example, to use English note names, add `\include "english.ly"` to the input file.

Note: Because some other include files (such as `predefined-fretboards.ly`) use default (Nederlands) note names, the `\include` command for the language file should be placed after all other LilyPond distribution files.

The available language files and the note names they define are:

Language File	Note Names
<code>'nederlands.ly'</code>	c d e f g a bes b
<code>'arabic.ly'</code>	do re mi fa sol la sib si
<code>'catalan.ly'</code>	do re mi fa sol la sib si
<code>'deutsch.ly'</code>	c d e f g a b h
<code>'english.ly'</code>	c d e f g a bf b
<code>'espanol.ly'</code>	do re mi fa sol la sib si
<code>'italiano.ly'</code>	do re mi fa sol la sib si
<code>'norsk.ly'</code>	c d e f g a b h
<code>'portugues.ly'</code>	do re mi fa sol la sib si
<code>'suomi.ly'</code>	c d e f g a b h
<code>'svenska.ly'</code>	c d e f g a b h
<code>'vlaams.ly'</code>	do re mi fa sol la sib si

and the accidental suffixes they define are:

Language File	sharp	flat	double sharp	double flat
<code>'nederlands.ly'</code>	-is	-es	-isis	-eses
<code>'arabic.ly'</code>	-d	-b	-dd	-bb
<code>'catalan.ly'</code>	-d/-s	-b	-dd/-ss	-bb
<code>'deutsch.ly'</code>	-is	-es	-isis	-eses
<code>'english.ly'</code>	-s/-sharp	-f/-flat	-ss/-x/-sharpsharp	-ff/-flatflat
<code>'espanol.ly'</code>	-s	-b	-ss/-x	-bb
<code>'italiano.ly'</code>	-d	-b	-dd	-bb
<code>'norsk.ly'</code>	-iss/-is	-ess/-es	-ississ/-isis	-essess/-eses
<code>'portugues.ly'</code>	-s	-b	-ss	-bb
<code>'suomi.ly'</code>	-is	-es	-isis	-eses
<code>'svenska.ly'</code>	-iss	-ess	-ississ	-essess
<code>'vlaams.ly'</code>	-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 File	semi-sharp	semi-flat	sesqui-sharp	sesqui-flat
‘nederlands.ly’	-ih	-eh	-isih	-eseh
‘arabic.ly’	-sd	-sb	-dsd	-bsb
‘deutsch.ly’	-ih	-eh	-isih	-eseh
‘english.ly’	-qs	-qf	-tqs	-tqf
‘espanol.ly’	-cs	-cb	-tcs	-tcb
‘italiano.ly’	-sd	-sb	-dsd	-bsb
‘portugues.ly’	-sqt	-bqt	-stqt	-btqt

See also

Music Glossary: [Section “Pitch names” in *Music Glossary*](#).

Snippets: [Section “Pitches” in *Snippets*](#).

Non-Western note names and accidentals

Many non-Western musics (and some Western folk and traditional musics) employ alternative or extended tuning systems that do not fit readily into standard classical notation.

In some cases standard notation is still used, with the pitch differences being implicit. For example, Arabic music is notated with standard semitone and quarter-tone accidentals, with the precise pitch alterations being determined by context. Others require extended or unique notations.

Turkish classical music, or Ottoman music, employs melodic forms known as *makamlar*, whose intervals are based on 1/9 divisions of the whole tone. From a modern notational point of view, it is convenient to use the standard Western staff notes (c, d, e, ...) with special accidentals unique to Turkish music. These accidentals are defined in ‘*makam.ly*’ (to locate this file on your system, see [Section “Other sources of information” in *Learning Manual*](#)). The following table gives their names, the accidental suffix that must be added to notes, and their pitch alteration as a fraction of one whole tone.

Accidental name	suffix	pitch alteration
büyük mücenneb (sharp)	-bm	+8/9
küçük mücenneb (sharp)	-k	+5/9
bakiye (sharp)	-b	+4/9
koma (sharp)	-c	+1/9
koma (flat)	-fc	-1/9
bakiye (flat)	-fb	-4/9
küçük mücenneb (flat)	-fk	-5/9
büyük mücenneb (flat)	-fbm	-8/9

For further information on Turkish classical music and makamlar, see [Section 2.10.2 \[Turkish classical music\], page 327](#).

Selected Snippets

Makam example

Makam is a type of melody from Turkey using 1/9th-tone microtonal alterations. Consult the initialization file ‘*ly/makam.ly*’ for details of pitch names and alterations.

```
% Initialize makam settings
\include "makam.ly"

\relative c' {
  \set Staff.keySignature = #`((6 . ,(- KOMA)) (3 . ,BAKIYE))
  c4 cc db fk
  gbm4 gfc gfb efk
  fk4 db cc c
}
```



1.1.2 Changing multiple pitches

This section discusses how to modify pitches.

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 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 `\octaveCheck 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
  \octaveCheck c'
  e2 f
}
```



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


```

      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.

```

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

```



`\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 [\[Instrument transpositions\]](#), page 19.

Selected Snippets

Transposing pitches with minimum accidentals ("Smart" transpose)

This example uses some Scheme code to enforce enharmonic modifications for notes in order to have the minimum number of accidentals. In this case, the following rules apply:

Double accidentals should be removed

B sharp -> C

E sharp -> F

C flat -> B

F flat -> E

In this manner, the most natural enharmonic notes are chosen.

```

#(define (naturalize-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 (naturalize 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) (naturalize x)) es)))
    (if (ly:music? e)
        (ly:music-set-property!
         music 'element
         (naturalize e)))
    (if (ly:pitch? p)
        (begin
         (set! p (naturalize-pitch p))
         (ly:music-set-property! music 'pitch p)))
    music))

naturalizeMusic =
#(define-music-function (parser location m)
  (ly:music?)
  (naturalize m))

music = \relative c' { c4 d e g }

\score {
  \new Staff {
    \transpose c ais { \music }
    \naturalizeMusic \transpose c ais { \music }
    \transpose c deses { \music }
    \naturalizeMusic \transpose c deses { \music }
  }
}

```

```
\layout { }
}
```



See also

Notation Reference: [\[Relative octave entry\]](#), page 2, [\[Instrument transpositions\]](#), page 19.

Snippets: [Section “Pitches” in *Snippets*](#).

Internals Reference: [Section “TransposedMusic” in *Internals Reference*](#).

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.

Clef

The clef may be altered. Middle C is shown in every example. The following clef names can (but do not need to) be enclosed in quotes.

```
\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

\break

\clef G % synonym for treble
c2 c
\clef F % synonym for bass
c2 c
\clef C % synonym for alto
c2 c

```



By adding `_8` or `^8` to the clef name, the clef is transposed one octave down or up respectively, and `_15` and `^15` transpose by two octaves. Other integers can be used if required. Clef names containing non-alphabetic characters must be enclosed in quotes

```

\clef treble
c2 c
\clef "treble_8"
c2 c
\clef "bass^15"
c2 c
\clef "alto_2"
c2 c
\clef "G_8"
c2 c
\clef "F^5"
c2 c

```



Some special purpose clefs are described in [\[Mensural clefs\]](#), page 302, [\[Gregorian clefs\]](#), page 308, [\[Default tablatures\]](#), page 233, and [\[Custom tablatures\]](#), page 239.

Selected Snippets

Tweaking clef properties

The command `\clef "treble_8"` is equivalent to setting `clefGlyph`, `clefPosition` (which controls the vertical position of the clef), `middleCPosition` and `clefOctavation`. A clef is printed when any of the properties except `middleCPosition` 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 center line, positive numbers displacing upwards, counting one for each line and space. The `clefOctavation` value would normally be set to 7, -7, 15 or -15, but other values are valid.

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 `Staff` property `explicitClefVisibility` to the value `end-of-line-invisible`. The default behavior 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.

```
\layout { ragged-right = ##t }

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

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

  \set Staff.clefGlyph = #"clefs.F"
  \set Staff.clefPosition = #2
```

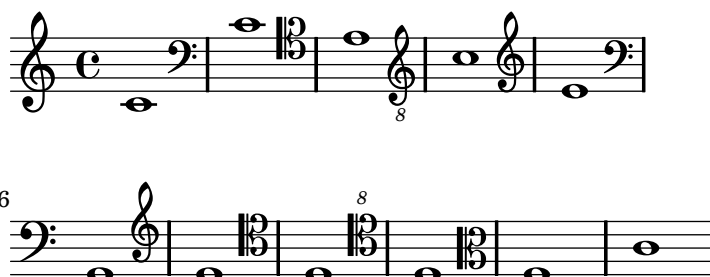
```

c'1
\set Staff.clefGlyph = #"clefs.G"
c'1
\set Staff.clefGlyph = #"clefs.C"
c'1
\set Staff.clefOctavation = #7
c'1
\set Staff.clefOctavation = #0
\set Staff.clefPosition = #0
c'1

% Return to the normal clef:

\set Staff.middleCPosition = #0
c'1
}

```



See also

Notation Reference: [Mensural clefs], page 302, [Gregorian clefs], page 308, [Default tablatures], page 233, [Custom tablatures], page 239.

Snippets: Section “Pitches” in *Snippets*.

Internals Reference: Section “Clef_engraver” in *Internals Reference*, Section “Clef” in *Internals Reference*, Section “OctavateEight” in *Internals Reference*, Section “clef-interface” in *Internals Reference*.

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 Section “Accidentals and key signatures” in *Learning Manual*.

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. The key signature may be altered:

```
\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

Preventing natural signs from being printed when the key signature changes

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

```
\relative c' {
  \key d \major
  a4 b cis d
  \key g \minor
  a4 bes c d
  \set Staff.printKeyCancellation = ##f
  \key d \major
  a4 b cis d
  \key g \minor
  a4 bes c d
}
```



Non-traditional key signatures

The commonly used `\key` command sets the `keySignature` property, in the **Staff** context.

To create non-standard key signatures, set 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.)

Alternatively, for each item in the list, using the more concise format `(step . alter)` 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 . 6) . ,FLAT)
                                ((0 . 5) . ,FLAT)
                                ((0 . 3) . ,SHARP))

  c4 d e fis
  aes4 bes c2
}
```



```
\set Staff.ottavation = #"Text"
c''1
}
```



See also

Music Glossary: [Section “ottavation” in *Music Glossary*](#).

Snippets: [Section “Pitches” in *Snippets*](#).

Internals Reference: [Section “Ottava-spanner-engraver” in *Internals Reference*](#), [Section “OttavaBracket” in *Internals Reference*](#), [Section “ottava-bracket-interface” in *Internals Reference*](#).

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 [\[Quoting other voices\]](#), page 157.

`\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 = "violin" {
    \relative c'' {
      \set Staff.instrumentName = #"Vln"
      \set Staff.midiInstrument = #"violin"
      % not strictly 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

      \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.

```

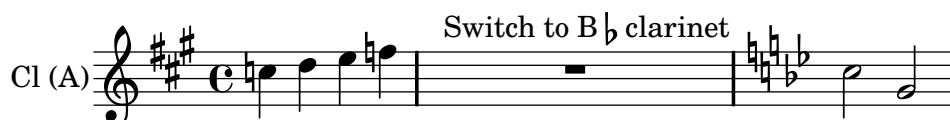
\set Staff.instrumentName = #"Cl (A)"
\key a \major
\transposition a
c d e f
\textLengthOn
s1*0^\markup { Switch to B\flat clarinet }
R1

```

```

\key bes \major
\transposition bes
c2 g

```



See also

Music Glossary: [Section “concert pitch”](#) in *Music Glossary*, [Section “transposing instrument”](#) in *Music Glossary*.

Notation Reference: [\[Quoting other voices\]](#), page 157, [\[Transpose\]](#), page 10.

Snippets: [Section “Pitches”](#) in *Snippets*.

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:

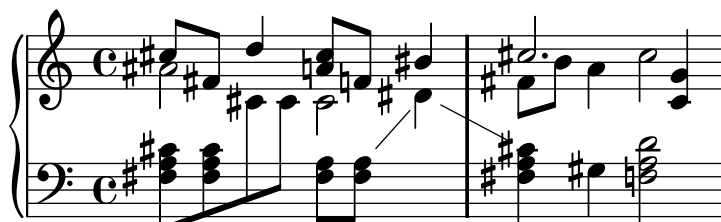
```

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>8 <fis a cis>
      \change Staff = up
      cis' cis
      \change Staff = down
      <fis, a> <fis a>
      \showStaffSwitch
      \change Staff = up
      dis'4 |
      \change Staff = down
      <fis, a cis>4 gis <f a d>2 |
    }
  }
}

\new PianoStaff {
  <<
    \context Staff = "up" {
      #(set-accidental-style 'default)
      \musicA
    }
    \context Staff = "down" {
      #(set-accidental-style 'default)
      \musicB
    }
  >>
}

```

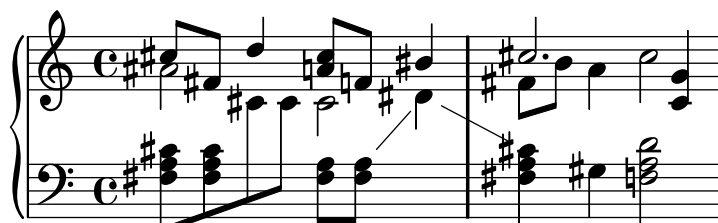


Note that the last lines of this example can be replaced by the following, as long as the same accidental style should be used in both staves.

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

default

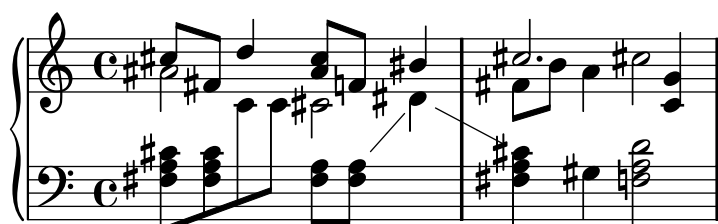
This is the default typesetting behavior. It corresponds to eighteenth-century common practice: accidentals are remembered to the end of the measure in which they occur and only in their own octave. Thus, in the example below, no natural signs are printed before the **b** in the second measure or the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



voice

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

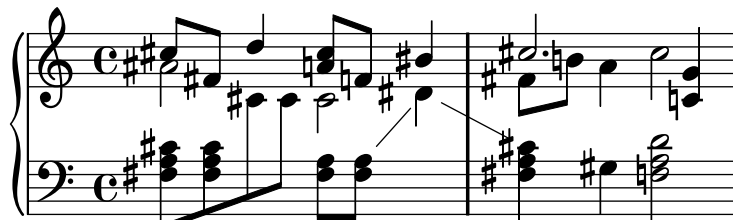
As a result, accidentals from one voice do not get canceled in other voices, which is often an unwanted result: in the following example, it is hard to determine whether the second **a** should be played natural or sharp. The **voice** option should therefore be used only if the voices are to be read 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

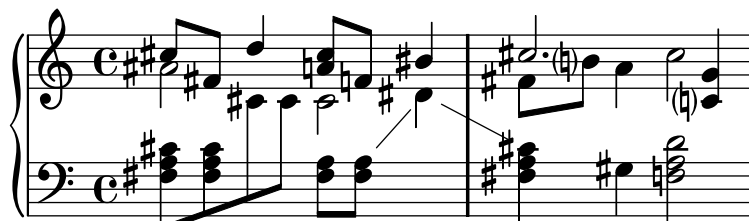
This rule corresponds to the common practice in the twentieth century. It prints the same accidentals as **default**, with two exceptions that serve to avoid ambiguity: after temporary accidentals, cancellation marks are printed also in the following measure (for notes in the same octave) and, in the same measure, for notes in other

octaves. Hence the naturals before the **b** and the **c** in the second measure of the upper staff:



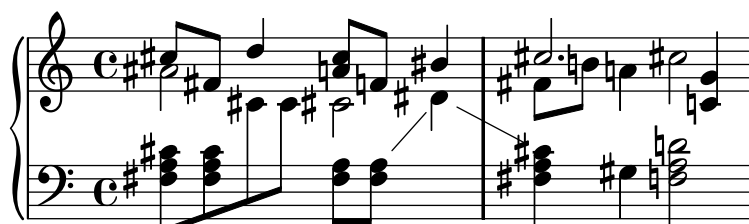
modern-cautionary

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`.



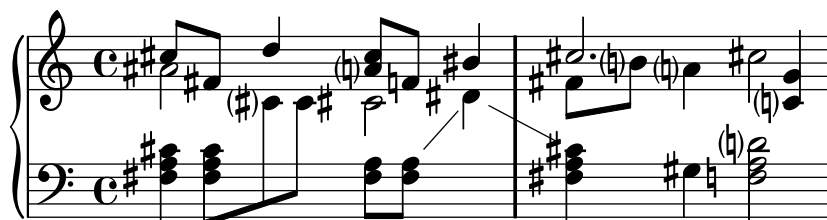
modern-voice

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:

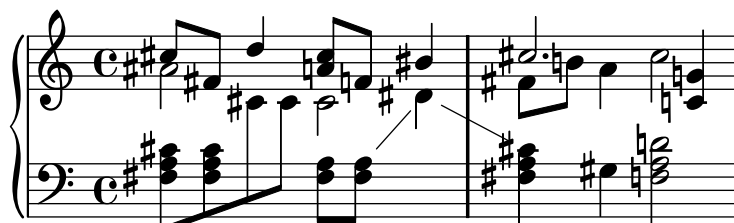


modern-voice-cautionary

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.

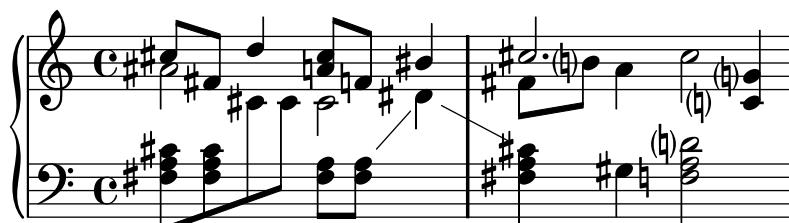


This rule reflects twentieth-century practice for piano notation. Its behavior is very similar to `modern` style, but here accidentals also get canceled across the staves in the same `GrandStaff` or `PianoStaff`, hence all the cancellations of the final notes. This accidental style applies to the current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



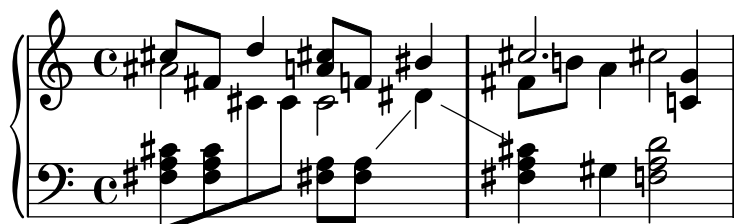
piano-cautionary

This is the same as `piano` but with the extra accidentals typeset as cautionaries.



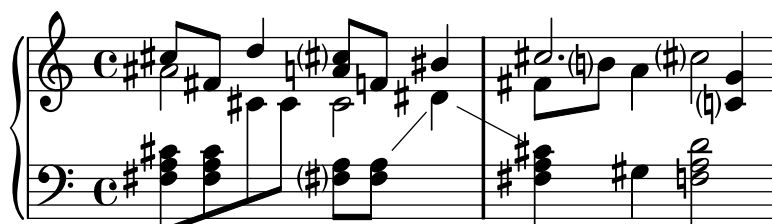
neo-modern

This rule reproduces a common practice in contemporary music: accidentals are printed like with `modern`, but they are printed again if the same note appears later in the same measure – except if the note is immediately repeated.



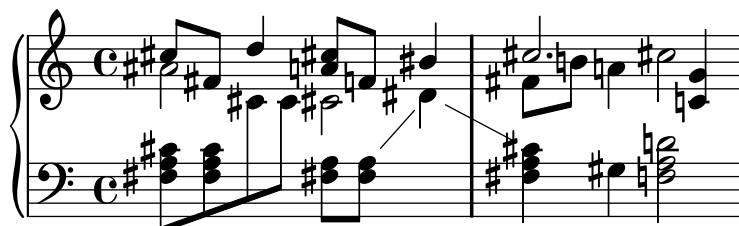
neo-modern-cautionary

This rule is similar to `neo-modern`, but the extra accidentals are printed as cautionary accidentals.

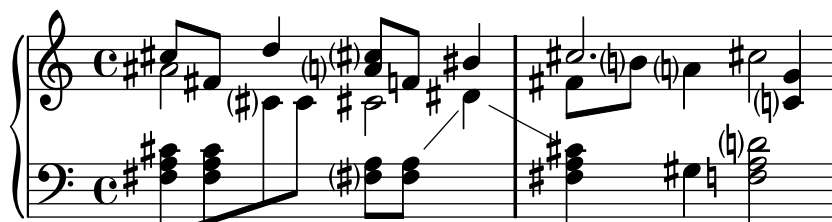


neo-modern-voice

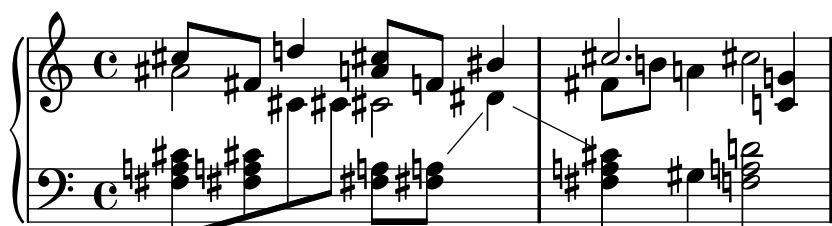
This rule is used for multivoice accidentals to be read both by musicians playing one voice and musicians playing all voices. Accidentals are typeset for each voice as with `neo-modern`, but they are canceled across voices in the same `Staff`.

**neo-modern-voice-cautionary**

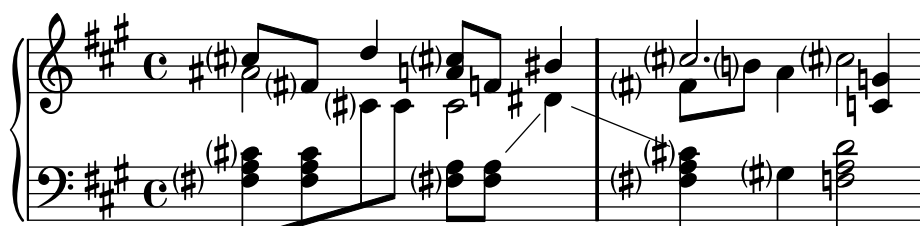
This rule is similar to **neo-modern-voice**, but the extra accidentals are printed as cautionary accidentals.

**dodecaphonic**

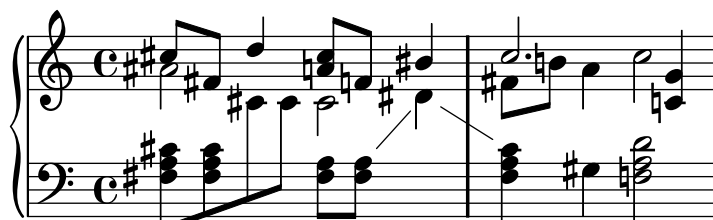
This rule reflects a practice introduced by composers at the beginning of the 20th century, in an attempt to abolish the hierarchy between natural and non-natural notes. With this style, *every* note gets an accidental sign, including natural signs.

**teaching**

This rule is intended for students, and makes it easy to create scale sheets with automatically created cautionary accidentals. Accidentals are printed like with **modern**, but cautionary accidentals are added for all sharp or flat tones specified by the key signature, except if the note is immediately repeated.

**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 came before in the music. Unlike **dodecaphonic**, this rule never prints any naturals.



Selected Snippets

Dodecaphonic-style accidentals for each note including naturals

In early 20th century works, starting with Schoenberg, 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.

```
\markup {
  This snippet is deprecated as of version 2.12 and
  will be removed from the documentation in 2.14.
}
```

This snippet is deprecated as of version 2.12 and will be removed from the documentation in 2.14

See also

Snippets: [Section “Pitches” in *Snippets*](#).

Internals Reference: [Section “Accidental” in *Internals Reference*](#), [Section “Accidental-engraver” in *Internals Reference*](#), [Section “GrandStaff” in *Internals Reference*](#), [Section “PianoStaff” in *Internals Reference*](#), [Section “Staff” in *Internals Reference*](#), [Section “AccidentalSuggestion” in *Internals Reference*](#), [Section “AccidentalPlacement” in *Internals Reference*](#), [Section “accidental-suggestion-interface” in *Internals Reference*](#).

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.

Cautionary cancellation of accidentals is done by looking at previous measure. However, in the **\alternative** block following a **\repeat volta N** section, one would expect the cancellation being calculated using the previous *played* measure, not previous *printed* measure. In the following example, the natural **c** in the second alternative does not need a natural sign:



The following work-around can be used: define a function that locally changes the accidental style to forget:

```
forget = #(define-music-function (parser location music) (ly:music?) #{
  #(set-accidental-style 'forget)
  $music
  #(set-accidental-style 'modern)
#})
{
  #(set-accidental-style 'modern)
  \time 2/4
  \repeat volta 2 {
    c'2
  }
  \alternative {
    cis'
    \forget c'
  }
}
```



Ambitus

The term *ambitus* (pl. *ambitus*) 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. Ambitus are printed on vocal parts so that performers can easily determine if it matches their capabilities.

Ambitus 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,1
}
```



Selected Snippets

Adding ambitus per voice

Ambitus can be added per voice. In this 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
  }
}>>
```



Ambitus with multiple voices

Adding the `Ambitus_engraver` to the `Staff` context creates a single ambitus per staff, even in the case of staves with multiple voices.

```
\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: [Section “ambitus” in *Music Glossary*](#).

Snippets: [Section “Pitches” in *Snippets*](#).

Internals Reference: [Section “Ambitus_engraver” in *Internals Reference*](#), [Section “Voice” in *Internals Reference*](#), [Section “Staff” in *Internals Reference*](#), [Section “Ambitus” in *Internals Reference*](#), [Section “AmbitusAccidental” in *Internals Reference*](#), [Section “AmbitusLine” in *Internals Reference*](#), [Section “AmbitusNoteHead” in *Internals Reference*](#), [Section “ambitus-interface” in *Internals Reference*](#).

Known issues and warnings

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

1.1.4 Note heads

This section suggests ways of altering note heads.

Special note heads

The appearance of note heads may be altered:

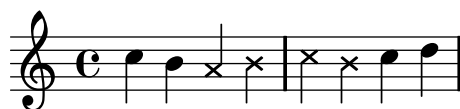
```
c4 b
\override NoteHead #'style = #'cross
c4 b
\revert NoteHead #'style
a b
\override NoteHead #'style = #'harmonic
a b
\revert NoteHead #'style
c4 d e f
```



To see all note head styles, see [Section A.7 \[Note head styles\]](#), page 473.

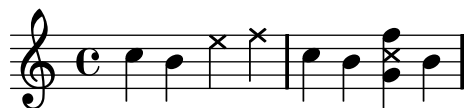
The `cross` style is used to represent a variety of musical intentions. The following generic predefined commands modify the note head in both staff and tablature contexts and can be used to represent any musical meaning:

```
c4 b
\xNotesOn
a b c4 b
\xNotesOff
c4 d
```



The music function form of this predefined command may be used inside and outside chords to generate crossed note heads in both staff and tablature contexts:

```
c4 b
\xNote { e f }
c b < g \xNote c f > b
```



As synonyms for `\xNote`, `\xNotesOn` and `\xNotesOff`, `\deadNote`, `\deadNotesOn` and `\deadNotesOff` can be used. The term *dead note* is commonly used by guitarists.

There is also a shorthand for diamond shapes which can be used only inside chords:

```
<c f\harmonic>2 <d a'\harmonic>4 <c g'\harmonic>
```



Predefined commands

`\harmonic`, `\xNotesOn`, `\xNotesOff`, `\xNote`.

See also

Snippets: [Section “Pitches” in *Snippets*](#).

Notation Reference: [Section A.7 \[Note head styles\]](#), page 473, [\[Chorded notes\]](#), page 121, [\[Indicating harmonics and dampened notes\]](#), page 262.

Internals Reference: [Section “note-event” in *Internals Reference*](#), [Section “Note_heads_engraver” in *Internals Reference*](#), [Section “Ledger_line_engraver” in *Internals Reference*](#), [Section “NoteHead” in *Internals Reference*](#), [Section “LedgerLineSpanner” in *Internals Reference*](#), [Section “note-head-interface” in *Internals Reference*](#), [Section “ledger-line-spanner-interface” in *Internals Reference*](#).

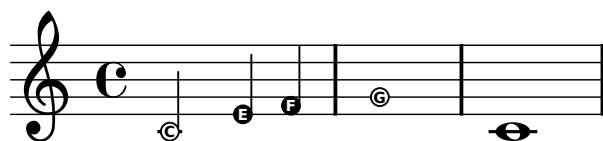
Easy notation note heads

The ‘easy play’ note head includes a note name inside the head. It is used in music for beginners. To make the letters readable, it should be printed in a large font size. To print with a larger font, see [Section 4.2.1 \[Setting the staff size\]](#), page 365.

```

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

```



Predefined commands

`\easyHeadsOn`, `\easyHeadsOff`.

Selected Snippets

Numbers as easy note heads

Easy notation note heads use the `note-names` property of the `NoteHead` object to determine what appears inside the note head. By overriding this property, it is possible to print numbers representing the scale-degree.

A simple engraver can be created to do this for every note head object it sees.

```
#(define Ez_numbers_engraver
  (list
    (cons 'acknowledgers
      (list
        (cons 'note-head-interface
          (lambda (engraver grob source-engraver)
            (let* ((context (ly:translator-context engraver))
                  (tonic-pitch (ly:context-property context 'tonic))
                  (tonic-name (ly:pitch-notename tonic-pitch))
                  (grob-pitch
                    (ly:event-property (event-cause grob) 'pitch))
                  (grob-name (ly:pitch-notename grob-pitch))
                  (delta (modulo (- grob-name tonic-name) 7))
                  (note-names
                    (make-vector 7 (number->string (1+ delta))))))
              (ly:grob-set-property! grob 'note-names note-names))))))

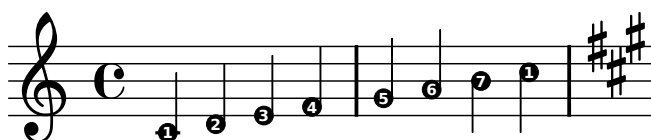
#(set-global-staff-size 26)

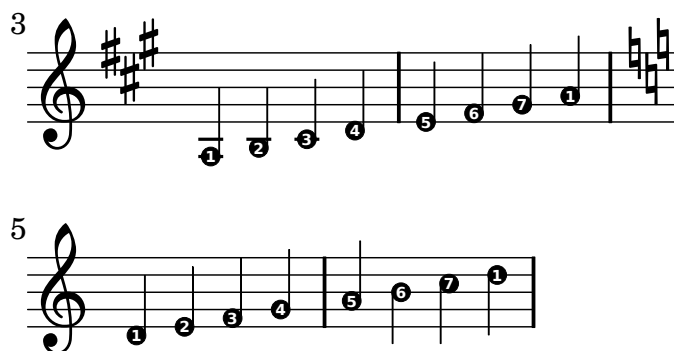
\layout {
  ragged-right = ##t
  \context {
    \Voice
    \consists \Ez_numbers_engraver
  }
}

\relative c' {
  \easyHeadsOn
  c4 d e f
  g4 a b c \break

  \key a \major
  a,4 b cis d
  e4 fis gis a \break

  \key d \dorian
  d,4 e f g
  a4 b c d
}
```





See also

Notation Reference: [Section 4.2.1 \[Setting the staff size\]](#), page 365.

Snippets: [Section “Pitches”](#) in *Snippets*.

Internals Reference: [Section “note-event”](#) in *Internals Reference*, [Section “Note_heads_engraver”](#) in *Internals Reference*, [Section “NoteHead”](#) in *Internals Reference*, [Section “note-head-interface”](#) in *Internals Reference*.

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 in Sacred Harp, Southern Harmony, and Aiken (Christian Harmony) styles:

```
\aikenHeads
c, d e f g2 a b1 c \break
\sacredHarpHeads
c,4 d e f g2 a b1 c \break
\southernHarmonyHeads
c,4 d e f g2 a b1 c \break
```



Shapes are typeset according to the step in the scale, where the base of the scale is determined by the `\key` command. When writing in a minor key, the scale step can be determined from the relative major:

```
\key a \minor
\aikenHeads
a b c d e2 f g1 a \break
\aikenHeadsMinor
a,4 b c d e2 f g1 a \break
\sacredHarpHeadsMinor
a,2 b c d \break
```



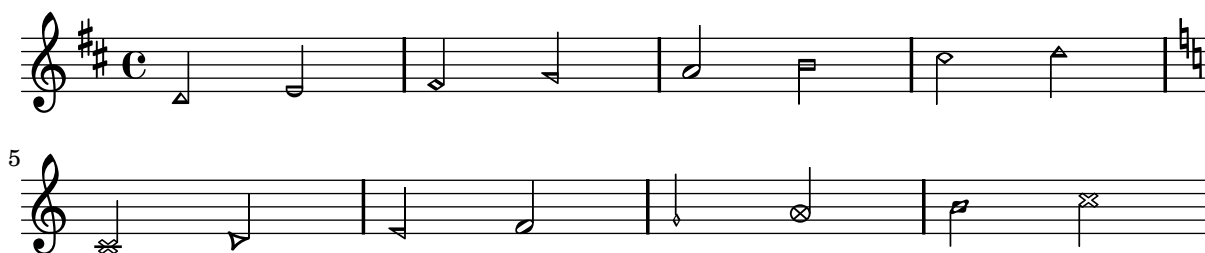
```

}

\break

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

```



To see all note head styles, see [Section A.7 \[Note head styles\]](#), page 473.

See also

Snippets: [Section “Pitches” in *Snippets*](#).

Notation Reference: [Section A.7 \[Note head styles\]](#), page 473.

Internals Reference: [Section “note-event” in *Internals Reference*](#), [Section “Note_heads_engraver” in *Internals Reference*](#), [Section “NoteHead” in *Internals Reference*](#), [Section “note-head-interface” in *Internals Reference*](#).

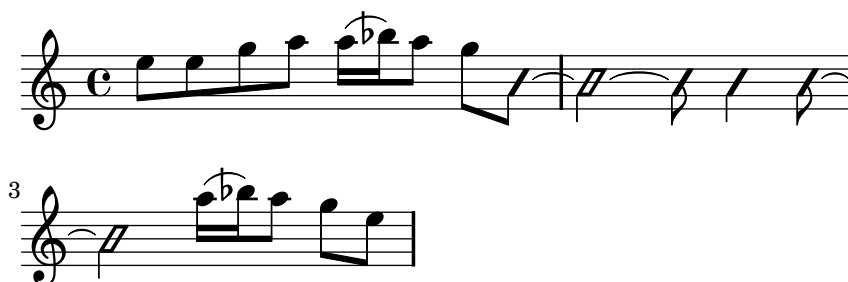
Improvisation

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"
} {
  e8 e g a a16( bes) a8 g
  \improvisationOn
  e8 ~
  e2 ~ e8 f4 f8 ~
  f2
  \improvisationOff
  a16( bes) a8 g e
}

```



Predefined commands

`\improvisationOn`, `\improvisationOff`.

See also

Snippets: [Section “Pitches” in *Snippets*](#).

Internals Reference: [Section “Pitch_squash_engraver” in *Internals Reference*](#), [Section “Voice” in *Internals Reference*](#), [Section “RhythmicStaff” in *Internals Reference*](#).

1.2 Rhythms



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

1.2.1 Writing rhythms

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 128th 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 c128 c128
```



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 c128 c128
```



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 2.9 \[Ancient notation\]](#), [page 298](#).

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 [\[Ties\]](#), [page 42](#).

For ways of specifying durations for the syllables of lyrics and ways of aligning lyrics to notes, see [Section 2.1 \[Vocal music\]](#), [page 198](#).

Optionally, notes can be spaced strictly proportionately to their duration. For details of this and other settings which control proportional notation, see [Section 4.5.5 \[Proportional notation\]](#), [page 388](#).

Dots are normally moved up to avoid staff lines, except in polyphonic situations. Dots may be manually placed above or below the staff; see [Section 5.4.2 \[Direction and placement\]](#), [page 419](#).

Predefined commands

```
\autoBeamOn, \autoBeamOff, \dotsUp, \dotsDown, \dotsNeutral.
```

Selected Snippets

Changing the number of augmentation dots per note

This code demonstrates how to change the number of augmentation dots on a single note.

```
\relative c' {
  c4.. a16 r2 |
  \override Dots #'dot-count = #4
  c4.. a16 r2 |
  \override Dots #'dot-count = #0
  c4.. a16 r2 |
  \revert Dots #'dot-count
  c4.. a16 r2 |
}
```



See also

Music Glossary: Section “breve” in *Music Glossary*, Section “longa” in *Music Glossary*, Section “maxima” in *Music Glossary*, Section “note value” in *Music Glossary*, Section “Duration names notes and rests” in *Music Glossary*.

Notation Reference: [Automatic beams], page 63, [Ties], page 42, [Stems], page 170, Section 1.2.1 [Writing rhythms], page 35, Section 1.2.2 [Writing rests], page 45, Section 2.1 [Vocal music], page 198, Section 2.9 [Ancient notation], page 298, Section 4.5.5 [Proportional notation], page 388.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “Dots” in *Internals Reference*, Section “DotColumn” in *Internals Reference*.

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.

Tuplets

Tuplets are made from a music expression by multiplying all the durations with a fraction:

```
\times fraction { music }
```

The duration of *music* 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 triplet is the triplet in which 3 notes have the duration 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 }
```



Tuplet brackets may be manually placed above or below the staff; see Section 5.4.2 [Direction and placement], page 419.

Tuplets may be nested:

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



Modifying nested triplets which begin at the same musical moment must be done with `\tweak`.

To modify the duration of notes without printing a tuplet bracket, see [Scaling durations], page 41.

Predefined commands

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

Selected Snippets

Entering several tuplets using only one `\times` command

The property `tupletSpannerDuration` sets how long each of the tuplets contained within the brackets after `\times` should last. Many consecutive tuplets can then be placed within a single `\times` expression, thus saving typing.

In the example, two triplets are shown, while `\times` was entered only once.

Read the relevant sections of the Notation Reference for more information about `ly:make-moment`.

```
\relative c' {
  \time 2/4
  \set tupletSpannerDuration = #(ly:make-moment 1 4)
  \times 2/3 { c8 c c c c c }
}
```



Changing the tuplet number

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.

```
\relative c'' {
  \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 #'stencil = ##f
  \times 2/3 { c8 c c }
}
```



Non-default tuplet numbers

LilyPond also provides formatting functions to print tuplet numbers different than the actual fraction, as well as to append a note value to the tuplet number or tuplet fraction.

```
\relative c'' {
  \once \override TupletNumber #'text =
    #(tuplet-number::non-default-tuplet-denominator-text 7)
  \times 2/3 { c4. c4. c4. c4. }
  \once \override TupletNumber #'text =
    #(tuplet-number::non-default-tuplet-fraction-text 12 7)
  \times 2/3 { c4. c4. c4. c4. }
  \once \override TupletNumber #'text =
    #(tuplet-number::append-note-wrapper
      (tuplet-number::non-default-tuplet-fraction-text 12 7) "8")
}
```

```

\times 2/3 { c4. c4. c4. c4. }

\once \override TupletNumber #'text =
  #(tuplet-number::append-note-wrapper
    tuplet-number::calc-denominator-text "4")
\times 2/3 { c8 c8 c8 c8 c8 c8 }
\once \override TupletNumber #'text =
  #(tuplet-number::append-note-wrapper
    tuplet-number::calc-fraction-text "4")
\times 2/3 { c8 c8 c8 c8 c8 c8 }

\once \override TupletNumber #'text =
  #(tuplet-number::fraction-with-notes "4." "8")
\times 2/3 { c4. c4. c4. c4. }
\once \override TupletNumber #'text =
  #(tuplet-number::non-default-fraction-with-notes 12 "8" 4 "4")
\times 2/3 { c4. c4. c4. c4. }
}

```



Controlling tuplet bracket visibility

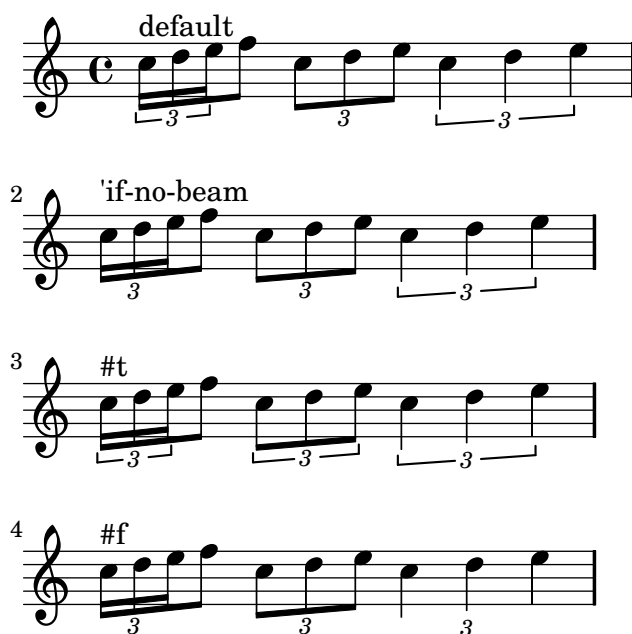
The default behavior of tuplet-bracket visibility is to print a bracket unless there is a beam of the same length as the tuplet. To control the visibility of tuplet brackets, set the property 'bracket-visibility to either `#t` (always print a bracket), `#f` (never print a bracket) or `#if-no-beam` (only print a bracket if there is no beam).

```

music = \relative c' {
  \times 2/3 { c16[ d e ] f8]
  \times 2/3 { c8 d e }
  \times 2/3 { c4 d e }
}

\new Voice {
  \relative c' {
    << \music s4^"default" >>
    \override TupletBracket #'bracket-visibility = #if-no-beam
    << \music s4^"if-no-beam" >>
    \override TupletBracket #'bracket-visibility = ##t
    << \music s4^"#t" >>
    \override TupletBracket #'bracket-visibility = ##f
    << \music s4^"#f" >>
  }
}

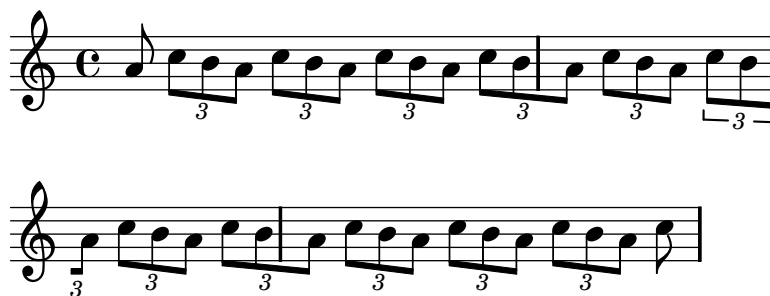
```



Permitting line breaks within beamed triplets

This artificial example shows how both manual and automatic line breaks may be permitted to within a beamed triplet. Note that such off-beat triplets have to be beamed manually.

```
\layout {
  \context {
    \Voice
    % Permit line breaks within triplets
    \remove "Forbid_line_break_engraver"
    % Allow beams to be broken at line breaks
    \override Beam #'breakable = ##t
  }
}
\relative c'' {
  a8
  \repeat unfold 5 { \times 2/3 { c[ b a] } }
  % Insert a manual line break within a triplet
  \times 2/3 { c[ b \bar "" \break a] }
  \repeat unfold 5 { \times 2/3 { c[ b a] } }
  c8
}
```



See also

Music Glossary: [Section “triplet” in *Music Glossary*](#), [Section “tuplet” in *Music Glossary*](#), [Section “polymetric” in *Music Glossary*](#).

Learning Manual: [Section “Tweaking methods” in *Learning Manual*](#).

Notation Reference: [\[Time administration\]](#), page 89, [\[Scaling durations\]](#), page 41, [Section 5.3.4 \[The tweak command\]](#), page 416, [\[Polymetric notation\]](#), page 57.

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “TupletBracket” in *Internals Reference*](#), [Section “TupletNumber” in *Internals Reference*](#), [Section “TimeScaledMusic” in *Internals Reference*](#).

Known issues and warnings

Grace notes may be placed within tuplet brackets, *except* when a staff begins with a grace note followed by a tuplet. In this particular case, the grace note must be placed before the `\times` command to avoid errors.

When using a tuplet at the beginning of a piece with a `\tempo` mark, the music must be explicitly entered in a `\new Voice` block, as discussed in [Section “Voices contain music” in *Learning Manual*](#).

Scaling durations

You can alter the duration of single notes, rests or chords by a fraction N/M by appending `*N/M` (or `*N` if M is 1) to the duration. 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 `*L*M/N`.

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

```
\time 2/4
% Alter durations to triplets
a4*2/3 gis4*2/3 a4*2/3
% Normal durations
a4 a4
% Double the duration of chord
<a d>4*2
% Duration of quarter, appears like sixteenth
b16*4 c4
```



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

Longer stretches of music may be compressed by a fraction in the same way, as if every note, chord or rest had the fraction as a multiplier. This leaves the appearance of the music unchanged but the internal duration of the notes will be multiplied by the fraction num/den . The spaces around the dot are required. Here is an example showing how music can be compressed and expanded:

```
\time 2/4
% Normal durations
```

```

<c a>4 c8 a
% Scale music by *2/3
\scaleDurations #'(2 . 3) {
  <c a f>4. c8 a f
}
% Scale music by *2
\scaleDurations #'(2 . 1) {
  <c' a>4 c8 b
}

```



One application of this command is in polymetric notation, see [\[Polymetric notation\]](#), page 57.

See also

Notation Reference: [\[Tuplets\]](#), page 37, [\[Invisible rests\]](#), page 47, [\[Polymetric notation\]](#), page 57.

Snippets: [Section “Rhythms” in Snippets](#).

Ties

A tie connects two adjacent note heads of the same pitch. The tie in effect extends the duration 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 (~).

```
a2 ~ a
```



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:

```

\relative c' {
  r8 c8 ~ c2 r4 |
  r8^"not" c2 ~ c8 r4
}

```



If you need to tie many notes across bar lines, it may be easier to use automatic note splitting, see [\[Automatic note splitting\]](#), page 59. 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 specify the repeated tie as follows:

```
\repeat volta 2 { c g <c e>2 ~ }
\alternative {
  % First alternative: following note is tied normally
  { <c e>2. r4 }
  % Second alternative: following note has a repeated tie
  { <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 as follows:

```
<c f g>1\laissezVibrer
```



Ties may be manually placed above or below the staff; see [Section 5.4.2 \[Direction and placement\]](#), page 419.

Ties may be made dashed, dotted, or a combination of solid and dashed.

```
\tieDotted
c2 ~ c
\tieDashed
c2 ~ c
\tieHalfDashed
c2 ~ c
\tieHalfSolid
c2 ~ c
\tieSolid
c2 ~ c
```



Custom dash patterns can be specified:

```

\tieDashPattern #0.3 #0.75
c2 ~ c
\tieDashPattern #0.7 #1.5
c2 ~ c
\tieSolid
c2 ~ c

```



Dash pattern definitions for ties have the same structure as dash pattern definitions for slurs. For more information about complex dash patterns, see the snippets under [\[Slurs\]](#), page 99.

Predefined commands

```

\tieUp, \tieDown, \tieNeutral, \tieDotted, \tieDashed, \tieDashPattern,
\tieHalfDashed, \tieHalfSolid, \tieSolid.

```

Selected Snippets

Using ties with arpeggios

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 `#t`. 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.

```

\relative c' {
  \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
  g8 ~ c g2
}

```



Engraving ties manually

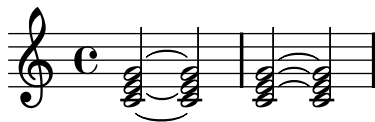
Ties may be engraved manually by changing the `tie-configuration` property of the `TieColumn` object. The first number indicates the distance from the center of the staff in staff-spaces, and the second number indicates the direction (1 = up, -1 = down).

```

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

```

}



See also

Music Glossary: [Section “tie” in *Music Glossary*](#), [Section “laissez vibrer” in *Music Glossary*](#).

Notation Reference: [\[Slurs\]](#), page 99, [\[Automatic note splitting\]](#), page 59.

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “LaissezVibrerTie” in *Internals Reference*](#), [Section “LaissezVibrerTieColumn” in *Internals Reference*](#), [Section “TieColumn” in *Internals Reference*](#), [Section “Tie” in *Internals Reference*](#).

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.

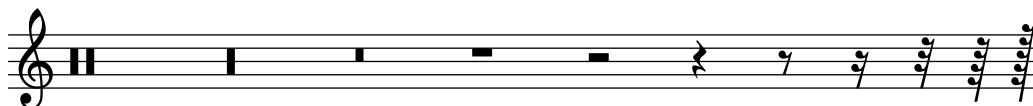
1.2.2 Writing rests

Rests are entered as part of the music in music expressions.

Rests

Rests are entered like notes with the note name `r`. Durations longer than a whole rest use the predefined commands shown:

```
\new Staff {
  % These two lines are just to prettify this example
  \time 16/1
  \override Staff.TimeSignature #'stencil = ##f
  % Print a maxima rest, equal to four breves
  r\maxima
  % Print a longa rest, equal to two breves
  r\longa
  % Print a breve rest
  r\breve
  r1 r2 r4 r8 r16 r32 r64 r128
}
```



Whole measure rests, centered in the 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 [\[Full measure rests\]](#), page 48.

To explicitly specify a rest’s vertical position, write a note followed by `\rest`. A rest of the duration of the note will be placed at the staff position where the note would appear. This allows for precise manual formatting of polyphonic music, since the automatic rest collision formatter will not move these rests.

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



Selected Snippets

Rest styles

Rests may be used in various styles.

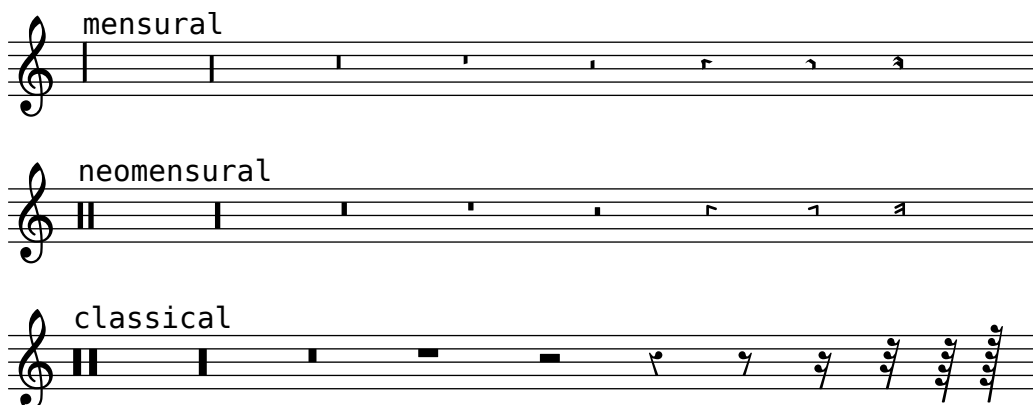
```
\layout {
  indent = 0
  \context {
    \Staff
    \remove "Time_signature_engraver"
  }
}

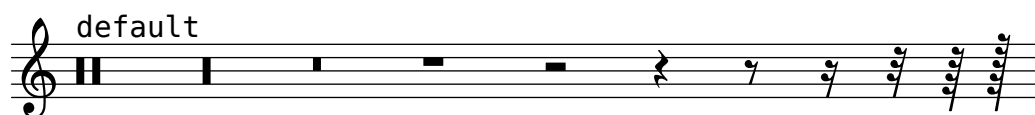
\new Staff \relative c {
  \cadenzaOn
  \override Staff.Rest #'style = #'mensural
  r\maxima\markup \typewriter { mensural }
  r\longa r\breve r1 r2 r4 r8 r16 s32 s64 s128 s128
  \bar ""

  \override Staff.Rest #'style = #'neomensural
  r\maxima\markup \typewriter { neomensural }
  r\longa r\breve r1 r2 r4 r8 r16 s32 s64 s128 s128
  \bar ""

  \override Staff.Rest #'style = #'classical
  r\maxima\markup \typewriter { classical }
  r\longa r\breve r1 r2 r4 r8 r16 r32 r64 r128 s128
  \bar ""

  \override Staff.Rest #'style = #'default
  r\maxima\markup \typewriter { default }
  r\longa r\breve r1 r2 r4 r8 r16 r32 r64 r128 s128
}
```





See also

Music Glossary: [Section “breve”](#) in *Music Glossary*, [Section “longa”](#) in *Music Glossary*, [Section “maxima”](#) in *Music Glossary*.

Notation Reference: [\[Full measure rests\]](#), page 48.

Snippets: [Section “Rhythms”](#) in *Snippets*.

Internals Reference: [Section “Rest”](#) in *Internals Reference*.

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).

Invisible rests

An invisible rest (also called a ‘spacer rest’) can be entered like a note with the note name `s`:

```
c4 c s c
s2 c
```



Spacer rests are available only in note mode and chord mode. In other situations, for example, when entering lyrics, the command `\skip` is used to skip a musical moment. `\skip` requires an explicit duration.

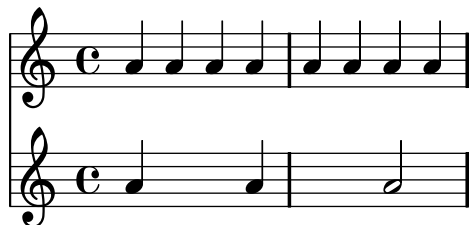
```
<<
{
  a2 \skip2 a2 a2
}
\new Lyrics {
  \lyricmode {
    foo2 \skip 1 bla2
  }
}
>>
```



Because `\skip` is a command, it does not affect the default durations of following notes, unlike `s`.

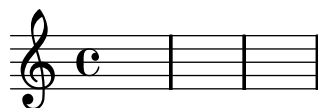
```
<<
{
  \repeat unfold 8 {a4}
}
{
  a4 \skip 2 a |
s2 a
```

```
}
>>
```



A spacer rest implicitly causes **Staff** and **Voice** contexts to be created if none exist, just like notes and rests do:

```
s1 s s
```



`\skip` simply skips musical time; it creates no output of any kind.

```
% This is valid input, but does nothing
\skip 1 \skip1 \skip 1
```

See also

Learning Manual: [Section “Visibility and color of objects”](#) in *Learning Manual*.

Notation Reference: [\[Hidden notes\]](#), page 168, [Section 5.4.7 \[Visibility of objects\]](#), page 426.

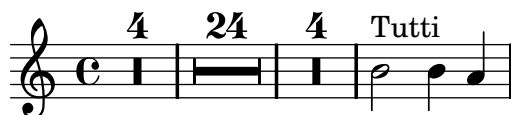
Snippets: [Section “Rhythms”](#) in *Snippets*.

Internals Reference: [Section “SkipMusic”](#) in *Internals Reference*.

Full measure rests

Rests for one or more full measures are entered like notes with the note name uppercase R:

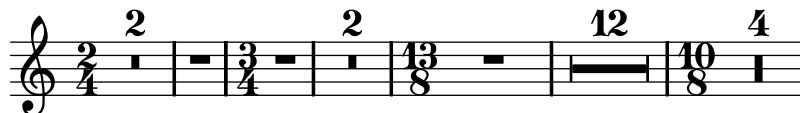
```
% Rest measures contracted to single measure
\compressFullBarRests
R1*4
R1*24
R1*4
b2^"Tutti" b4 a4
```



The duration of full-measure rests is identical to the duration notation used for notes. The duration in a multi-measure rest must always be an integral number of measure-lengths, so augmentation dots or fractions must often be used:

```
\compressFullBarRests
\time 2/4
R1 | R2 |
\time 3/4
R2. | R2.*2 |
\time 13/8
```

```
R1*13/8 | R1*13/8*12 |
\time 10/8
R4*5*4 |
```



A full-measure rest is printed as either a whole or breve rest, centered in the measure, depending on the time signature.

```
\time 4/4
R1 |
\time 6/4
R1*3/2 |
\time 8/4
R1*2 |
```



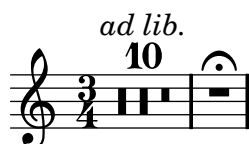
By default a multi-measure rest is expanded in the printed score to show all the rest measures explicitly. Alternatively, a multi-measure rest can be shown as a single measure containing a multi-measure rest symbol, with the number of measures of rest printed above the measure:

```
% Default behavior
\time 3/4 r2. | R2.*2 |
\time 2/4 R2 |
\time 4/4
% Rest measures contracted to single measure
\compressFullBarRests
r1 | R1*17 | R1*4 |
% Rest measures expanded
\expandFullBarRests
\time 3/4
R2.*2 |
```



Markups can be added to multi-measure rests. The predefined command `\fermataMarkup` is provided for adding fermatas.

```
\compressFullBarRests
\time 3/4
R2.*10^\markup { \italic "ad lib." }
R2.^{\fermataMarkup}
```



Note: Markups attached to a multi-measure rest are objects of type `MultiMeasureRestText`, not `TextScript`. Overrides must be directed to the correct object, or they will be ignored. See the following example.

```
% This fails, as the wrong object name is specified
\override TextScript #'padding = #5
R1^"wrong"
% This is correct and works
\override MultiMeasureRestText #'padding = #5
R1^"right"
```

right



When a multi-measure rest immediately follows a `\partial` setting, resulting bar-check warnings may not be displayed.

Predefined commands

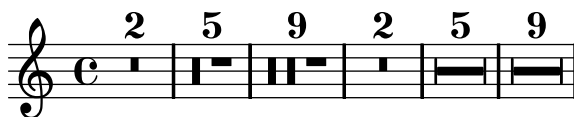
```
\textLengthOn,      \textLengthOff,      \fermataMarkup,      \compressFullBarRests,
\expandFullBarRests.
```

Selected Snippets

Changing form of multi-measure rests

If there are ten or fewer measures of rests, a series of longa and breve rests (called in German “Kirchenpausen” - church rests) is printed within the staff; otherwise a simple line is shown. This default number of ten may be changed by overriding the `expand-limit` property.

```
\relative c'' {
  \compressFullBarRests
  R1*2 | R1*5 | R1*9
  \override MultiMeasureRest #'expand-limit = #3
  R1*2 | R1*5 | R1*9
}
```



Positioning multi-measure rests

Unlike ordinary rests, there is no predefined command to change the staff position of a multi-measure rest symbol of either form by attaching it to a note. However, in polyphonic music multi-measure rests in odd-numbered and even-numbered voices are vertically separated. The positioning of multi-measure rests can be controlled as follows:

```
\relative c'' {
  % MMR - Multi-Measure Rest
  % MMRs by default are set under the fourth line
  R1
  % They can be moved with an override
  \override MultiMeasureRest #'staff-position = #-2
  R1
```

```

% A value of 0 is the default position;
% the following trick moves the rest to the center line
\override MultiMeasureRest #'staff-position = #-0.01
R1
% MMRs in odd-numbered voices are under the top line
<< { R1 } \\\ { a1 } >>
% MMRs in even-numbered voices are under the bottom line
<< { c1 } \\\ { R1 } >>
% They remain separated even in empty measures
<< { R1 } \\\ { R1 } >>
% This brings them together even though there are two voices
\compressFullBarRests
<<
  \revert MultiMeasureRest #'staff-position
  { R1*3 }
  \\\
  \revert MultiMeasureRest #'staff-position
  { R1*3 }
>>
}

```



Multi-measure rest markup

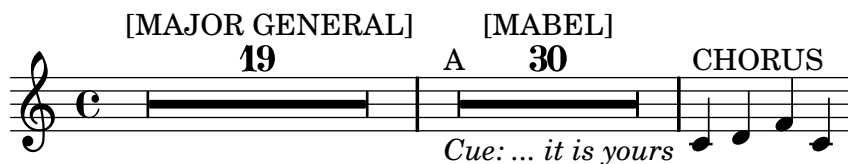
Markups attached to a multi-measure rest will be centered above or below it. Long markups attached to multi-measure rests do not cause the measure to expand. To expand a multi-measure rest to fit the markup, use a spacer rest with an attached markup before the multi-measure rest.

Note that the spacer rest causes a bar line to be inserted. Text attached to a spacer rest in this way is left-aligned to the position where the note would be placed in the measure, but if the measure length is determined by the length of the text, the text will appear to be centered.

```

\relative c' {
  \compressFullBarRests
  \textLengthOn
  s1*0^\markup { [MAJOR GENERAL] }
  R1*19
  s1*0_\markup { \italic { Cue: ... it is yours } }
  s1*0^\markup { A }
  R1*30^\markup { [MABEL] }
  \textLengthOff
  c4^\markup { CHORUS } d f c
}

```



See also

Music Glossary: [Section “multi-measure rest”](#) in *Music Glossary*.

Notation Reference: [\[Durations\]](#), page 35, [Section 1.8 \[Text\]](#), page 175, [Section 1.8.2 \[Formatting text\]](#), page 182, [\[Text scripts\]](#), page 176.

Snippets: [Section “Rhythms”](#) in *Snippets*.

Internals Reference: [Section “MultiMeasureRest”](#) in *Internals Reference*, [Section “MultiMeasureRestNumber”](#) in *Internals Reference*, [Section “MultiMeasureRestText”](#) in *Internals Reference*.

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 ordinary rests into a single multi-measure rest.

Multi-measure rests do not take part in rest collisions.

1.2.3 Displaying rhythms

Time signature

The time signature is set as follows:

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



Time signatures are printed at the beginning of a piece and whenever the time signature changes. If a change takes place at the end of a line a warning time signature sign is printed there. This default behavior may be changed, see [Section 5.4.7 \[Visibility of objects\]](#), page 426.

```
\time 2/4
c2 c
\break
c c
\break
\time 4/4
c c c c
```



The time signature symbol that is used in 2/2 and 4/4 time can be changed to a numeric style:

```
% Default style
\time 4/4 c1
\time 2/2 c1
% Change to numeric style
\numericTimeSignature
\time 4/4 c1
\time 2/2 c1
% Revert to default style
\defaultTimeSignature
\time 4/4 c1
\time 2/2 c1
```



Mensural time signatures are covered in [\[Mensural time signatures\]](#), page 303.

Predefined commands

`\numericTimeSignature`, `\defaultTimeSignature`.

Selected Snippets

Changing the time signature without affecting the beaming

The `\time` command sets the properties `timeSignatureFraction`, `beatLength`, `beatGrouping` and `measureLength` in the `Timing` context, which is normally aliased to `Score`. Changing the value of `timeSignatureFraction` causes the new time signature symbol to be printed without changing any of the other properties:

```
\markup {
  This snippet is deprecated as of 2.13.5 and will be removed in 2.14
}
```

This snippet is deprecated as of 2.13.5 and will be removed in 2.14

Compound time signatures

Odd 20th century time signatures (such as "5/8") can often be played as compound time signatures (e.g. "3/8 + 2/8"), which combine two or more unequal metrics. LilyPond can make such music quite easy to read and play, by explicitly printing the compound time signatures and adapting the automatic beaming behavior. (Graphic measure grouping indications can also be added; see the appropriate snippet in this database.)

```
#(define ((compound-time one two num) grob)
  (grob-interpret-markup grob
    (markup #:override '(baseline-skip . 0) #:number
      (#:line ((#:column (one num))
        #:vcenter "+"
        (#:column (two num)))))))

\relative c' {
  \override Staff.TimeSignature #'stencil = #(compound-time "2" "3" "8")
  \time 5/8
```

```

\overrideBeamSettings #'Staff #'(5 . 8) #'end
  #'((* . (2 3)))
c8 d e fis gis
c8 fis, gis e d
c8 d e4 gis8
}

```



Time signature printing only the numerator as a number (instead of the fraction)

Sometimes, a time signature should not print the whole fraction (e.g. 7/4), but only the numerator (7 in this case). This can be easily done by using `\override Staff.TimeSignature #'style = #'single-digit` to change the style permanently. By using `\revert Staff.TimeSignature #'style`, this setting can be reversed. To apply the single-digit style to only one time signature, use the `\override` command and prefix it with a `\once`.

```

\relative c' {
  \time 3/4
  c4 c c
  % Change the style permanently
  \override Staff.TimeSignature #'style = #'single-digit
  \time 2/4
  c4 c
  \time 3/4
  c4 c c
  % Revert to default style:
  \revert Staff.TimeSignature #'style
  \time 2/4
  c4 c
  % single-digit style only for the next time signature
  \once \override Staff.TimeSignature #'style = #'single-digit
  \time 5/4
  c4 c c c c
  \time 2/4
  c4 c
}

```



See also

Music Glossary: [Section “time signature” in *Music Glossary*](#)

Notation Reference: [\[Mensural time signatures\]](#), page 303, [\[Time administration\]](#), page 89.

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “TimeSignature” in *Internals Reference*](#), [Section “Timing_translator” in *Internals Reference*](#).

Upbeats

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

```
\partial duration
```

where `duration` is the rhythmic length of the remaining interval of the current measure before the start of the next.

```
\partial 4 e4 |
a2. c,4 |
```



The partial measure can be any duration less than the full measure:

```
\partial 8*3 c8 d e |
a2. c,4 |
```



Internally, `\partial duration` is translated into:

```
\set Timing.measurePosition -duration
```

For example, `\partial 8*3` becomes:

```
\set Timing.measurePosition = #(ly:make-moment -3 8)
```

The property `measurePosition` contains a rational number indicating how much of the measure has passed at this point. Note that this is set to a negative number by the `\partial` command: i.e., `\partial 4` is internally translated to `-4`, meaning “there is a quarter note left in the measure.”

See also

Music Glossary: [Section “anacrusis” in *Music Glossary*](#).

Notation Reference: [\[Grace notes\]](#), page 84.

Snippets: [Section “Rhythms” in *Snippets*](#).

Internal Reference: [Section “Timing_translator” in *Internals Reference*](#).

Known issues and warnings

The `\partial` command is intended to be used only at the beginning of a piece. If you use it after the beginning, some odd warnings or effects may occur, in this case use `\set Timing.measurePosition` instead.

Unmetered music

Bar lines and bar numbers are calculated automatically. For unmetered music (some cadenzas, for example), this is not desirable. To turn off automatic calculation of bar lines and bar numbers, use the command `\cadenzaOn`, and use `\cadenzaOff` to turn them on again.

```
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:

```
% Show all bar numbers
\override Score.BarNumber #'break-visibility = #all-visible
c4 d e d
\cadenzaOn
c4 c d8 d d f4 g4.
\cadenzaOff
\bar "|"
d4 e d c
```



Note that these predefined commands affect all staves in the score, even when they are placed in just one **Voice** context. To change this, move the **Timing_translator** from the **Score** context to the **Staff** context, as shown in [\[Polymetric notation\]](#), page 57.

Predefined commands

`\cadenzaOn`, `\cadenzaOff`.

See also

Music Glossary: [Section “cadenza” in *Music Glossary*](#).

Notation Reference: [Section 5.4.7 \[Visibility of objects\]](#), page 426, [\[Polymetric notation\]](#), page 57.

Snippets: [Section “Rhythms” in *Snippets*](#).

Known issues and warnings

LilyPond will insert line breaks and page breaks only at a bar line. Unless the unmeasured 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.

You must explicitly create a **Voice** context when starting a piece with `\cadenzaOn`, otherwise strange errors may occur.

```
\new Voice \relative c'{
  \cadenzaOn
  c16~"Solo Free Time" d e f g2. \bar "||"
  \cadenzaOff
}
```



Polymetric notation

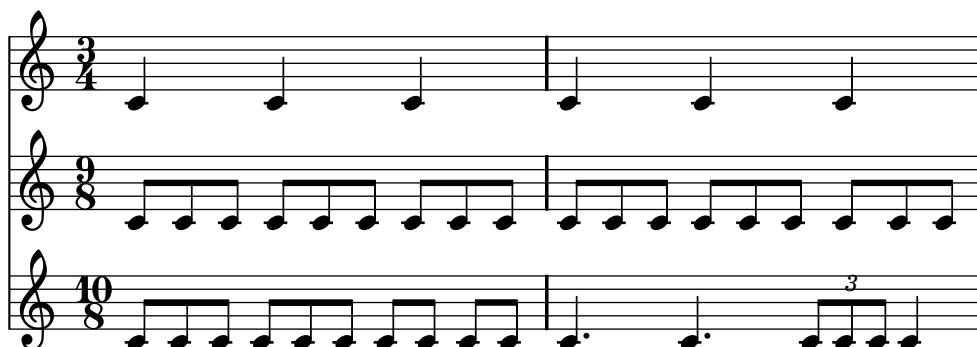
Polymetric notation is supported, either explicitly or by modifying the visible time signature symbol and scaling the note durations.

Staves with different time signatures, equal measure lengths

This notation can be created by setting a common time signature for each staff but replacing the symbol manually by setting `timeSignatureFraction` to the desired fraction and scaling the printed durations in each staff to the common time signature; see [Time signature], page 52. The scaling is done with `\scaleDurations`, which is used in a similar way to `\times`, but does not create a tuplet bracket; see [Scaling durations], page 41.

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, as $2/3 * 9/8 = 3/4$, and in the third staff, shown durations are multiplied by 3/5, as $3/5 * 10/8 = 3/4$. It will often be necessary to insert beams manually, as the duration scaling affects the autobeaming rules.

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



Staves with different time signatures, unequal bar lengths

Each staff can be given its own independent time signature by moving the `Timing_translator` and the `Default_bar_line_engraver` to the `Staff` context.

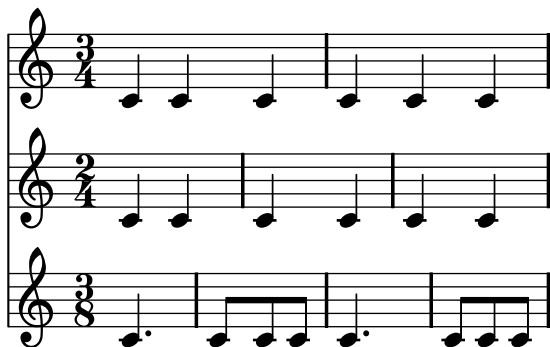
```

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

```



Selected Snippets

Compound time signatures

Odd 20th century time signatures (such as "5/8") can often be played as compound time signatures (e.g. "3/8 + 2/8"), which combine two or more unequal metrics. LilyPond can make

such music quite easy to read and play, by explicitly printing the compound time signatures and adapting the automatic beaming behavior. (Graphic measure grouping indications can also be added; see the appropriate snippet in this database.)

```
#(define ((compound-time one two num) grob)
  (grob-interpret-markup grob
    (markup #:override '(baseline-skip . 0) #:number
      (:line ((#:column (one num))
        #:vcenter "+"
        (:column (two num)))))))

\relative c' {
  \override Staff.TimeSignature #'stencil = #(compound-time "2" "3" "8")
  \time 5/8
  \override BeamSettings #'Staff #'(5 . 8) #'end
  #'((* . (2 3)))
  c8 d e fis gis
  c8 fis, gis e d
  c8 d e4 gis8
}
```



See also

Music Glossary: [Section “polymetric”](#) in *Music Glossary*, [Section “polymetric time signature”](#) in *Music Glossary*, [Section “meter”](#) in *Music Glossary*.

Notation Reference: [\[Time signature\]](#), page 52, [\[Scaling durations\]](#), page 41.

Snippets: [Section “Rhythms”](#) in *Snippets*.

Internals Reference: [Section “TimeSignature”](#) in *Internals Reference*, [Section “Timing-translator”](#) in *Internals Reference*, [Section “Default-bar-line-engraver”](#) in *Internals Reference*, [Section “Staff”](#) in *Internals Reference*.

Known issues and warnings

When using different time signatures in parallel, notes at the same moment will be placed at the same horizontal location. However, the bar lines in the different staves will cause the note spacing to be less regular in each of the individual staves than would be normal without the different time signatures.

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` with the `Completion_heads_engraver`. In the following example, notes crossing the bar lines 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 show exactly how much each measure is off.

See also

Music Glossary: [Section “tie” in *Music Glossary*](#)

Learning Manual: [Section “Engravers explained” in *Learning Manual*](#), [Section “Adding and removing engravers” in *Learning Manual*](#).

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “Note_heads_engraver” in *Internals Reference*](#), [Section “Completion_heads_engraver” in *Internals Reference*](#), [Section “Forbid_line_break_engraver” in *Internals Reference*](#).

Known issues and warnings

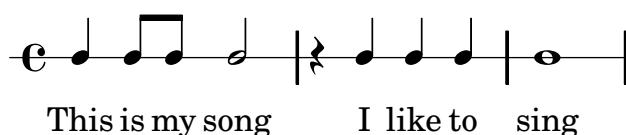
Not all durations (especially those containing tuplets) can be represented exactly with normal notes and dots, but the `Completion_heads_engraver` will not insert tuplets.

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

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

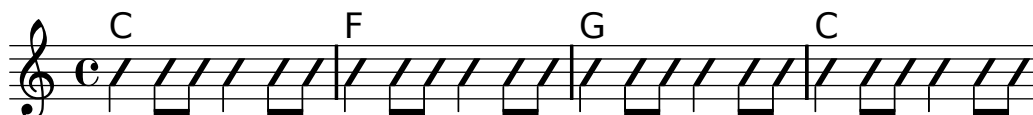
```
<<
\new RhythmicStaff {
  \new Voice = "myRhythm" {
    \time 4/4
    c4 e8 f g2
    r4 g g f
    g1
  }
}
\new Lyrics {
  \lyricsto "myRhythm" {
    This is my song
    I like to sing
  }
}
>>
```



Guitar chord charts often show the strumming rhythms. This can be done with the `Pitch_squash_engraver` and `\improvisationOn`.

```
<<
\new ChordNames {
  \chordmode {
    c1 f g c
  }
}

\new Voice \with {
  \consists Pitch_squash_engraver
} \relative c'' {
  \improvisationOn
  c4 c8 c c4 c8 c
  f4 f8 f f4 f8 f
  g4 g8 g g4 g8 g
  c4 c8 c c4 c8 c
}
>>
```



Predefined commands

`\improvisationOn`, `\improvisationOff`.

Selected Snippets

Guitar strum rhythms

For guitar music, it is possible to show strum rhythms, along with melody notes, chord names and fret diagrams.

```
\include "predefined-guitar-fretboards.ly"
```

```
<<
\new ChordNames {
  \chordmode {
    c1 | f | g | c
  }
}

\new FretBoards {
  \chordmode {
    c1 | f | g | c
  }
}

\new Voice \with {
  \consists "Pitch_squash_engraver"
} {
  \relative c'' {
    \improvisationOn
    c4 c8 c c4 c8 c
    f4 f8 f f4 f8 f
    g4 g8 g g4 g8 g
    c4 c8 c c4 c8 c
  }
}
```

```

    }
  }
  \new Voice = "melody" {
    \relative c'' {
      c2 e4 e4
      f2. r4
      g2. a4
      e4 c2.
    }
  }
  \new Lyrics {
    \lyricsto "melody" {
      This is my song.
      I like to sing.
    }
  }
}
>>

```

Chord diagrams for C, F, and G are shown above the staff. The melody is written in treble clef with a common time signature. The lyrics are: This is my song. I like

Chord diagram for C is shown above the staff. The melody is written in treble clef with a common time signature. The lyrics are: to sing.

See also

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “RhythmicStaff” in *Internals Reference*](#), [Section “Pitch-squash-engraver” in *Internals Reference*](#).

1.2.4 Beams

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 [\[Manual beams\]](#), page 69. Beams *must* be entered manually if beams are to be extended over rests.

If automatic beaming is not required, it may be turned off with `\autoBeamOff` and on with `\autoBeamOn`:

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



Note: If beams are used to indicate melismata in songs, then automatic beaming should be switched off with `\autoBeamOff` and the beams indicated manually.

Beaming patterns that differ from the automatic defaults can be created; see [\[Setting automatic beam behavior\]](#), page 64.

Predefined commands

`\autoBeamOff`, `\autoBeamOn`.

Selected Snippets

Beams across line breaks

Line breaks are normally forbidden when beams cross bar lines. This behavior can be changed as shown:

```
\relative c'' {
  \override Beam #'breakable = ##t
  c8 c[ c] c[ c] c[ c] c[ \break
  c8] c[ c] c[ c] c[ c] c
}
```





Changing beam knee gap

Kneaded 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 kneaded 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: [Manual beams], page 69, [Setting automatic beam behavior], page 64.

Installed Files: ‘`scm/auto-beam.scm`’.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “Auto_beam_engraver” in *Internals Reference*, Section “Beam_engraver” in *Internals Reference*, Section “Beam” in *Internals Reference*, Section “BeamEvent” in *Internals Reference*, Section “BeamForbidEvent” in *Internals Reference*, Section “beam-interface” in *Internals Reference*, Section “unbreakable-spanner-interface” in *Internals Reference*.

Known issues and warnings

Beams can collide with note heads and accidentals in other voices.

Setting automatic beam behavior

The placement of automatic beams is determined by the time signature. Three types of rules are used for determining the end of automatic beams: *default* rules for the time signature, *explicit* rules for the beam in the time signature, and the *beatLength* for the time signature.

The following rules, in order of priority, apply when determining the appearance of beams:

- If a manual beam is specified with `[...]` set the beam as specified, otherwise
- if `\autoBeamOff` is in force do not beam, otherwise
- if an explicit beam-ending rule is defined for the beam type in the time signature, use it to determine the valid places where beams may end, otherwise
- if a default beam-ending rule is defined in the time signature, use it to group notes with beams, otherwise
- use the value of `beatLength` to group notes with beams

Modifying the grouping of beats

By default `beatLength` is derived from the time signature set by the `\time` command. The `beatLength` is set to be one over the denominator of the time signature.

`beatLength` is a *moment*, a unit of musical duration. A quantity of type *moment* is created by the scheme function `ly:make-moment`. For more information about this function, see [Time administration], page 89.

Autobeam settings are stored in the `beamSettings` property. Default values of `beamSettings` are determined in '`scm/beam-settings.scm`'. Entries in `beamSettings` are indexed by time signature and rule type.

Time signature should be a scheme pair, e.g. `#'(4 . 4)`.

Rule type should be `#'end` for beam ending.

The ending rules consist of a scheme alist (or list of pairs) that indicates the beam type and the grouping to be applied to beams containing notes with a shortest duration of that beam type.

```
#'((beam-type1 . grouping-1)
    (beam-type2 . grouping-2)
    (beam-type3 . grouping-3))
```

Beam type is either a scheme pair indicating the duration of the beam, e.g. `(1 . 16)`, or `*` to indicate a default rule, which applies to all beams if no explicit rule is defined.

Grouping is a scheme list indicating the grouping to be applied to the beam. For default rules (where beam type is `*`), the grouping is in units of `beatLength`. For explicit rules, the grouping is in units of the beam type.

For reference, the default beaming rules are found in '`scm/beam-settings.scm`'.

Beam settings are changed with `\overrideBeamSettings` and `\revertBeamSettings`.

Note: The ending settings must be *complete* rules. That is, every rule that should apply to the current time signature must be included in the setting. It is not possible to change the grouping of only one beam type for a given time signature. While this may seem cumbersome, it means that the current beaming settings need not be known in order to specify a new beaming pattern.

```
\time 5/16
c8^"beats" c16 c8 |
\overrideBeamSettings #'Score #'(5 . 16) #'end #'(( * . (2 3)))
c8^"(2+3)" c16 c8
\overrideBeamSettings #'Score #'(5 . 16) #'end #'(( * . (3 2)))
c8^"(3+2)" c16 c8
```



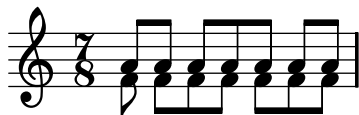
Beam setting changes can be limited to specific contexts. If no setting is included in a lower-level context, the setting of the enclosing context will apply.

```
\new Staff <<
  \time 7/8
  \new Voice = one {
    \relative c'' {
      \overrideBeamSettings #'Staff #'(7 . 8) #'end #'(( * . (2 3 2)))
      a8 a a a a a a
    }
  }
\new Voice = two {
```

```

\relative c' {
  \voiceTwo
  \overrideBeamSettings #'Voice #'(7 . 8) #'end #'((* . (1 3 3)))
  f8 f f f f f f
}
}
>>

```



When multiple voices are used the `Staff` context must be specified if the beaming is to be applied to all voices in the staff:

```

\time 7/8
% rhythm 3-1-1-2
% Context Voice specified -- does not work correctly
% Because of autogenerated voices, all beaming will
% be at beatLength (1 . 8)
\overrideBeamSettings #'Voice #'(7 . 8) #'end #'((* . (3 1 1 2)))
<< {a8 a a a16 a a a a8 a} \ {f4. f8 f f f} >>

% Works correctly with context Staff specified
\overrideBeamSettings #'Staff #'(7 . 8) #'end #'((* . (3 1 1 2)))
<< {a8 a a a16 a a a a8 a} \ {f4. f8 f f f} >>

```



Beam settings can be reverted to get back to default behavior. This is accomplished by using `\revertBeamSettings`. The arguments are the same as for `\overrideBeamSettings`, except no value for *grouping* is given:

```

\revertBeamSettings context time-signature rule-type
\time 4/4
\repeat unfold 8 {a8}
% set default rule for (1 1 1 1) grouping
\overrideBeamSettings #'Score #'(4 . 4) #'end #'((* . (1 1 1 1)))
\repeat unfold 8 {a8}
% revert the new rule
\revertBeamSettings #'Score #'(4 . 4) #'end
\repeat unfold 8 {a8}

```



Predefined commands

`\overrideBeamSettings`, `\revertBeamSettings`.

Selected Snippets

Sub-dividing beams

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 behavior 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` defaults to one over the denominator of the current time signature if not set explicitly. It must be set to a fraction giving the duration of the beam sub-group using the `ly:make-moment` function, as shown here:

```
\relative c'' {
  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 beatLength = #(ly:make-moment 1 8)
  c32[ c c c c c c c]

  % Set beam sub-group length to a sixteenth note
  \set beatLength = #(ly:make-moment 1 16)
  c32[ c c c c c c c]
}
```



Conducting signs measure grouping signs

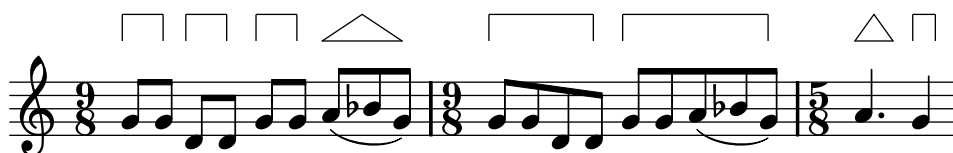
Beat grouping within a bar is controlled by the default grouping established in `beamSettings`. This grouping can be established by the use of `\overrideBeamSettings`. Alternatively, the Scheme function `set-time-signature` can be used to both set the time signature and establish the default grouping rule. `set-time-signature`, 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 in one of the display contexts, measure grouping signs will be created. Such signs ease reading rhythmically complex modern music. In the example, the 9/8 measure is grouped in two different patterns using the two different methods, while the 5/8 measure is grouped according to the default setting in ‘`scm/beam-settings.scm`’:

```
\score {
  \relative c'' {
    \time 9/8
    \overrideBeamSettings #'Score #'(9 . 8) #'end #'((* . (2 2 2 3)))
    g8 g d d g g a( bes g) |
    #(set-time-signature 9 8 '(4 5))
    g8 g d d g g a( bes g) |
    \time 5/8
    a4. g4 |
  }
  \layout {
    \context {
```

```

\Staff
\consists "Measure_grouping_engraver"
}
}
}

```



Beam endings in Score context

Beam-ending rules specified in the **Score** context apply to all staves, but can be modified at both **Staff** and **Voice** levels:

```

\relative c'' {
  \time 5/4
  % Set default beaming for all staves
  \overrideBeamSettings #'Score #'(5 . 4) #'end
    #'(((1 . 8) . (3 4 3))
      ((1 . 16) . (6 8 6))
      ((1 . 32) . (12 16 12)))
  <<
    \new Staff {
      c8 c c c c c c c c c
    }
    \new Staff {
      % Modify beaming for just this staff
      \overrideBeamSettings #'Staff #'(5 . 4) #'end
        #'(( * . (3 2)))
        c8 c c c c c c c c c c
    }
    \new Staff {
      % Inherit beaming from Score context
      <<
        {
          \voiceOne
            c8 c c c c c c c c c
          }
      % Modify beaming for this voice only
      \new Voice {
        \voiceTwo
        \overrideBeamSettings #'Voice #'(5 . 4) #'end
          #'(( * . (3 2)))
          a8 a a a a a a a a a
        }
      >>
    }
  >>
}
>>
}

```



See also

Installed files: ‘scm/beam-settings.scm’.

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “Auto_beam_engraver” in *Internals Reference*](#), [Section “Beam” in *Internals Reference*](#), [Section “BeamForbidEvent” in *Internals Reference*](#), [Section “beam-interface” in *Internals Reference*](#).

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. The workaround for these problems is to manually beam the last beam in the voice or score.

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
```



Beaming direction can be set manually using direction indicators:

```
c8^[ d e] c,_[ d e f g]
```



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

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



Grace note beams and normal note beams can occur simultaneously. Unbeamed grace notes are not put into normal note beams.

```
c4 d8[
\grace { e32[ d c d] }
e8] e8[ e
\grace { f16 }
e8 e]
```



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 eighth-note beam of the group as a whole.

```
a8[ r16 f g a]
a8[ r16
\set stemLeftBeamCount = #2
\set stemRightBeamCount = #1
f16
\set stemLeftBeamCount = #1
g16 a]
```



Predefined commands

`\noBeam.`

Selected Snippets

Flat flags and beam nibs

Flat flags on lone notes and beam nibs at the ends of beamed figures are both possible with a combination of `stemLeftBeamCount`, `stemRightBeamCount` and paired `[]` beam indicators.

For right-pointing flat flags on lone notes, use paired `[]` beam indicators and set `stemLeftBeamCount` to zero (see Example 1).

For left-pointing flat flags, set `stemRightBeamCount` instead (Example 2).

For right-pointing nibs at the end of a run of beamed notes, set `stemRightBeamCount` to a positive value. And for left-pointing nibs at the start of a run of beamed notes, set `stemLeftBeamCount` instead (Example 3).

Sometimes it may make sense for a lone note surrounded by rests to carry both a left- and right-pointing flat flag. Do this with paired `[]` beam indicators alone (Example 4).

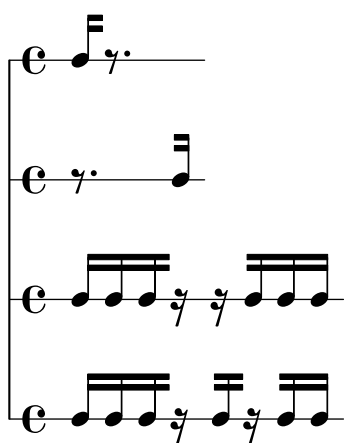
(Note that `\set stemLeftBeamCount` is always equivalent to `\once \set`. In other words, the beam count settings are not “sticky”, so the pair of flat flags attached to the lone `c'16[]` in the last example have nothing to do with the `\set` two notes prior.)

```
\score {
  <<
  % Example 1
  \new RhythmicStaff {
```

```

\set stemLeftBeamCount = #0
c16[]
r8.
}
% Example 2
\new RhythmicStaff {
  r8.
  \set stemRightBeamCount = #0
  c16[]
}
% Example 3
\new RhythmicStaff {
  c16 c
  \set stemRightBeamCount = #2
  c16 r r
  \set stemLeftBeamCount = #2
  c16 c c
}
% Example 4
\new RhythmicStaff {
  c16 c
  \set stemRightBeamCount = #2
  c16 r
  c16[]
  r16
  \set stemLeftBeamCount = #2
  c16 c
}
>>
}

```



See also

Notation Reference: [Section 5.4.2 \[Direction and placement\]](#), page 419, [\[Grace notes\]](#), page 84.

Snippets: [Section “Rhythms”](#) in *Snippets*.

Internals Reference: [Section “Beam”](#) in *Internals Reference*, [Section “BeamEvent”](#) in *Internals Reference*, [Section “Beam-engraver”](#) in *Internals Reference*, [Section “beam-interface”](#) in *Internals Reference*, [Section “Stem-engraver”](#) in *Internals Reference*.

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 `featherDurations` 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.

Predefined commands

`\featherDurations`.

See also

Snippets: [Section “Rhythms” in *Snippets*](#).

Known issues and warnings

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

1.2.5 Bars

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 "|."
```



It is not invalid if the final note in a measure does not end on the automatically entered bar line: the note is assumed to carry over into the next measure. But if a long sequence of such carry-over measures 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 measures, i.e., where all notes end before the end of a measure.

Note: An incorrect duration can cause line breaks to be inhibited, leading to a line of highly compressed music or music which flows off the page.

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

```
\bar ""
```

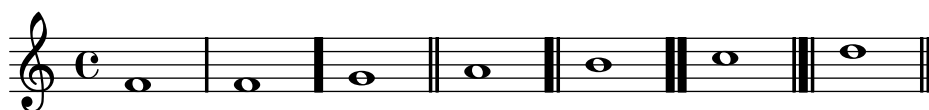
This will insert an invisible bar line and allow (but not force) a line break to occur at this point. The bar number counter is not increased. To force a line break see [Section 4.3.1 \[Line breaking\]](#), page 366.

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

Note that manual bar lines are purely visual. They do not affect any of the properties that a normal bar line would affect, such as measure numbers, accidentals, line breaks, etc. They do not affect the calculation and placement of subsequent automatic bar lines. When a manual bar line is placed where a normal bar line already exists, the effects of the original bar line are not altered.

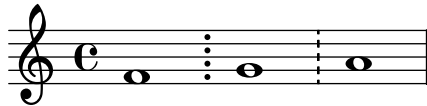
Two types of simple bar lines and five types of double bar lines are available for manual insertion:

```
f1 \bar "|"
f1 \bar " ."
g1 \bar "||"
a1 \bar " .|"
b1 \bar " .| ."
c1 \bar "||| ."
d1 \bar "||| ."
e1
```



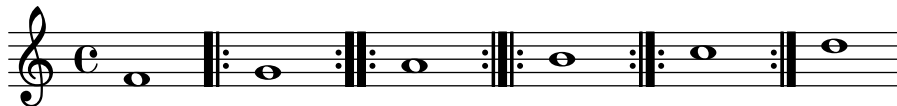
together with dotted and dashed bar lines:

```
f1 \bar ":"
g1 \bar "dashed"
a1
```



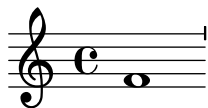
and five types of repeat bar line:

```
f1 \bar "|:"
g1 \bar ":|:"
a1 \bar ":|.|:"
b1 \bar ":|.|:"
c1 \bar ":|"
d1
```



Additionally, a bar line can be printed as a simple tick:

```
f1 \bar "'"
```



However, as such ticks are typically used in Gregorian chant, it is preferable to use `\divisioMinima` there instead, described in the section [\[Divisiones\]](#), [page 310](#) in Gregorian chant.

For in-line segno signs, there are three types of bar lines which differ in their behavior at line breaks:

```
c4 c c c
\bar "S"
c4 c c c \break
\bar "S"
c4 c c c
\bar "|S"
c4 c c c \break
\bar "|S"
c4 c c c
\bar "S|"
c4 c c c \break
\bar "S|"
c1
```





Although the bar line types signifying repeats may be inserted manually they do not in themselves cause LilyPond to recognize a repeated section. Such repeated sections are better entered using the various repeat commands (see [Section 1.4 \[Repeats\]](#), page 110), 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.

```
c4 c c c
\bar "||:"
c4 c c c \break
\bar "||:"
c4 c c c
```



For combinations of repeats with the segno sign, there are six different variations:

```
c4 c c c
\bar " :|S"
c4 c c c \break
\bar " :|S"
c4 c c c
\bar " :|S."
c4 c c c \break
\bar " :|S."
c4 c c c
\bar "S|:"
c4 c c c \break
\bar "S|:"
c4 c c c
\bar ".S|:"
c4 c c c \break
\bar ".S|:"
c4 c c c
\bar " :|S|:"
```

```

c4 c c c \break
\bar ":|S|:"
c4 c c c
\bar ":|S.|:"
c4 c c c \break
\bar ":|S.|:"
c1

```

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 {
      e4 d
      \bar "||"
      f4 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 4.3.1 \[Line breaking\]](#), page 366, [Section 1.4 \[Repeats\]](#), page 110, [\[Grouping staves\]](#), page 137.

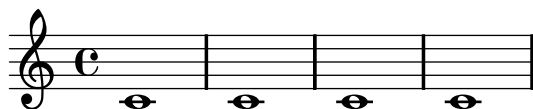
Snippets: [Section “Rhythms” in Snippets](#).

Internals Reference: [Section “BarLine” in Internals Reference](#) (created at `Staff` level), [Section “SpanBar” in Internals Reference](#) (across staves), [Section “Timing_translator” in Internals Reference](#) (for Timing properties).

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:

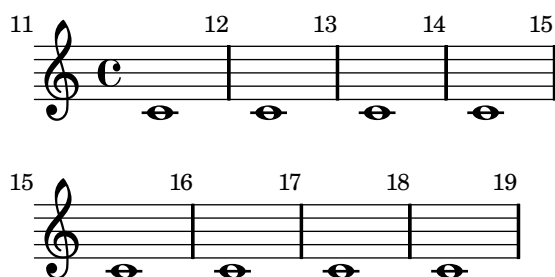
```
c1 c c c
\break
\set Score.currentBarNumber = #50
c1 c c c
```



Bar numbers can be typeset at regular intervals instead of just at the beginning of every line. To do this the default behavior 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
% Permit first bar number to be printed
\bar ""
c1 | c | c | c
\break
c1 | c | c | c
```

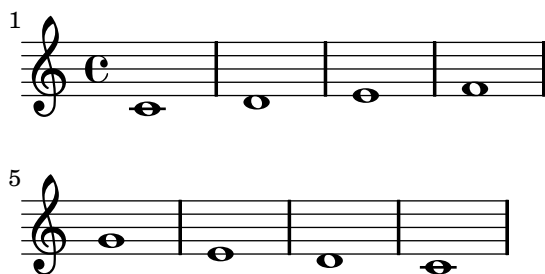


Selected Snippets

Printing the bar number for the first measure

By default, the first bar number in a score is suppressed if it is less than or equal to '1'. By setting `barNumberVisibility` to `all-bar-numbers-visible`, any bar number can be printed for the first measure and all subsequent measures. Note that an empty bar line must be inserted before the first note for this to work.

```
\relative c' {
  \set Score.barNumberVisibility = #all-bar-numbers-visible
  \bar ""
  c1 | d | e | f \break
  g1 | e | d | c
}
```

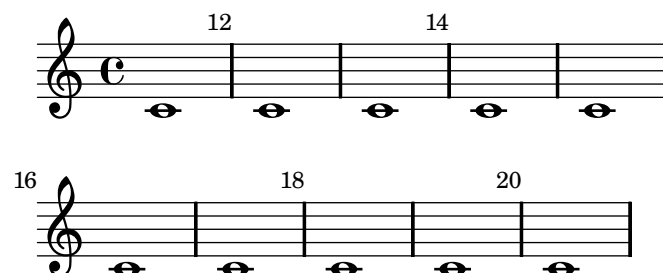


Printing bar numbers at regular intervals

Bar numbers can be printed at regular intervals by setting the property `barNumberVisibility`. Here the bar numbers are printed every two measures except at the end of the line.

```
\relative c' {
  \override Score.BarNumber #'break-visibility = #end-of-line-invisible
  \set Score.currentBarNumber = #11
  % Permit first bar number to be printed
  \bar ""
```

```
% Print a bar number every second measure
\set Score.barNumberVisibility = #(every-nth-bar-number-visible 2)
c1 | c | c | c | c
\break
c1 | c | c | c | c
}
```



Printing bar numbers inside boxes or circles

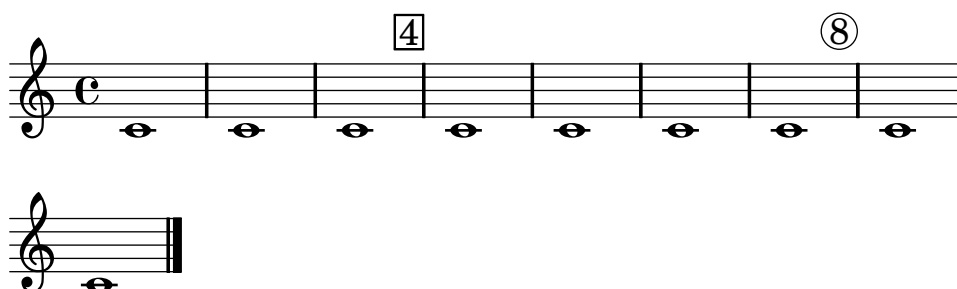
Bar numbers can also be printed inside boxes or circles.

```
\relative c' {
% Prevent bar numbers at the end of a line and permit them elsewhere
\override Score.BarNumber #'break-visibility = #end-of-line-invisible
\set Score.barNumberVisibility = #(every-nth-bar-number-visible 4)

% Increase the size of the bar number by 2
\override Score.BarNumber #'font-size = #2

% 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 5 { c1 }

% 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 "|."
}
```



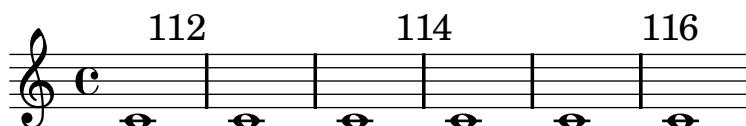
Aligning bar numbers

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

```

\relative c' {
  \set Score.currentBarNumber = #111
  \override Score.BarNumber #'break-visibility = #all-visible
  % Increase the size of the bar number by 2
  \override Score.BarNumber #'font-size = #2
  % Print a bar number every second measure
  \set Score.barNumberVisibility = #(every-nth-bar-number-visible 2)
  c1 | c1
  % Center-align bar numbers
  \override Score.BarNumber #'self-alignment-X = #CENTER
  c1 | c1
  % Left-align bar numbers
  \override Score.BarNumber #'self-alignment-X = #LEFT
  c1 | c1
}

```



Removing bar numbers from a score

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: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “BarNumber” in *Internals Reference*](#), [Section “Bar_number_engraver” in *Internals Reference*](#).

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. See [Section “StaffGroup” in *Internals Reference*](#) and [Section “BarNumber” in *Internals Reference*](#) for more.

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:

```
\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.

If successive bar checks are off by the same musical interval, only the first warning message is displayed. This allows the warning to focus on the source of the timing error.

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.

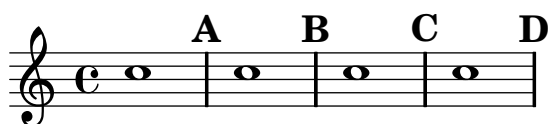
See also

Snippets: [Section “Rhythms” in *Snippets*](#).

Rehearsal marks

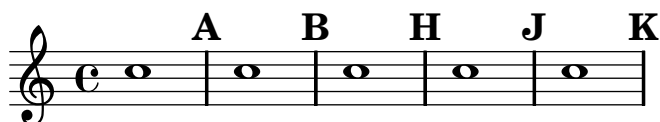
To print a rehearsal mark, use the `\mark` command.

```
c1 \mark \default
c1 \mark \default
c1 \mark \default
c1 \mark \default
```



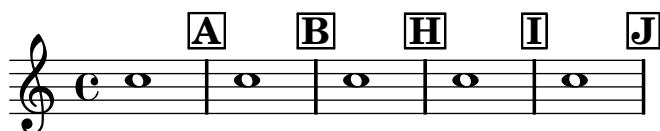
The mark is incremented automatically if you use `\mark \default`, but you can also use an integer argument to set the mark manually. The value to use is stored in the property `rehearsalMark`.

```
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 one of the following commands, depending on which style of rehearsal mark you want (letters only, letters in a hollow box, or letters in a hollow circle).

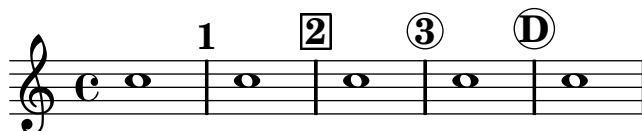
```
\set Score.markFormatter = #format-mark-alphabet
\set Score.markFormatter = #format-mark-box-alphabet
\set Score.markFormatter = #format-mark-circle-alphabet
\set Score.markFormatter = #format-mark-box-alphabet
c1 \mark \default
c1 \mark \default
c1 \mark #8
c1 \mark \default
c1 \mark \default
```



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

```
\set Score.markFormatter = #format-mark-circle-letters
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:

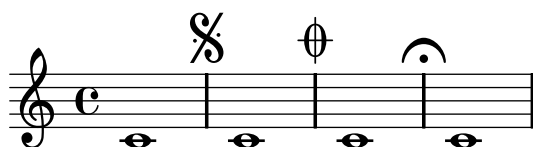
```
\mark "A1"
```

Note that `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 A.6 \[The Feta font\]](#), page 457, for a list of symbols which may be printed with `\musicglyph`.

For common tweaks to the positioning of rehearsal marks, see [Section 1.8.2 \[Formatting text\]](#), page 182. For more precise control, see `break-alignable-interface` in [Section 5.5.1 \[Aligning objects\]](#), page 433.

See also

Notation Reference: [Section A.6 \[The Feta font\]](#), page 457, [Section 1.8.2 \[Formatting text\]](#), page 182, [Section 5.5.1 \[Aligning objects\]](#), page 433.

Installed Files: ‘`scm/translation-functions.scm`’ contains the definitions of `format-mark-numbers` and `format-mark-letters`. They can be used as inspiration for other formatting functions.

Snippets: [Section “Rhythms”](#) in *Snippets*.

Internals Reference: [Section “MarkEvent”](#) in *Internals Reference*, [Section “Mark-engraver”](#) in *Internals Reference*, [Section “RehearsalMark”](#) in *Internals Reference*.

1.2.6 Special rhythmic concerns

Grace notes

Grace notes are ornaments that are written out. Grace notes are printed in a smaller font and take up no logical time in a measure.

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



LilyPond also supports two special types of grace notes, the *acciaccatura* – an unmeasured grace note indicated by a slurred small note with a slashed stem – and the *appoggiatura*, which takes a fixed fraction of the main note and appears in small print without a slash.

```
\grace c8 b4
\acciaccatura d8 c4
\appoggiatura e8 d4
\acciaccatura { g16[ f] } e4
```



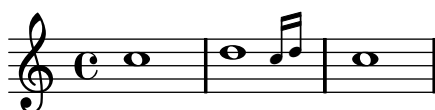
The placement of grace notes is synchronized between different staves. In the following example, there are two sixteenth grace notes for every eighth grace note

```
<<
  \new Staff { e2 \grace { c16[ d e f] } e2 }
  \new Staff { c2 \grace { g8[ b] } c2 }
>>
```



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] } c1
```



This will put the grace notes after a space lasting $3/4$ of the length of the main note. The default fraction $3/4$ can be changed by setting `afterGraceFraction`. The following example shows the results from setting the space at the default, at $15/16$, and finally at $1/2$ of the main note.

```
<<
\new Staff {
  c1 \afterGrace d1 { c16[ d] } c1
}
\new Staff {
  #(define afterGraceFraction (cons 15 16))
  c1 \afterGrace d1 { c16[ d] } c1
}
\new Staff {
  #(define afterGraceFraction (cons 1 2))
  c1 \afterGrace d1 { c16[ d] } c1
}
>>
```



The space between the main note and the grace note may also be specified using spacers. The following example places the grace note after a space lasting 7/8 of the main note.

```
\new Voice {
  <<
    { d1^\trill_( }
    { s2 s4. \grace { c16[ d] } }
  >>
  c1)
}
```



A `\grace` music expression will introduce special typesetting settings, for example, to produce smaller type, and set directions. Hence, when introducing layout tweaks to override the special settings, they should be placed inside the grace expression. The overrides should also be reverted inside the grace expression. Here, the grace note's default stem direction is overridden and then reverted.

```
\new Voice {
  \acciaccatura {
    \stemDown
    f16->
    \stemNeutral
  }
  g4 e c2
}
```



Selected Snippets

Using grace note slashes with normal heads

The slash through the stem found in acciaccaturas can be applied in other situations.

```
\relative c'' {
  \override Stem #'stroke-style = #"grace"
  c8( d2) e8( f4)
}
```



Tweaking grace layout within music

The layout of grace expressions can be changed throughout the music using the functions `add-grace-property` and `remove-grace-property`. The following example undefines the `Stem` direction for this grace, so that stems do not always point up, and changes the default note heads to crosses.

```
\relative c'' {
  \new Staff {
    #(remove-grace-property 'Voice 'Stem 'direction)
    #(add-grace-property 'Voice 'NoteHead 'style 'cross)
    \new Voice {
      \acciaccatura { f16 } g4
      \grace { d16[ e] } f4
      \appoggiatura { f,32[ g a] } e2
    }
  }
}
```



Redefining grace note global defaults

The global defaults for grace notes are stored in the identifiers `startGraceMusic`, `stopGraceMusic`, `startAcciaccaturaMusic`, `stopAcciaccaturaMusic`, `startAppoggiaturaMusic` and `stopAppoggiaturaMusic`, which are defined in the file `ly/grace-init.ly`. By redefining them other effects may be obtained.

```
startAcciaccaturaMusic = {
  s1*0(
  \override Stem #'stroke-style = #"grace"
  \slurDashed
}

stopAcciaccaturaMusic = {
  \revert Stem #'stroke-style
  \slurSolid
}
```

```

s1*0)
}

\relative c'' {
  \acciaccatura d8 c1
}

```



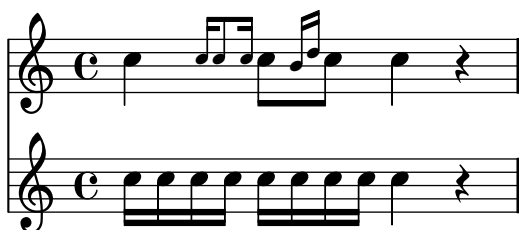
Positioning grace notes with floating space

Setting the property '`strict-grace-spacing`' makes the musical columns for grace notes 'floating', i.e., decoupled from the non-grace notes: first the normal notes are spaced, then the (musical columns of the) graces are put left of the musical columns for the main notes.

```

\relative c'' {
  <<
    \override Score.SpacingSpanner #'strict-grace-spacing = ##t
    \new Staff \new Voice {
      \afterGrace c4 { c16[ c8 c16] }
      c8[ \grace { b16[ d] } c8]
      c4 r
    }
    \new Staff {
      c16 c c c c c c c c4 r
    }
  >>
}

```



See also

Music Glossary: Section “grace notes” in *Music Glossary*, Section “acciaccatura” in *Music Glossary*, Section “appoggiatura” in *Music Glossary*.

Notation Reference: [Manual beams], page 69.

Installed Files: ‘`ly/grace-init.ly`’.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “GraceMusic” in *Internals Reference*, Section “Grace.beam-engraver” in *Internals Reference*, Section “Grace-engraver” in *Internals Reference*, Section “Grace-spacing-engraver” in *Internals Reference*.

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 d2. }
  \new Staff { c4 \bar "|:" d2. }
>>
```



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 d2. }
  \new Staff { c4 \bar "|:" \grace s16 d2. }
>>
```



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.

Aligning to cadenzas

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

One solution to this problem is to use the functions `mmrest-of-length` and `skip-of-length`. These Scheme functions take a defined piece of music as an argument and generate a multi-measure rest or `\skip` exactly as long as the piece.

```
MyCadenza = \relative c' {
  c4 d8 e f g g4
  f2 g4 g
}
```

```
\new GrandStaff <<
  \new Staff {
    \MyCadenza c'1
```

```

\MyCadenza c'1
}
\new Staff {
  #(ly:export (mmrest-of-length MyCadenza))
  c'1
  #(ly:export (skip-of-length MyCadenza))
  c'1
}
>>

```



See also

Music Glossary: [Section “cadenza” in *Music Glossary*](#).

Snippets: [Section “Rhythms” in *Snippets*](#).

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. To ensure that the `Timing` alias is available, you may need to explicitly instantiate the containing context (such as `Voice` or `Staff`).

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 [\[Bar numbers\]](#), page 77.

`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.

```

\new Voice \relative c' {
  \set Timing.measureLength = #(ly:make-moment 5 4)
  c1 c4 |
  c1 c4 |

```

```

c4 c
\set Timing.measurePosition = #(ly:make-moment 5 8)
b4 b 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

Notation Reference: [\[Bar numbers\]](#), page 77, [\[Unmetered music\]](#), page 55.

Snippets: [Section “Rhythms” in *Snippets*](#).

Internals Reference: [Section “Timing-translator” in *Internals Reference*](#), [Section “Score” in *Internals Reference*](#).

1.3 Expressive marks

RONDO
Allegro

This section lists various expressive marks that can be created in a score.

1.3.1 Attached to notes

This section explains how to create expressive marks that are attached to notes: articulations, ornamentations, and dynamics. Methods to create new dynamic markings are also discussed.

Articulations and ornamentations

A variety of symbols that denote articulations, ornamentations, and other performance indications can be attached to a note using this syntax:

`note\name`

The possible values for *name* are listed in [Section A.10 \[List of articulations\]](#), page 513. For example:

```
c4\staccato c\mordent b2\turn
c1\fermata
```



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. Predefined shorthands exist for *marcato*, *stopped*, *tenuto*, *staccatissimo*, *accent*, *staccato*, and *portato*. Their corresponding output appears as follows:

```
c4-^ c-+ c-- c-|
c4-> c-. c2-_
```



The rules for the default placement of articulations are defined in ‘`scm/script.scm`’. Articulations and ornamentations may be manually placed above or below the staff; see [Section 5.4.2 \[Direction and placement\]](#), page 419.

Articulations are `Script` objects. Their properties are described more fully in [Section “Script” in *Internals Reference*](#).

In addition to articulations, text and markups can be attached to notes. See [\[Text scripts\]](#), page 176.

For more information about the ordering of Scripts and TextScripts that are attached to the notes, see [Section “Placement of objects” in *Learning Manual*](#).

Selected Snippets

Modifying default values for articulation shorthand notation

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' { c1-+ }
```

```
dashPlus = "trill"
```

```
\relative c' { c1-+ }
```



Controlling the vertical ordering of scripts

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 determines which one comes first.

```
\relative c''' {
  \once \override TextScript #'script-priority = #-100
  a2^\prall^\markup { \sharp }

  \once \override Script #'script-priority = #-100
  a2^\prall^\markup { \sharp }
}
```



Creating a delayed turn

Creating a delayed turn, where the lower note of the turn uses the accidental, requires several overrides. The `outside-staff-priority` property must be set to `#f`, as otherwise this would take precedence over the `avoid-slur` property. The value of `halign` is used to position the turn horizontally.

```
\relative c'' {
  \once \override TextScript #'avoid-slur = #'inside
  \once \override TextScript #'outside-staff-priority = ##f
  c2(^\markup \tiny \override #'(baseline-skip . 1) {
    \halign #-4
    \center-column {
      \sharp
      \musicglyph #"scripts.turn"
    }
  })
  d4.) c8
}
```



See also

Music Glossary: [Section “tenuto”](#) in *Music Glossary*, [Section “accent”](#) in *Music Glossary*, [Section “staccato”](#) in *Music Glossary*, [Section “portato”](#) in *Music Glossary*.

Learning Manual: [Section “Placement of objects”](#) in *Learning Manual*.

Notation Reference: [\[Text scripts\]](#), page 176, [Section 5.4.2 \[Direction and placement\]](#), page 419, [Section A.10 \[List of articulations\]](#), page 513, [\[Trills\]](#), page 108.

Installed Files: '`scm/script.scm`'.

Snippets: Section “Expressive marks” in *Snippets*.

Internals Reference: Section “Script” in *Internals Reference*, Section “TextScript” in *Internals Reference*.

Dynamics

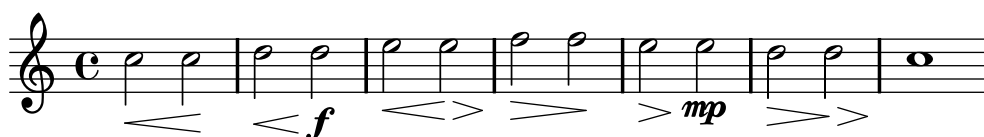
Absolute dynamic marks are specified using a command after a note, such as `c4\ff`. The available dynamic marks are `\ppppp`, `\pppp`, `\ppp`, `\pp`, `\p`, `\mp`, `\mf`, `\f`, `\ff`, `\fff`, `\ffff`, `\fffff`, `\fp`, `\sf`, `\sff`, `\sp`, `\spp`, `\sfz`, and `\rfz`. Dynamic marks may be manually placed above or below the staff; see Section 5.4.2 [Direction and placement], page 419.

```
c2\ppp c\mp
c2\rfz c^\mf
c2_\spp c^\ff
```



A *crescendo* mark is started with `\<` and terminated with `\!`, an absolute dynamic, or an additional crescendo or decrescendo mark. A *decrescendo* mark is started with `\>` and is also terminated with `\!`, an absolute dynamic, or another crescendo or decrescendo mark. `\cr` and `\decr` may be used instead of `\<` and `\>`. *Hairpins* are engraved by default using this notation.

```
c2\< c\!
d2\< d\f
e2\< e\>
f2\> f\!
e2\> e\mp
d2\> d\>
c1\!
```



Spacer rests are needed to engrave multiple marks on one note.

```
c4\< c\! d\> e\!
<< f1 { s4 s4\< s4\> s4\! } >>
```



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

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



Crescendos and decrescendos can be engraved as textual markings instead of hairpins. Dashed lines are printed to indicate their extent. The built-in commands `\crescTextCresc`, `\dimTextDecresc`, `\dimTextDecr`, and `\dimTextDim` will tell LilyPond to use such textual marks instead of hairpins for all subsequent `\<` and `\>` commands. The corresponding `\crescHairpin` and `\dimHairpin` commands will revert to hairpins again:

```
\crescTextCresc
c2\< d | e f\!
\dimTextDecresc
e2\> d | c b\!
\crescHairpin
c2\< d | e f\!
\dimHairpin
e2\> d\!
```



To create new absolute dynamic marks or text that should be aligned with dynamics, see [\[New dynamic marks\]](#), page 97.

Vertical positioning of dynamics is handled by [Section “DynamicLineSpanner”](#) in *Internals Reference*.

Predefined commands

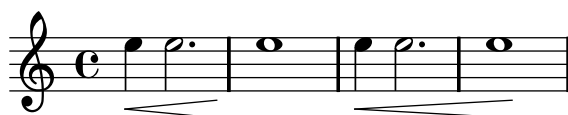
```
\dynamicUp, \dynamicDown, \dynamicNeutral, \crescTextCresc, \dimTextDim,
\dimTextDecr, \dimTextDecresc, \crescHairpin, \dimHairpin.
```

Selected Snippets

Setting hairpin behavior at bar lines

If the note which ends a hairpin falls on a downbeat, the hairpin stops at the bar line immediately preceding. This behavior can be controlled by overriding the `'to-barline` property.

```
\relative c'' {
  e4\< e2.
  e1\!
  \override Hairpin #'to-barline = ##f
  e4\< e2.
  e1\!
}
```



Setting the minimum length of hairpins

If hairpins are too short, they can be lengthened by modifying the `minimum-length` property of the `Hairpin` object.

```
\relative c'' {
  c4\< c\! d\> e\!
  \override Hairpin #'minimum-length = #5
  << f1 { s4 s\< s\> s\! } >>
```

}



Printing hairpins using *al niente* notation

Hairpin dynamics may be printed with a circled tip (“*al niente*” notation) by setting the `circled-tip` property of the `Hairpin` object to `#t`.

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



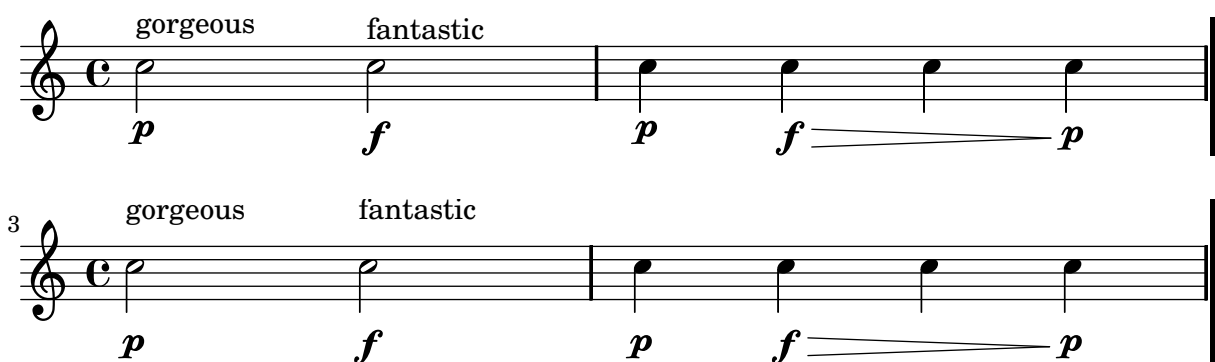
Vertically aligned dynamics and textscripts

By setting the `'Y-extent` property to a suitable value, all `DynamicLineSpanner` objects (hairpins and dynamic texts) can be aligned to a common reference point, regardless of their actual extent. This way, every element will be vertically aligned, thus producing a more pleasing output.

The same idea is used to align the text scripts along their baseline.

```
music = \relative c'' {
  c2\p^\markup { gorgeous } c\f^\markup { fantastic }
  c4\p c\f\> c c\!\p
}

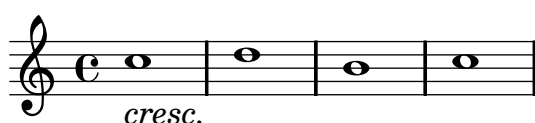
{
  \music \break
  \override DynamicLineSpanner #'staff-padding = #2.0
  \override DynamicLineSpanner #'Y-extent = #'(-1.5 . 1.5)
  \override TextScript #'Y-extent = #'(-1.5 . 1.5)
  \music
}
```



Hiding the extender line for text dynamics

Text style dynamic changes (such as cresc. and dim.) are printed with a dashed line showing their extent. This line can be suppressed in the following way:

```
\relative c'' {
  \override DynamicTextSpanner #'dash-period = #-1.0
  \crescTextCresc
  c1\< | d | b | c\!
}
```



Changing text and spanner styles for text dynamics

The text used for crescendos and decrescendos can be changed by modifying the context properties `crescendoText` and `decrescendoText`. The style of the spanner line can be changed by modifying the `'style` property of `DynamicTextSpanner`. The default value is `'hairpin`, and other possible values include `'line`, `'dashed-line` and `'dotted-line`.

```
\relative c'' {
  \set crescendoText = \markup { \italic { cresc. poco } }
  \set crescendoSpanner = #'text
  \override DynamicTextSpanner #'style = #'dotted-line
  a2\< a
  a2 a
  a2 a
  a2 a\mf
}
```



See also

Music Glossary: Section “al niente” in *Music Glossary*, Section “crescendo” in *Music Glossary*, Section “decrescendo” in *Music Glossary*, Section “hairpin” in *Music Glossary*.

Learning Manual: Section “Articulation and dynamics” in *Learning Manual*.

Notation Reference: Section 5.4.2 [Direction and placement], page 419, [New dynamic marks], page 97, Section 3.5.3 [What goes into the MIDI output?], page 350, Section 3.5.5 [Controlling MIDI dynamics], page 352.

Snippets: Section “Expressive marks” in *Snippets*.

Internals Reference: Section “DynamicText” in *Internals Reference*, Section “Hairpin” in *Internals Reference*, Section “DynamicLineSpanner” in *Internals Reference*.

New dynamic marks

The easiest way to create dynamic indications is to use `\markup` objects.

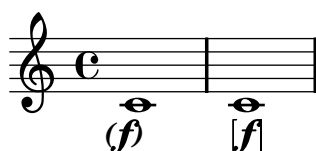
```
moltoF = \markup { molto \dynamic f }
```

```
\relative c' {
  <d e>16_\moltoF <d e>
  <d e>2..
}
```



In markup mode, editorial dynamics (within parentheses or square brackets) can be created. The syntax for markup mode is described in [Section 1.8.2 \[Formatting text\]](#), page 182.

```
roundF = \markup {
  \center-align \concat { \bold { \italic ( }
    \dynamic f \bold { \italic ) } } }
boxF = \markup { \bracket { \dynamic f } }
\relative c' {
  c1_\roundF
  c1_\boxF
}
```



Simple, centered dynamic marks are easily created with the `make-dynamic-script` function.

```
sfzp = #(make-dynamic-script "sfzp")
\relative c' {
  c4 c c\sfzp c
}
```



In general, `make-dynamic-script` takes any markup object as its argument. The dynamic font only contains the characters `f`, `m`, `p`, `r`, `s` and `z`, so if a dynamic mark that includes plain text or punctuation symbols is desired, markup commands that reverts font family and font encoding to normal text should be used, for example `\normal-text`. The interest of using `make-dynamic-script` instead of an ordinary markup is ensuring the vertical alignment of markup objects and hairpins that are attached to the same note head.

```
roundF = \markup { \center-align \concat {
  \normal-text { \bold { \italic ( } }
  \dynamic f
  \normal-text { \bold { \italic ) } } } }
```

```

boxF = \markup { \bracket { \dynamic f } }
mfEspress = \markup { \center-align \line {
    \hspace #3.7 mf \normal-text \italic espress. } }
roundFdynamic = #(make-dynamic-script roundF)
boxFdynamic = #(make-dynamic-script boxF)
mfEspressDynamic = #(make-dynamic-script mfEspress)
\relative c' {
  c4_\roundFdynamic\< d e f
  g,1~_\boxFdynamic\>
  g1
  g'1~\mfEspressDynamic
  g1
}

```



The Scheme form of markup mode may be used instead. Its syntax is explained in [Section “Markup construction in Scheme” in *Extending*](#).

```

moltoF = #(make-dynamic-script
  (markup #:normal-text "molto"
    #:dynamic "f"))
\relative c' {
  <d e>16 <d e>
  <d e>2..\moltoF
}

```



Font settings in markup mode are described in [\[Selecting font and font size\]](#), page 184.

See also

Notation Reference: [Section 1.8.2 \[Formatting text\]](#), page 182, [\[Selecting font and font size\]](#), page 184, [Section 3.5.3 \[What goes into the MIDI output?\]](#), page 350, [Section 3.5.5 \[Controlling MIDI dynamics\]](#), page 352.

Snippets: [Section “Expressive marks” in *Snippets*](#).

Extend: [Section “Markup construction in Scheme” in *Extending*](#).

1.3.2 Curves

This section explains how to create various expressive marks that are curved: normal slurs, phrasing slurs, breath marks, falls, and doits.

Slurs

Slurs are entered using parentheses:

Note: In polyphonic music, a slur must be terminated in the same voice it began.

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



Slurs may be manually placed above or below the staff; see [Section 5.4.2 \[Direction and placement\]](#), page 419.

Simultaneous or overlapping slurs are not permitted, but a phrasing slur can overlap a slur. This permits two slurs to be printed at once. For details, see [\[Phrasing slurs\]](#), page 101.

Slurs can be solid, dotted, or dashed. Solid is the default slur style:

```
c4( e g2)
\slurDashed
g4( e c2)
\slurDotted
c4( e g2)
\slurSolid
g4( e c2)
```



Slurs can also be made half-dashed (the first half dashed, the second half solid) or half-solid (the first half solid, the second half dashed):

```
c4( e g2)
\slurHalfDashed
g4( e c2)
\slurHalfSolid
c4( e g2)
\slurSolid
g4( e c2)
```



Custom dash patterns for slurs can be defined:

```
c4( e g2)
\slurDashPattern #0.7 #0.75
g4( e c2)
\slurDashPattern #0.5 #2.0
```

```
c4( e g2)
\slurSolid
g4( e c2)
```



Predefined commands

`\slurUp`, `\slurDown`, `\slurNeutral`, `\slurDashed`, `\slurDotted`, `\slurHalfDashed`, `\slurHalfSolid`, `\slurDashPattern`, `\slurSolid`.

Selected Snippets

Using double slurs for legato chords

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

```
\relative c' {
  \set doubleSlurs = ##t
  <c e>4( <d f> <c e> <d f>)
}
```



Positioning text markups inside slurs

Text markups need to have the `outside-staff-priority` property set to false in order to be printed inside slurs.

```
\relative c' {
  \override TextScript #'avoid-slur = #'inside
  \override TextScript #'outside-staff-priority = ##f
  c2(^{\markup { \halign #-10 \natural } d4.}) c8
}
```



Making slurs with complex dash structure

Slurs can be made with complex dash patterns by defining the `dash-definition` property. `dash-definition` is a list of `dash-elements`. A `dash-element` is a list of parameters defining the dash behavior for a segment of the slur.

The slur is defined in terms of the bezier parameter `t` which ranges from 0 at the left end of the slur to 1 at the right end of the slur. `dash-element` is a list (`start-t stop-t dash-fraction dash-period`). The region of the slur from `start-t` to `stop-t` will have a fraction `dash-fraction` of each `dash-period` black. `dash-period` is defined in terms of staff spaces. `dash-fraction` is set to 1 for a solid slur.

```
\relative c' {
  \once \override
    Slur #'dash-definition = #'((0 0.3 0.1 0.75)
                                (0.3 0.6 1 1)
                                (0.65 1.0 0.4 0.75))

  c4( d e f)
  \once \override
    Slur #'dash-definition = #'((0 0.25 1 1)
                                (0.3 0.7 0.4 0.75)
                                (0.75 1.0 1 1))

  c4( d e f)
}
```



See also

Music Glossary: [Section “slur” in *Music Glossary*](#).

Learning Manual: [Section “On the un-nestedness of brackets and ties” in *Learning Manual*](#).

Notation Reference: [Section 5.4.2 \[Direction and placement\]](#), page 419, [Phrasing slurs], page 101.

Snippets: [Section “Expressive marks” in *Snippets*](#).

Internals Reference: [Section “Slur” in *Internals Reference*](#).

Phrasing slurs

Phrasing slurs (or phrasing marks) that indicate a musical sentence are written using the commands `\(` and `\)` respectively:

```
c4\( d( e) f(
e2) d\)
```



Typographically, a 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. Phrasing may be manually placed above or below the staff; see [Section 5.4.2 \[Direction and placement\]](#), page 419.

Simultaneous or overlapping phrasing slurs are not permitted.

Phrasing slurs can be solid, dotted, or dashed. Solid is the default style for phrasing slurs:

```
c4\( e g2\
\phrasingSlurDashed
g4\( e c2\
\phrasingSlurDotted
c4\( e g2\
\phrasingSlurSolid
g4\( e c2\)
```



Phrasing slurs can also be made half-dashed (the first half dashed, the second half solid) or half-solid (the first half solid, the second half dashed):

```
c4\ ( e g2\ )
\phrasingSlurHalfDashed
g4\ ( e c2\ )
\phrasingSlurHalfSolid
c4\ ( e g2\ )
\phrasingSlurSolid
g4\ ( e c2\ )
```



Custom dash patterns for phrasing slurs can be defined:

```
c4\ ( e g2\ )
\phrasingSlurDashPattern #0.7 #0.75
g4\ ( e c2\ )
\phrasingSlurDashPattern #0.5 #2.0
c4\ ( e g2\ )
\phrasingSlurSolid
g4\ ( e c2\ )
```



Dash pattern definitions for phrasing slurs have the same structure as dash pattern definitions for slurs. For more information about complex dash patterns, see the snippets under [\[Slurs\]](#), [page 99](#).

Predefined commands

```
\phrasingSlurUp, \phrasingSlurDown, \phrasingSlurNeutral, \phrasingSlurDashed,
\phrasingSlurDotted, \phrasingSlurHalfDashed, \phrasingSlurHalfSolid,
\phrasingSlurDashPattern, \phrasingSlurSolid.
```

See also

Learning Manual: [Section “On the un-nestedness of brackets and ties”](#) in *Learning Manual*.

Notation Reference: [Section 5.4.2 \[Direction and placement\]](#), [page 419](#), [\[Slurs\]](#), [page 99](#).

Snippets: [Section “Expressive marks”](#) in *Snippets*.

Internals Reference: [Section “PhrasingSlur”](#) in *Internals Reference*.

Breath marks

Breath marks are entered using `\breathe`:

```
c2. \breathe d4
```



A breath mark will end an automatic beam; to override this behavior, see [\[Manual beams\]](#), page 69.

```
c8 \breathe d e f g2
```



Musical indicators for breath marks in ancient notation, *divisiones*, are supported. For details, see [\[Divisiones\]](#), page 310.

Selected Snippets

Changing the breath mark symbol

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

```
\relative c'' {
  c2
  \override BreathingSign #'text = \markup { \musicglyph #"scripts.rvarcomma" }
  \breathe
  d2
}
```



Inserting a caesura

Caesura marks can be created by overriding the `text` property of the `BreathingSign` object. A curved caesura mark is also available.

```
\relative c'' {
  \override BreathingSign #'text = \markup {
    \musicglyph #"scripts.caesura.straight"
  }
  c8 e4. \breathe g8. e16 c4

  \override BreathingSign #'text = \markup {
    \musicglyph #"scripts.caesura.curved"
  }
  g8 e'4. \breathe g8. e16 c4
}
```



See also

Music Glossary: [Section “caesura”](#) in *Music Glossary*.

Notation Reference: [\[Divisiones\]](#), page 310.

Snippets: [Section “Expressive marks”](#) in *Snippets*.

Internals Reference: [Section “BreathingEvent”](#) in *Internals Reference*, [Section “BreathingSign”](#) in *Internals Reference*, [Section “Breathing-sign-engraver”](#) in *Internals Reference*.

Falls and doits

Falls and *doits* can be added to notes using the `\bendAfter` command. The direction of the fall or doit is indicated with a plus or minus (up or down). The number indicates the pitch interval that the fall or doit will extend *beyond* the main note.

```
c2-\bendAfter #+4
c2-\bendAfter #-4
c2-\bendAfter #+6.5
c2-\bendAfter #-6.5
c2-\bendAfter #+8
c2-\bendAfter #-8
```



The dash - immediately preceding the `\bendAfter` command is *required* when writing falls and doits.

Selected Snippets

Adjusting the shape of falls and doits

The `shortest-duration-space` property may have to be tweaked to adjust the shape of falls and doits.

```
\relative c' ' {
  \override Score.SpacingSpanner #'shortest-duration-space = #4.0
  c2-\bendAfter #5
  c2-\bendAfter #-4.75
  c2-\bendAfter #8.5
  c2-\bendAfter #-6
}
```



See also

Music Glossary: [Section “fall” in Music Glossary](#), [Section “doit” in Music Glossary](#).

Snippets: [Section “Expressive marks” in Snippets](#).

1.3.3 Lines

This section explains how to create various expressive marks that follow a linear path: glissandos, arpeggios, and trills.

Glissando

A *glissando* is created by attaching `\glissando` to a note:

```
g2\glissando g'
c2\glissando c,
```



Different styles of glissandi can be created. For details, see [Section 5.4.8 \[Line styles\]](#), [page 431](#).

Selected Snippets

Contemporary glissando

A contemporary glissando without a final note can be typeset using a hidden note and cadenza timing.

```
\relative c'' {
  \time 3/4
  \override Glissando #'style = #'zigzag
  c4 c
  \cadenzaOn
  c4\glissando
  \hideNotes
  c,,4
  \unHideNotes
  \cadenzaOff
  \bar "|"
}
```



See also

Music Glossary: [Section “glissando” in *Music Glossary*](#).

Notation Reference: [Section 5.4.8 \[Line styles\]](#), [page 431](#).

Snippets: [Section “Expressive marks” in *Snippets*](#).

Internals Reference: [Section “Glissando” in *Internals Reference*](#).

Known issues and warnings

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

Arpeggio

An *arpeggio* on a chord (also known as a broken chord) is denoted by appending `\arpeggio` to the chord construct:

```
<c e g c>1\arpeggio
```



Different types of arpeggios may be written. `\arpeggioNormal` reverts to a normal arpeggio:

```
<c e g c>2\arpeggio
```

```
\arpeggioArrowUp
```

```
<c e g c>2\arpeggio
```

```
\arpeggioArrowDown
```

```
<c e g c>2\arpeggio
```

```
\arpeggioNormal
```

```
<c e g c>2\arpeggio
```



Special *bracketed* arpeggio symbols can be created:

```
<c e g c>2
```

```
\arpeggioBracket
```

```
<c e g c>2\arpeggio
```

```
\arpeggioParenthesis
```

```
<c e g c>2\arpeggio
```

```
\arpeggioParenthesisDashed
```

```
<c e g c>2\arpeggio
```

```
\arpeggioNormal
```

```
<c e g c>2\arpeggio
```



The dash properties of the parenthesis arpeggio are controlled with the 'dash-details property, which is described at [\[Slurs\]](#), page 99.

Arpeggios can be explicitly written out with ties. For more information, see [\[Ties\]](#), page 42.

Predefined commands

```
\arpeggio, \arpeggioArrowUp, \arpeggioArrowDown, \arpeggioNormal, \arpeggioBracket, \arpeggioParenthesis \arpeggioParenthesisDashed.
```

Selected Snippets

Creating cross-staff arpeggios in a piano staff

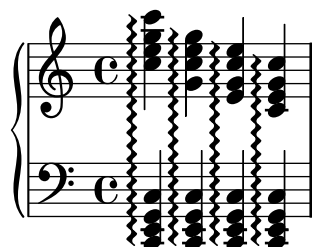
In a PianoStaff, it is possible to let an arpeggio cross between the staves by setting the property PianoStaff.connectArpeggios = ##t.

```
\new PianoStaff \relative c'' <<
  \set PianoStaff.connectArpeggios = ##t
  \new Staff {
    <c e g c>4\arpeggio
    <g c e g>4\arpeggio
```

```

    <e g c e>4\arpeggio
    <c e g c>4\arpeggio
  }
  \new Staff {
    \clef bass
    \repeat unfold 4 {
      <c,, e g c>4\arpeggio
    }
  }
>>

```



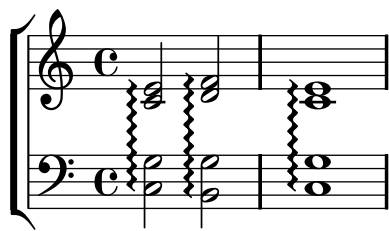
Creating cross-staff arpeggios in other contexts

Cross-staff arpeggios can be created in contexts other than `GrandStaff`, `PianoStaff` and `StaffGroup` if the `Span_arpeggio_engraver` is included in the `Score` context.

```

\score {
  \new ChoirStaff {
    \set Score.connectArpeggios = ##t
    <<
      \new Voice \relative c' {
        <c e>2\arpeggio
        <d f>2\arpeggio
        <c e>1\arpeggio
      }
      \new Voice \relative c {
        \clef bass
        <c g'>2\arpeggio
        <b g'>2\arpeggio
        <c g'>1\arpeggio
      }
    >>
  }
  \layout {
    \context {
      \Score
      \consists "Span_arpeggio_engraver"
    }
  }
}

```



Creating arpeggios across notes in different voices

An arpeggio can be drawn across notes in different voices on the same staff if the `Span_arpeggio_engraver` is added 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

Music Glossary: [Section “arpeggio” in Music Glossary](#).

Notation Reference: [\[Slurs\]](#), page 99, [\[Ties\]](#), page 42.

Snippets: [Section “Expressive marks” in Snippets](#).

Internals Reference: [Section “Arpeggio” in Internals Reference](#), [Section “Slur” in Internals Reference](#), [Section “PianoStaff” in Internals Reference](#).

Known issues and warnings

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

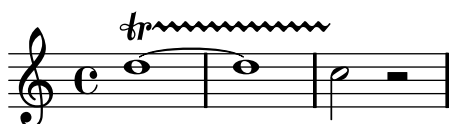
The parenthesis-style arpeggio brackets do not work for cross-staff arpeggios.

Trills

Short *trills* without an extender line are printed with `\trill`; see [\[Articulations and ornaments\]](#), page 90.

Longer trills with an extender line are made with `\startTrillSpan` and `\stopTrillSpan`:

```
d1~\startTrillSpan
d1
c2\stopTrillSpan r2
```



In the following example, a trill is combined with grace notes. The syntax of this construct and the method to precisely position the grace notes are described in [\[Grace notes\]](#), page 84.

```
c1 \afterGrace
d1\startTrillSpan { c32[ d]\stopTrillSpan }
e2 r2
```



Trills that require an auxiliary note with an explicit pitch can be typeset with the `\pitchedTrill` command. The first argument is the main note, and the second is the *trilled* note, printed as a stemless note head in parentheses.

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



In the following example, the second pitched trill is ambiguous; the accidental of the trilled note is not printed. As a workaround, the accidentals of the trilled notes can be forced. The second measure illustrates this method:

```
\pitchedTrill eis4\startTrillSpan fis
g\stopTrillSpan
\pitchedTrill eis4\startTrillSpan fis
g\stopTrillSpan
\pitchedTrill eis4\startTrillSpan fis
g\stopTrillSpan
\pitchedTrill eis4\startTrillSpan fis!
g\stopTrillSpan
```



Predefined commands

`\startTrillSpan`, `\stopTrillSpan`.

See also

Music Glossary: [Section “trill” in Music Glossary](#).

Notation Reference: [\[Articulations and ornamentations\]](#), page 90, [\[Grace notes\]](#), page 84.

Snippets: [Section “Expressive marks” in Snippets](#).

Internals Reference: [Section “TrillSpanner” in Internals Reference](#).

1.4 Repeats



Repetition is a central concept in music, and multiple notations exist for repetitions. LilyPond supports the following kinds of repeats:

- volta** The repeated music is not written out but enclosed between repeat bar lines. If the repeat is at the beginning of a piece, a repeat bar line is only printed at the end of the repeat. 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.
- percent** These are beat or measure repeats. They look like single slashes or percent signs.
- tremolo** This is used to write tremolo beams.

1.4.1 Long repeats

This section discusses how to input long (usually multi-measure) repeats. The repeats can take two forms: repeats enclosed between repeat signs; or written out repeats, used to input repetitious music. Repeat signs can also be controlled manually.

Normal repeats

The syntax for a normal repeat is

```
\repeat volta repeatcount musicexpr
```

where *musicexpr* is a music expression. Alternate endings can be produced using `\alternative`. In order to delimit the alternate endings, the group of alternatives must be enclosed in a set of braces. If there are more repeats than there are alternate endings, the earliest repeats are given the first alternative.

Normal repeats without alternate endings:

```
\repeat volta 2 { c4 d e f }
c2 d
\repeat volta 2 { d4 e f g }
```



Normal repeats with alternate endings:

```
\repeat volta 4 { c4 d e f }
\alternative {
  { d2 e }
  { f2 g }
}
c1
```



Note: If you include `\relative` inside a `\repeat` without explicitly instantiating the `Voice` context, extra (unwanted) staves will appear. See [Section “An extra staff appears” in *Application Usage*](#).

If a repeat starts in the middle of a measure and has no alternate endings, normally the end of the repeat will also fall in the middle of a measure, so that the two ends add up to one complete measure. In such cases, the repeat signs do not constitute true bar lines. Do not use `\partial` commands or bar checks where these repeat signs are printed:

```
% no \partial here
c4 e g % no bar check here
% no \partial here
\repeat volta 4 {
  e4 |
  c2 e |
  % no \partial here
  g4 g g % no bar check here
}
% no \partial here
g4 |
a2 a |
g1 |
```



Similarly, if a repeat begins with the initial partial measure of a score and has no alternate endings, the same conditions apply as in the above example, except that in this case the `\partial` command is required at the beginning of the score:

```
\partial 4 % required
\repeat volta 4 {
  e4 |
  c2 e |
  % no \partial here
  g4 g g % no bar check here
}
% no \partial here
g4 |
a2 a |
```

g1 |



When alternate endings are added to a repeat that begins with an incomplete measure, it becomes necessary to set the `Timing.measureLength` context property manually, in the following specific places:

- at the start of any incomplete measures in the `\alternative` block, which normally occur at the end of each alternative, except (in most cases) the last.
- at the start of each alternative, except the first.

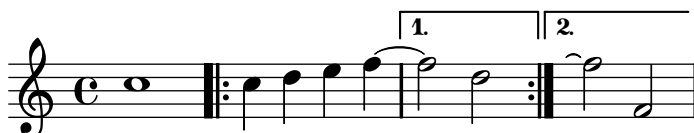
```
\partial 4
\repeat volta 2 { e4 | c2 e | }
\alternative {
  {
    f2 d |
    \set Timing.measureLength = #(ly:make-moment 3 4)
    g4 g g % optional bar check is allowed here
  }
  {
    \set Timing.measureLength = #(ly:make-moment 4 4)
    a2 a |
  }
}
g1 |
```



The `measureLength` property is described in [\[Time administration\]](#), page 89.

Ties may be added to a second ending:

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



Selected Snippets

Shortening volta brackets

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
  c4 c c
  \set Score.voltaSpannerDuration = #(ly:make-moment 3 4)
  \repeat volta 5 { d4 d d }
  \alternative {
    {
      e4 e e
      f4 f f
    }
    { g4 g g }
  }
}

```



Adding volta brackets to additional staves

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 the `Volta_engraver` to the `Staff` context where the brackets should appear; see also the “Volta multi staff” snippet.

```

<<
  \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' } }
>>

```



See also

Music Glossary: [Section “repeat” in Music Glossary](#), [Section “volta” in Music Glossary](#).

Notation Reference: [Bar lines], page 72, Section 5.1.4 [Modifying context plug-ins], page 403, [Time administration], page 89.

Snippets: Section “Repeats” in *Snippets*.

Internals Reference: Section “VoltaBracket” in *Internals Reference*, Section “RepeatedMusic” in *Internals Reference*, Section “VoltaRepeatedMusic” in *Internals Reference*, Section “UnfoldedRepeatedMusic” in *Internals Reference*.

Known issues and warnings

Slurs that span from a `\repeat` block into an `\alternative` block will only work for the first alternate ending. Also, slurs cannot wrap around from the end of one alternative back to the beginning of the repeat.

If a repeat that begins with an incomplete measure has an `\alternative` block that contains modifications to the `measureLength` property, using `\unfoldRepeats` will result in wrongly-placed bar lines and bar check warnings.

A nested repeat like

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

is ambiguous, since it 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.

Manual repeat marks

Note: These methods are only used for displaying unusual repeat constructs, and may produce unexpected behavior. In most cases, repeats should be created using the standard `\repeat` command or by printing the relevant bar lines. For more information, see [Bar lines], page 72.

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.

```
c1
\set Score.repeatCommands = #'(start-repeat)
d4 e f g
c1
```



As per standard engraving practice, repeat signs are not printed at the beginning of a piece.

end-repeat

Print a :| bar line:

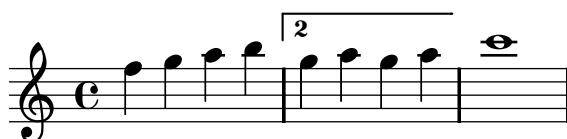
```
c1
d4 e f g
\set Score.repeatCommands = #'(end-repeat)
c1
```



(volta *number*) ... (volta #f)

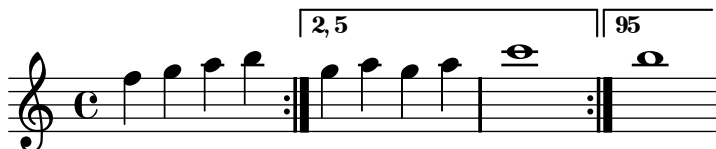
Create a new volta with the specified number. The volta bracket must be explicitly terminated, or it will not be printed.

```
f4 g a b
\set Score.repeatCommands = #'((volta "2"))
g4 a g a
\set Score.repeatCommands = #'((volta #f))
c1
```



Multiple repeat commands may occur at the same point:

```
f4 g a b
\set Score.repeatCommands = #'((volta "2, 5") end-repeat)
g4 a g a
c1
\set Score.repeatCommands = #'((volta #f) (volta "95") end-repeat)
b1
\set Score.repeatCommands = #'((volta #f))
```



Text can be included with the volta bracket. The text can be a number or numbers or markup text, see [Section 1.8.2 \[Formatting text\]](#), page 182. The simplest way to use markup text is to define the markup first, then include the markup in a Scheme list.

```
voltaAdLib = \markup { 1. 2. 3... \text \italic { ad lib. } }
\relative c' {
  c1
  \set Score.repeatCommands = #(list(list 'volta voltaAdLib) 'start-repeat)
  c4 b d e
  \set Score.repeatCommands = #'((volta #f) (volta "4.") end-repeat)
  f1
  \set Score.repeatCommands = #'((volta #f))
}
```

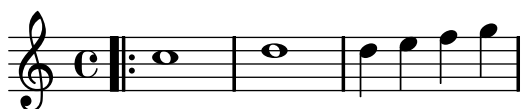


Selected Snippets

Printing a repeat sign at the beginning of a piece

A |: bar line can be printed at the beginning of a piece, by overriding the relevant property:

```
\relative c'' {
  \once \override Score.BreakAlignment #'break-align-orders =
    #(make-vector 3 '(instrument-name
                      left-edge
                      ambitus
                      span-bar
                      breathing-sign
                      clef
                      key-signature
                      time-signature
                      staff-bar
                      custos
                      span-bar))
  \once \override Staff.TimeSignature #'space-alist =
    #'((first-note . (fixed-space . 2.0))
       (right-edge . (extra-space . 0.5))
       ;; free up some space between time signature
       ;; and repeat bar line
       (staff-bar . (extra-space . 1)))
  \bar "||:"
  c1
  d1
  d4 e f g
}
```



See also

Notation Reference: [Bar lines], page 72, Section 1.8.2 [Formatting text], page 182.

Snippets: Section “Repeats” in *Snippets*.

Internals Reference: Section “VoltaBracket” in *Internals Reference*, Section “RepeatedMusic” in *Internals Reference*, Section “VoltaRepeatedMusic” in *Internals Reference*.

Written-out repeats

By using the `unfold` command, repeats can be used to simplify the writing out of repetitious music. The syntax is

```
\repeat unfold repeatcount musicexpr
```

where *musicexpr* is a music expression and *repeatcount* is the number of times *musicexpr* is repeated.

```
c1
\repeat unfold 2 { c4 d e f }
c1
```



Unfold repeats can be made with alternate endings. If there are more repeats than there are alternate endings, the first alternative ending is applied to the earliest endings.

```
c1
\repeat unfold 2 { g4 f e d }
  \alternative {
    { cis2 g' }
    { cis,2 b }
  }
c1
```



Note: If you include `\relative` inside a `\repeat` without explicitly instantiating the `Voice` context, extra (unwanted) staves will appear. See [Section “An extra staff appears” in Application Usage](#).

See also

Snippets: [Section “Repeats” in Snippets](#).

Internals Reference: [Section “RepeatedMusic” in Internals Reference](#), [Section “UnfoldedRepeatedMusic” in Internals Reference](#).

1.4.2 Short repeats

This section discusses how to input short repeats. Short repeats can take two basic forms: repeats of a single note to two measures, represented by slashes or percent signs; and tremolos.

Percent repeats

Repeated short patterns of notes are supported. The music is printed once, and the pattern is replaced with a special sign. Patterns that are shorter than one measure are replaced by slashes, and patterns of one or two measures are replaced by percent-like signs. The syntax is

```
\repeat percent number musicexpr
where musicexpr is a music expression.
\repeat percent 4 { c4 }
\repeat percent 2 { b4 a g f }
\repeat percent 2 { c2 es | f4 fis g c | }
```

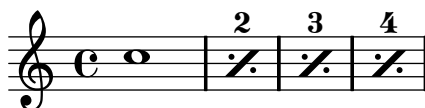


Selected Snippets

Percent repeat counter

Measure repeats of more than two repeats can get a counter when the `convenient` property is switched, as shown in this example:

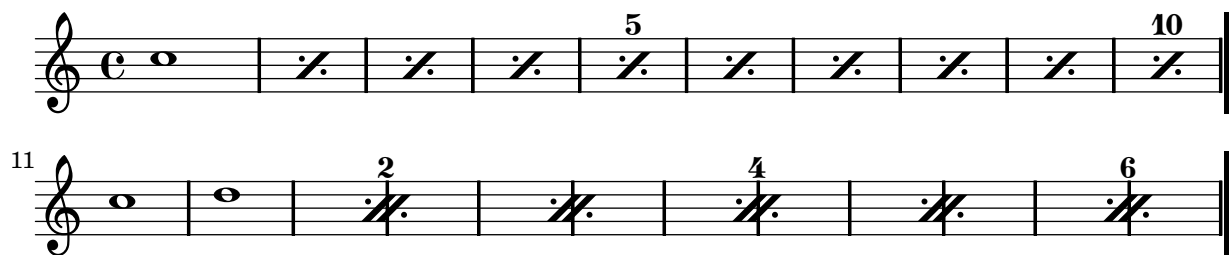
```
\relative c' {
  \set countPercentRepeats = ##t
  \repeat percent 4 { c1 }
}
```



Percent repeat count visibility

Percent repeat counters can be shown at regular intervals by setting the context property `repeatCountVisibility`.

```
\relative c' {
  \set countPercentRepeats = ##t
  \set repeatCountVisibility = #(every-nth-repeat-count-visible 5)
  \repeat percent 10 { c1 } \break
  \set repeatCountVisibility = #(every-nth-repeat-count-visible 2)
  \repeat percent 6 { c1 d1 }
}
```



Isolated percent repeats

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

```
\relative c' {
  \override MultiMeasureRest #'stencil
    = #ly:multi-measure-rest::percent
  \override MultiMeasureRest #'thickness = #0.48
  R1
}
```



See also

Music Glossary: [Section “percent repeat”](#) in *Music Glossary*, [Section “simile”](#) in *Music Glossary*.

Snippets: [Section “Repeats”](#) in *Snippets*.

Internals Reference: [Section “RepeatSlash”](#) in *Internals Reference*, [Section “PercentRepeat”](#) in *Internals Reference*, [Section “DoublePercentRepeat”](#) in *Internals Reference*, [Section “DoublePercentRepeatCounter”](#) in *Internals Reference*, [Section “PercentRepeatCounter”](#) in *Internals Reference*, [Section “PercentRepeatedMusic”](#) in *Internals Reference*.

Known issues and warnings

Only three kinds of percent repeats are supported: a single slash representing a single beat (regardless of the duration of the repeated notes); a single slash with dots representing one full measure; and two slashes with dots crossing a bar line representing two full measures. Neither multiple slashes representing single beat repeats consisting of sixteenth or shorter notes, nor two slashes with dots representing single beat repeats consisting of notes of varying durations, are supported.

Tremolo repeats

Tremolos can take two forms: alternation between two chords or two notes, and rapid repetition of a single note or chord. Tremolos consisting of an alternation are indicated by adding beams between the notes or chords being alternated, while tremolos consisting of the rapid repetition of a single note are indicated by adding beams or slashes to a single note.

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

```
\repeat tremolo 8 { c16 d }
\repeat tremolo 6 { c16 d }
\repeat tremolo 2 { c16 d }
```



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 is also used 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:

```
c2:8 c:32
c: c:
```



Selected Snippets

Cross-staff tremolos

Since `\repeat tremolo` expects exactly two musical arguments for chord tremolos, the note or chord which changes staff within a cross-staff tremolo should be placed inside curly braces together with its `\change Staff` command.

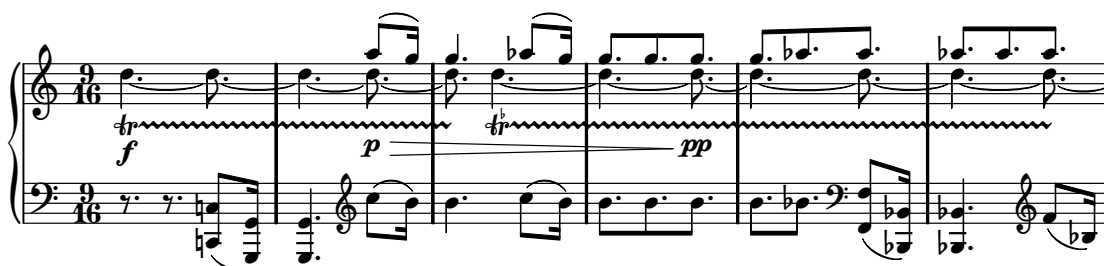
```
\new PianoStaff <<
  \new Staff = "up" \relative c'' {
    \key a \major
    \time 3/8
    s4.
  }
  \new Staff = "down" \relative c'' {
    \key a \major
    \time 3/8
    \voiceOne
    \repeat tremolo 6 {
      <a e'>32
      {
        \change Staff = "up"
        \voiceTwo
        <cis a' dis>32
      }
    }
  }
}
>>
```



See also

Snippets: Section “Repeats” in *Snippets*.

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

This section discusses simultaneous notes inside the same voice.

Chorded notes

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

```
<c e g>2 <c f a>4-> <e g c>-.
```



Relative mode can be used for pitches in chords. The octave of each pitch is chosen using the preceding pitch as a reference except in the case of the first pitch in a chord: the reference for the first pitch is the *first* pitch of the preceding chord.

For more information about chords, see [Section 2.7 \[Chord notation\]](#), page 278.

See also

Music Glossary: [Section “chord” in Music Glossary](#).

Learning Manual: [Section “Combining notes into chords” in Learning Manual](#).

Notation Reference: [Section 2.7 \[Chord notation\]](#), page 278.

Snippets: [Section “Simultaneous notes” in Snippets](#).

Chord repetition

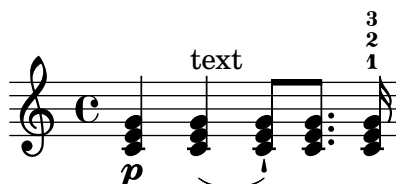
In order to save typing, a shortcut can be used to repeat the preceding chord. The chord repetition symbol is `q`:

```
<c e g>4 q q q
```



As with regular chords, the chord repetition symbol can be used with durations, articulations, markups, slurs, beams, etc. as only the pitches of the previous chord are duplicated.

```
<c e g>4\p q^"text"( q8)[-| q8.] q16-1-2-3
```



The chord repetition symbol always remembers the last instance of a chord so it is possible to repeat the most recent chord even if other non-chorded notes or rests have been added since.

```
<c e g>8 c'4 q r8 q
```



See also

Installed Files: 'ly/chord-repetition-init.ly'.

Simultaneous expressions

One or more music expressions enclosed in double angle brackets are taken to be simultaneous. If the first expression begins with a single note or if the whole simultaneous expression appears explicitly within a single voice, the whole expression is placed on a single staff; otherwise the elements of the simultaneous expression are placed on separate staves.

The following examples show simultaneous expressions on one staff:

```
\new Voice { % explicit single voice
  << { a4 b g2 } { d4 g c,2 } >>
}
```



```
% single first note
a << { a4 b g } { d4 g c, } >>
```



This can be useful if the simultaneous sections have identical rhythms, but attempts to attach notes with different durations to the same stem will cause errors.

The following example shows how simultaneous expressions can generate multiple staves implicitly:

```
% no single first note
<< { a4 b g2 } { d4 g2 c,4 } >>
```

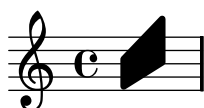


Here different rhythms cause no problems.

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 { <g b>2 <c g'> }
```



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

Music Glossary: [Section “cluster” in *Music Glossary*](#).

Snippets: [Section “Simultaneous notes” in *Snippets*](#).

Internals Reference: [Section “ClusterSpanner” in *Internals Reference*](#), [Section “ClusterSpannerBeacon” in *Internals Reference*](#), [Section “Cluster_spanner_engraver” in *Internals Reference*](#).

Known issues and warnings

Clusters look good only if they span at least two chords; otherwise they appear too narrow.

Clusters do not have a stem and cannot indicate durations by themselves, but the length of the printed cluster is determined by the durations of the defining chords. Separate clusters need a separating rest between them.

Clusters do not produce MIDI output.

1.5.2 Multiple voices

This section discusses simultaneous notes in multiple voices or multiple staves.

Single-staff polyphony

Explicitly instantiating voices

The basic structure needed to achieve multiple independent voices in a single staff is illustrated in the following example:

```
\new Staff <<
  \new Voice = "first"
    { \voiceOne r8 r16 g e8. f16 g8[ c,] f e16 d }
  \new Voice= "second"
    { \voiceTwo d16 c d8~ d16 b c8~ c16 b c8~ c16 b8. }
>>
```



Here, voices are instantiated explicitly and are given names. The `\voiceOne ... \voiceFour` commands set up the voices so that first and third voices get stems up, second and fourth voices get stems down, third and fourth voice note heads are horizontally shifted, and rests in the

respective voices are automatically moved to avoid collisions. The `\oneVoice` command returns all the voice settings to the neutral default directions.

Temporary polyphonic passages

A temporary polyphonic passage can be created with the following construct:

```
<< { \voiceOne ... }
  \new Voice { \voiceTwo ... }
>> \oneVoice
```

Here, the first expression within a temporary polyphonic passage is placed into the `Voice` context which was in use immediately before the polyphonic passage, and that same `Voice` context continues after the temporary section. Other expressions within the angle brackets are assigned to distinct temporary voices. This allows lyrics to be assigned to one continuing voice before, during and after a polyphonic section:

```
<<
  \new Voice = "melody" {
    a4
    <<
      {
        \voiceOne
        g f
      }
      \new Voice {
        \voiceTwo
        d2
      }
    >>
    \oneVoice
    e4
  }
  \new Lyrics \lyricsto "melody" {
    This is my song.
  }
>>
```



Here, the `\voiceOne` and `\voiceTwo` commands are required to define the settings of each voice.

The double backslash construct

The `<< { ... } \ { ... } >>` construct, where the two (or more) expressions are separated by double backslashes, behaves differently to the similar construct without the double backslashes: *all* the expressions within this construct are assigned to new `Voice` contexts. These new `Voice` contexts are created implicitly and are given the fixed names "1", "2", etc.

The first example could be typeset as follows:

```
<<
{ r8 r16 g e8. f16 g8[ c,] f e16 d }
\\
{ d16 c d8~ d16 b c8~ c16 b c8~ c16 b8. }
```

>>



This syntax can be used where it does not matter that temporary voices are created and then discarded. These implicitly created voices are given the settings equivalent to the effect of the `\voiceOne ... \voiceFour` commands, in the order in which they appear in the code.

In the following example, the intermediate voice has stems up, therefore we enter it in the third place, so it becomes voice three, which has the stems up as desired. Spacer rests are used to avoid printing doubled rests.

```
<<
{ r8 g g g g f16 ees f8 d }
\\
{ ees,8 r ees r d r d r }
\\
{ d'8 s c s bes s a s }
>>
```



In all but the simplest works it is advisable to create explicit `Voice` contexts as explained in [Section “Contexts and engravers” in *Learning Manual*](#) and [Section “Explicitly instantiating voices” in *Learning Manual*](#).

Identical rhythms

In the special case that we want to typeset parallel pieces of music that have the same rhythm, we can combine them into a single `Voice` context, thus forming chords. To achieve this, enclose them in a simple simultaneous music construct within an explicit voice:

```
\new Voice <<
{ e4 f8 d e16 f g8 d4 }
{ c4 d8 b c16 d e8 b4 }
>>
```



This method leads to strange beamings and warnings if the pieces of music do not have the same rhythm.

Predefined commands

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

See also

Learning Manual: Section “Voices contain music” in *Learning Manual*, Section “Explicitly instantiating voices” in *Learning Manual*.

Notation Reference: [Percussion staves], page 267, [Invisible rests], page 47, [Stems], page 170.

Snippets: Section “Simultaneous notes” in *Snippets*.

Voice styles

Voices may be given distinct colors and shapes, allowing them to be easily identified:

```
<<
{ \voiceOneStyle d4 c2 b4 }
\\
{ \voiceTwoStyle e,2 e }
\\
{ \voiceThreeStyle b2. c4 }
\\
{ \voiceFourStyle g'2 g }
>>
```



The `\voiceNeutralStyle` command is used to revert to the standard presentation.

Predefined commands

`\voiceOneStyle`, `\voiceTwoStyle`, `\voiceThreeStyle`, `\voiceFourStyle`,
`\voiceNeutralStyle`.

See also

Learning Manual: Section “I’m hearing Voices” in *Learning Manual*, Section “Other sources of information” in *Learning Manual*.

Snippets: Section “Simultaneous notes” in *Snippets*.

Collision resolution

The note heads of notes in different voices with the same pitch, same note head and opposite stem direction are automatically merged, but notes with different note heads or the same stem direction are not. Rests opposite a stem in a different voice are shifted vertically. The following example shows three different circumstances, on beats 1 and 3 in bar 1 and beat 1 in bar 2, where the automatic merging fails.

```
<<
{
  c8 d e d c d c4
  g'2 fis
} \\ {
  c2 c8. b16 c4
  e,2 r
} \\ {
  \oneVoice
  s1
  e8 a b c d2
```

```
}
>>
```



Notes with different note heads may be merged, with the exception of half-note heads and quarter-note heads, as shown below. Here the note heads on beat 1 of bar 1 are now merged:

```
<<
```

```
{
  \mergeDifferentlyHeadedOn
  c8 d e d c d c4
  g'2 fis
} \ {
  c2 c8. b16 c4
  e,2 r
} \ {
  \oneVoice
  s1
  e8 a b c d2
}
>>
```



Note heads with different dots as shown in beat 3 of bar 1 may be also be merged:

```
<<
```

```
{
  \mergeDifferentlyHeadedOn
  \mergeDifferentlyDottedOn
  c8 d e d c d c4
  g'2 fis
} \ {
  c2 c8. b16 c4
  e,2 r
} \ {
  \oneVoice
  s1
  e8 a b c d2
}
>>
```



The half note and eighth note at the start of the second measure are incorrectly merged because the automatic merge cannot successfully complete the merge when three or more notes

line up in the same note column, and in this case the merged note head is incorrect. To allow the merge to select the correct note head a `\shift` must be applied to the note that should not be merged. Here, `\shiftOn` is applied to move the top *g* out of the column, and `\mergeDifferentlyHeadedOn` then works properly.

```
<<
{
  \mergeDifferentlyHeadedOn
  \mergeDifferentlyDottedOn
  c8 d e d c d c4
  \shiftOn
  g'2 fis
} \ {
  c2 c8. b16 c4
  e,2 r
} \ {
  \oneVoice
  s1
  e8 a b c d2
}
>>
```



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

Notes are only merged if they have opposing stem directions (as they have, for example, in voices one and two by default or when the stems are explicitly set in opposite directions).

Predefined commands

`\mergeDifferentlyDottedOn`, `\mergeDifferentlyDottedOff`, `\mergeDifferentlyHeadedOn`, `\mergeDifferentlyHeadedOff`.

`\shiftOn`, `\shiftOnn`, `\shiftOnnn`, `\shiftOff`.

Selected Snippets

Additional voices to avoid collisions

In some instances of complex polyphonic music, additional voices are necessary to prevent collisions between notes. If more than four parallel voices are needed, additional voices can be added by defining a variable using the Scheme function `context-spec-music`.

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

```
\relative c'' {
  \time 3/4
  \key d \minor
  \partial 2
  <<
```

```

{
  \voiceOne
  a4. a8
  e'4 e4. e8
  f4 d4. c8
}
\\
{
  \voiceThree
  f,2
  bes4 a2
  a4 s2
}
\\
{
  \voiceFive
  s2
  g4 g2
  f4 f2
}
\\
\bar "||"{
  \voiceTwo
  d2
  d4 cis2
  d4 bes2
}
>>
}

```



Forcing horizontal shift of notes

When the typesetting engine cannot cope, the following syntax can be used to override typesetting decisions. The units of measure used here are staff spaces.

```

\relative c' <<
{
  <d g>2 <d g>
}
\\
{
  <b f'>2
  \once \override NoteColumn #'force-hshift = #1.7
  <b f'>2
}
>>

```



See also

Music Glossary: [Section “polyphony”](#) in *Music Glossary*.

Learning Manual: [Section “Multiple notes at once”](#) in *Learning Manual*, [Section “Voices contain music”](#) in *Learning Manual*, [Section “Real music example”](#) in *Learning Manual*.

Snippets: [Section “Simultaneous notes”](#) in *Snippets*.

Internals Reference: [Section “NoteColumn”](#) in *Internals Reference*, [Section “NoteCollision”](#) in *Internals Reference*, [Section “RestCollision”](#) in *Internals Reference*.

Known issues and warnings

There is no support for chords 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 [\[Clusters\]](#), [page 123](#)).

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 marked by default.

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. The same variables are used for the independent parts and the combined staff.

```
instrumentOne = \relative c' {
  c4 d e f
  R1
  d'4 c b a
  b4 g2 f4
  e1
}

instrumentTwo = \relative g' {
  R1
  g4 a b c
  d c b a
  g f( e) d
  e1
}

<<
  \new Staff \instrumentOne
  \new Staff \instrumentTwo
  \new Staff \partcombine \instrumentOne \instrumentTwo
>>
```



The notes in the third measure appear only once, although they were specified in both parts. Stem, slur, and tie directions are set automatically, depending whether there is a solo or unison. When needed in polyphony situations, the first part (with context called *one*) always gets up stems, while the second (called *two*) always gets down stems. In solo situations, the first and second parts get marked with ‘Solo’ and ‘Solo II’, respectively. The unisono (*a due*) parts are marked by default with the text “a2”.

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*.

Selected Snippets

Combining two parts on the same staff

The part combiner tool (`\partcombine` command) allows the combination of several different parts on the same staff. Text directions such as “solo” or “a2” are added by default; to remove them, simply set the property `printPartCombineTexts` to `f`. For vocal scores (hymns), there is no need to add “solo/a2” texts, so they should be switched off. However, it might be better not to use it if there are any solos, as they won’t be indicated. In such cases, standard polyphonic notation may be preferable.

This snippet presents the three ways two parts can be printed on a same staff: standard polyphony, `\partcombine` without texts, and `\partcombine` with texts.

```
musicUp = \relative c'' {
  \time 4/4
  a4 c4.( g8) a4 |
  g4 e' g,( a8 b) |
  c b a2.
}

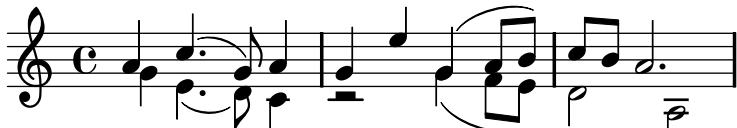


musicDown = \relative c'' {
  g4 e4.( d8) c4 |
  r2 g'4( f8 e) |
  d2 \stemDown a
}

\score {
  <<
  <<
  \new Staff {
    \set Staff.instrumentName = #"Standard polyphony"
```

```

    << \musicUp \\\ \musicDown >>
  }
  \new Staff \with { printPartCombineTexts = ##f } {
    \set Staff.instrumentName = #"PartCombine without texts"
    \partcombine \musicUp \musicDown
  }
  \new Staff {
    \set Staff.instrumentName = #"PartCombine with texts"
    \partcombine \musicUp \musicDown
  }
  >>
>>
\layout {
  indent = 6.0\cm
  \context {
    \Score
    \override SystemStartBar #'collapse-height = #30
  }
}
}

```

Standard polyphony	
PartCombine without texts	
PartCombine with texts	

Changing partcombine texts

When using the automatic part combining feature, the printed text for the solo and unison sections may be changed:

```

\new Staff <<
  \set Staff.soloText = #"girl"
  \set Staff.soloIIText = #"boy"
  \set Staff.aDueText = #"together"
  \partcombine
    \relative c'' {
      g4 g r r
      a2 g
    }
    \relative c'' {
      r4 r a( b)
      a2 g
    }
  }

```

>>



See also

Music Glossary: [Section “a due” in *Music Glossary*](#), [Section “part” in *Music Glossary*](#).

Notation Reference: [Section 1.6.3 \[Writing parts\]](#), page 151.

Snippets: [Section “Simultaneous notes” in *Snippets*](#).

Internals Reference: [Section “PartCombineMusic” in *Internals Reference*](#), [Section “Voice” in *Internals Reference*](#).

Known issues and warnings

`\partcombine` can only accept two voices.

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` and decides when the parts can be combined. When they have different durations they cannot be combined and are given the names `one` and `two`. Consequently, if the arguments switch to differently named [Section “Voice” in *Internals Reference*](#) contexts, the events in those will be ignored. Likewise, partcombining isn’t designed to work with lyrics; when one of the voices is explicitly named in order to attach lyrics to it, the partcombining stops working.

`\partcombine` only observes onset times of notes. It cannot determine whether a previously started note is playing or not, leading to various problems.

Writing music in parallel

Music for multiple parts can be interleaved in input code. The function `\parallelMusic` accepts a list with the names of a number of variables to be created, and a musical expression. The content of alternate measures from the expression become the value of the respective variables, so you can use them afterwards to print the music.

Note: Bar checks `|` must be used, and the measures must be of the same length.

```
\parallelMusic #'(voiceA voiceB voiceC) {
  % Bar 1
  r8 g'16 c'' e'' g' c'' e'' r8 g'16 c'' e'' g' c'' e'' |
  r16 e'8.~ e'4 r16 e'8.~ e'4 |
  c'2 c'2 |

  % Bar 2
  r8 a'16 d'' f'' a' d'' f'' r8 a'16 d'' f'' a' d'' f'' |
  r16 d'8.~ d'4 r16 d'8.~ d'4 |
  c'2 c'2 |}
```

```

}
\new StaffGroup <<
  \new Staff << \voiceA \\\ \voiceB >>
  \new Staff { \clef bass \voiceC }
>>

```



Relative mode may be used. Note that the `\relative` command is not used inside `\parallelMusic` itself. The notes are relative to the preceding note in the voice, not to the previous note in the input – in other words, relative notes for `voiceA` ignore the notes in `voiceB`.

```

\parallelMusic #'(voiceA voiceB voiceC) {
  % Bar 1
  r8 g16 c e g, c e r8 g,16 c e g, c e |
  r16 e8.~ e4          r16 e8.~ e4          |
  c2                  c                    |

  % Bar 2
  r8 a,16 d f a, d f r8 a,16 d f a, d f |
  r16 d8.~ d4          r16 d8.~ d4          |
  c2                  c                    |

}
\new StaffGroup <<
  \new Staff << \relative c'' \voiceA \\\ \relative c' \voiceB >>
  \new Staff \relative c' { \clef bass \voiceC }
>>

```



This works quite well for piano music. This example maps four consecutive measures to four variables:

```

global = {
  \key g \major
  \time 2/4
}

\parallelMusic #'(voiceA voiceB voiceC voiceD) {
  % Bar 1
  a8    b    c    d    |
  d4          e          |
}

```

```

c16 d e fis d e fis g |
a4          a          |

% Bar 2
e8      fis g      a  |
fis4          g      |
e16 fis g a fis g a b |
a4          a          |

% Bar 3 ...
}

\score {
  \new PianoStaff <<
    \new Staff {
      \global
      <<
        \relative c'' \voiceA
        \\
        \relative c'  \voiceB
      >>
    }
    \new Staff {
      \global \clef bass
      <<
        \relative c \voiceC
        \\
        \relative c \voiceD
      >>
    }
  >>
}

```



See also

Learning Manual: [Section “Organizing pieces with variables”](#) in *Learning Manual*.

Snippets: [Section “Simultaneous notes”](#) in *Snippets*.

1.6 Staff notation

Trumpet Bb

Tambourine

Piano

Comodo

p grazioso

This section explains how to influence the appearance of staves, how to print scores with more than one staff, and how to add tempo indications and cue notes to staves.

1.6.1 Displaying staves

This section describes the different methods of creating and grouping staves.

Instantiating new staves

Staves (singular: *staff*) are created with the `\new` or `\context` commands. For details, see [Section 5.1.2 \[Creating contexts\]](#), page 400.

The basic staff context is `Staff`:

```
\new Staff { c4 d e f }
```

The `DrumStaff` context creates a five-line staff set up for a typical drum set. Each instrument is shown with a different symbol. The instruments are entered in drum mode following a `\drummode` command, with each instrument specified by name. For details, see [\[Percussion staves\]](#), page 267.

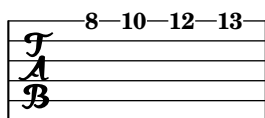
```
\new DrumStaff {
  \drummode { cymc hh ss tomh }
}
```

`RhythmicStaff` creates a single-line staff that only displays the rhythmic values of the input. Real durations are preserved. For details, see [\[Showing melody rhythms\]](#), page 60.

```
\new RhythmicStaff { c4 d e f }
```

`TabStaff` creates a tablature with six strings in standard guitar tuning. For details, see [\[Default tablatures\]](#), page 233.

```
\new TabStaff { c4 d e f }
```



There are two staff contexts specific for the notation of ancient music: `MensuralStaff` and `VaticanaStaff`. They are described in [\[Pre-defined contexts\]](#), page 300.

The `GregorianTranscriptionStaff` context creates a staff to notate modern Gregorian chant. It does not show bar lines.

```
\new GregorianTranscriptionStaff { c4 d e f e d }
```



New single staff contexts may be defined. For details, see [Section 5.1.6 \[Defining new contexts\]](#), page 406.

See also

Music Glossary: [Section “staff” in *Music Glossary*](#), [Section “staves” in *Music Glossary*](#).

Notation Reference: [Section 5.1.2 \[Creating contexts\]](#), page 400, [\[Percussion staves\]](#), page 267, [\[Showing melody rhythms\]](#), page 60, [\[Default tablatures\]](#), page 233, [\[Pre-defined contexts\]](#), page 300, [\[Staff symbol\]](#), page 142, [\[Gregorian chant contexts\]](#), page 308, [\[Mensural contexts\]](#), page 301, [Section 5.1.6 \[Defining new contexts\]](#), page 406.

Snippets: [Section “Staff notation” in *Snippets*](#).

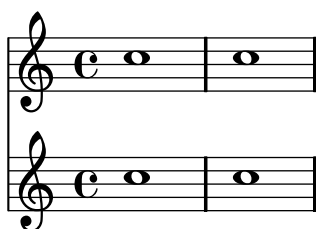
Internals Reference: [Section “Staff” in *Internals Reference*](#), [Section “DrumStaff” in *Internals Reference*](#), [Section “GregorianTranscriptionStaff” in *Internals Reference*](#), [Section “RhythmicStaff” in *Internals Reference*](#), [Section “TabStaff” in *Internals Reference*](#), [Section “MensuralStaff” in *Internals Reference*](#), [Section “VaticanaStaff” in *Internals Reference*](#), [Section “StaffSymbol” in *Internals Reference*](#).

Grouping staves

Various contexts exist to group single staves together in order to form multi-stave systems. Each grouping context sets the style of the system start delimiter and the behavior of bar lines.

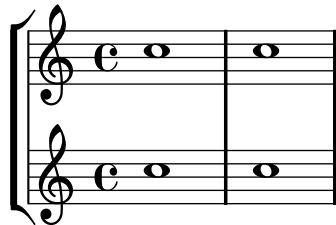
If no context is specified, the default properties will be used: the group is started with a vertical line, and the bar lines are not connected.

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



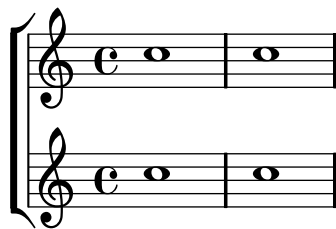
In the `StaffGroup` context, the group is started with a bracket and bar lines are drawn through all the staves.

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



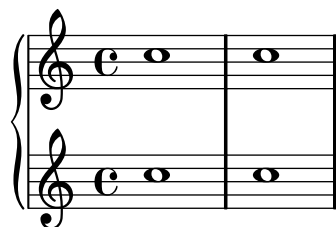
In a `ChoirStaff`, the group starts with a bracket, but bar lines are not connected.

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



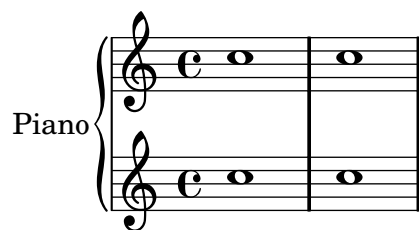
In a `GrandStaff`, the group begins with a brace, and bar lines are connected between the staves.

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



The `PianoStaff` is identical to a `GrandStaff`, except that it supports printing the instrument name directly. For details, see [\[Instrument names\]](#), page 154.

```
\new PianoStaff <<
  \set PianoStaff.instrumentName = #"Piano"
  \new Staff { c1 c }
  \new Staff { c1 c }
>>
```



Each staff group context sets the property `systemStartDelimiter` to one of the following values: `SystemStartBar`, `SystemStartBrace`, or `SystemStartBracket`. A fourth delimiter, `SystemStartSquare`, is also available, but it must be explicitly specified.

New staff group contexts may be defined. For details, see [Section 5.1.6 \[Defining new contexts\]](#), page 406.

Selected Snippets

Use square bracket at the start of a staff group

The system start delimiter `SystemStartSquare` can be used by setting it explicitly in a `StaffGroup` or `ChoirStaffGroup` context.

```
\score {
  \new StaffGroup { <<
    \set StaffGroup.systemStartDelimiter = #'SystemStartSquare
    \new Staff { c'4 d' e' f' }
    \new Staff { c'4 d' e' f' }
  >> }
}
```



Display bracket with only one staff in a system

If there is only one staff in one of the staff types `ChoirStaff` or `StaffGroup`, the bracket and the starting bar line will not be displayed as standard behavior. This can be changed by overriding the relevant properties.

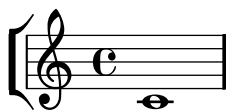
Note that in contexts such as `PianoStaff` and `GrandStaff` where the systems begin with a brace instead of a bracket, another property has to be set, as shown on the second system in the example.

```
\markup \left-column {
  \score {
    \new StaffGroup <<
      % Must be lower than the actual number of staff lines
      \override StaffGroup.SystemStartBracket #'collapse-height = #1
      \override Score.SystemStartBar #'collapse-height = #1
      \new Staff {
        c'1
      }
    >>
    \layout { }
  }
}
```

```

\score {
  \new PianoStaff <<
    \override PianoStaff.SystemStartBrace #'collapse-height = #1
    \override Score.SystemStartBar #'collapse-height = #1
    \new Staff {
      c'1
    }
  >>
  \layout { }
}

```



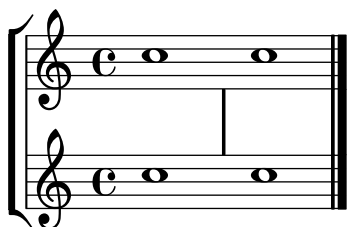
Mensurstriche layout (bar lines between the staves)

The mensurstriche-layout where the bar lines do not show on the staves but between staves can be achieved with a `StaffGroup` instead of a `ChoirStaff`. The bar line on staves is blanked out by setting the `transparent` property.

```

global = {
  \override Staff.BarLine #'transparent = ##t
  s1 s
  % the final bar line is not interrupted
  \revert Staff.BarLine #'transparent
  \bar "|."
}
\new StaffGroup \relative c'' {
  <<
    \new Staff { << \global { c1 c } >> }
    \new Staff { << \global { c c } >> }
  >>
}

```



See also

Music Glossary: [Section “brace” in Music Glossary](#), [Section “bracket” in Music Glossary](#), [Section “grand staff” in Music Glossary](#).

Notation Reference: [\[Instrument names\]](#), page 154, [Section 5.1.6 \[Defining new contexts\]](#), page 406.

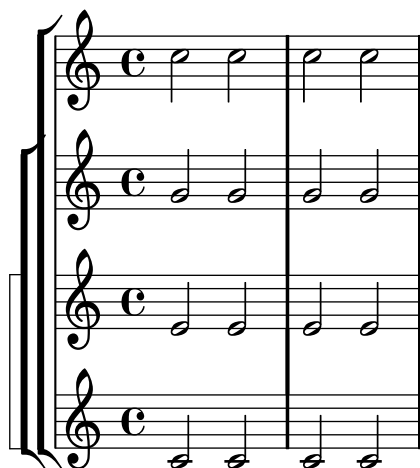
Snippets: Section “Staff notation” in *Snippets*.

Internals Reference: Section “Staff” in *Internals Reference*, Section “StaffGroup” in *Internals Reference*, Section “ChoirStaff” in *Internals Reference*, Section “GrandStaff” in *Internals Reference*, Section “PianoStaff” in *Internals Reference*, Section “SystemStartBar” in *Internals Reference*, Section “SystemStartBrace” in *Internals Reference*, Section “SystemStartBracket” in *Internals Reference*, Section “SystemStartSquare” in *Internals Reference*.

Nested staff groups

Staff-group contexts can be nested to arbitrary depths. In this case, each child context creates a new bracket adjacent to the bracket of its parent group.

```
\new StaffGroup <<
  \new Staff { c2 c | c2 c }
  \new StaffGroup <<
    \new Staff { g2 g | g2 g }
    \new StaffGroup \with {
      systemStartDelimiter = #'SystemStartSquare
    }
    <<
      \new Staff { e2 e | e2 e }
      \new Staff { c2 c | c2 c }
    >>
  >>
>>
```



New nested staff group contexts can be defined. For details, see [Section 5.1.6 \[Defining new contexts\]](#), page 406.

Selected Snippets

Nesting staves

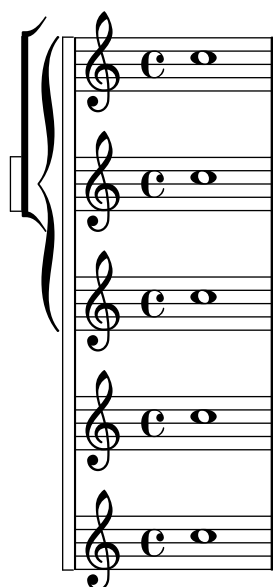
The property `systemStartDelimiterHierarchy` can be used to make more complex nested staff groups. The command `\set StaffGroup.systemStartDelimiterHierarchy` takes an alphabetical list of the number of staves produced. Before each staff a system start delimiter can be given. It has to be enclosed in brackets and takes as much staves as the brackets enclose. Elements in the list can be omitted, but the first bracket takes always the complete number of staves. The possibilities are `SystemStartBar`, `SystemStartBracket`, `SystemStartBrace`, and `SystemStartSquare`.

```

\new StaffGroup
\relative c'' <<
  \set StaffGroup.systemStartDelimiterHierarchy
    = #'(SystemStartSquare (SystemStartBrace (SystemStartBracket a
                                              (SystemStartSquare b) ) c ) d)

  \new Staff { c1 }
  \new Staff { c1 }
  \new Staff { c1 }
  \new Staff { c1 }
  \new Staff { c1 }
>>

```



See also

Notation Reference: [\[Grouping staves\]](#), page 137, [\[Instrument names\]](#), page 154, [Section 5.1.6 \[Defining new contexts\]](#), page 406.

Snippets: [Section “Staff notation” in *Snippets*](#).

Internals Reference: [Section “StaffGroup” in *Internals Reference*](#), [Section “ChoirStaff” in *Internals Reference*](#), [Section “SystemStartBar” in *Internals Reference*](#), [Section “SystemStartBrace” in *Internals Reference*](#), [Section “SystemStartBracket” in *Internals Reference*](#), [Section “SystemStartSquare” in *Internals Reference*](#).

1.6.2 Modifying single staves

This section explains how to change specific attributes of one staff: for example, modifying the number of staff lines or the staff size. Methods to start and stop staves and set ossia sections are also described.

Staff symbol

The lines of a staff belong to the `StaffSymbol` grob. `StaffSymbol` properties can be modified to change the appearance of a staff, but they must be modified before the staff is created.

The number of staff lines may be changed. The clef position and the position of middle C may need to be modified to fit the new staff. For an explanation, refer to the snippet section in [\[Clef\]](#), page 13.

```
\new Staff \with {
  \override StaffSymbol #'line-count = #3
}
{ d4 d d d }
```



Staff line thickness can be modified. The thickness of ledger lines and stems are also affected, since they depend on staff line thickness.

```
\new Staff \with {
  \override StaffSymbol #'thickness = #3
}
{ e4 d c b }
```



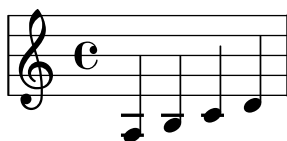
Ledger line thickness can be set independently of staff line thickness. In the example the two numbers are factors multiplying the staff line thickness and the staff line spacing. The two contributions are added to give the ledger line thickness.

```
\new Staff \with {
  \override StaffSymbol #'ledger-line-thickness = #'(1 . 0.2)
}
{ e4 d c b }
```



The distance between staff lines can be changed. This setting affects the spacing of ledger lines as well.

```
\new Staff \with {
  \override StaffSymbol #'staff-space = #1.5
}
{ a4 b c d }
```



Further details about the properties of `StaffSymbol` can be found in [Section “staff-symbol-interface”](#) in *Internals Reference*.

Modifications to staff properties in the middle of a score can be placed between `\stopStaff` and `\startStaff`:

```

c2 c
\stopStaff
\override Staff.StaffSymbol #'line-count = #2
\startStaff
b2 b
\stopStaff
\revert Staff.StaffSymbol #'line-count
\startStaff
a2 a

```

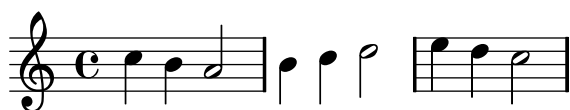


In general, `\startStaff` and `\stopStaff` can be used to stop or start a staff in the middle of a score.

```

c4 b a2
\stopStaff
b4 c d2
\startStaff
e4 d c2

```



Predefined commands

`\startStaff`, `\stopStaff`.

Selected Snippets

Making some staff lines thicker than the others

For pedagogical purposes, a staff line can be thickened (e.g., the middle line, or to emphasize the line of the G clef). This can be achieved by adding extra lines very close to the line that should be emphasized, using the `line-positions` property of the `StaffSymbol` object.

```

{
  \override Staff.StaffSymbol #'line-positions = #'(-4 -2 -0.2 0 0.2 2 4)
  d'4 e' f' g'
}

```



See also

Music Glossary: [Section “line”](#) in *Music Glossary*, [Section “ledger line”](#) in *Music Glossary*, [Section “staff”](#) in *Music Glossary*.

Notation Reference: [\[Clef\]](#), page 13.

Snippets: [Section “Staff notation”](#) in *Snippets*.

Internals Reference: [Section “StaffSymbol”](#) in *Internals Reference*, [Section “staff-symbol-interface”](#) in *Internals Reference*.

Ossia staves

Ossia staves can be set by creating a new simultaneous staff in the appropriate location:

```
\new Staff \relative c'' {
  c4 b d c
  <<
    { c4 b d c }
    \new Staff { e4 d f e }
  >>
  c4 b c2
}
```



However, the above example is not what is usually desired. To create ossia staves that are above the original staff, have no time signature or clef, and have a smaller font size, tweaks must be used. The Learning Manual describes a specific technique to achieve this goal, beginning with [Section “Nesting music expressions” in *Learning Manual*](#).

The following example uses the `alignAboveContext` property to align the ossia staff. This method is most appropriate when only a few ossia staves are needed.

```
\new Staff = main \relative c'' {
  c4 b d c
  <<
    { c4 b d c }

    \new Staff \with {
      \remove "Time_signature_engraver"
      alignAboveContext = #"main"
      fontSize = #-3
      \override StaffSymbol #'staff-space = #(magstep -3)
      \override StaffSymbol #'thickness = #(magstep -3)
      firstClef = ##f
    }
    { e4 d f e }
  >>
  c4 b c2
}
```



If many isolated ossia staves are needed, creating an empty `Staff` context with a specific *context id* may be more appropriate; the ossia staves may then be created by *calling* this context and using `\startStaff` and `\stopStaff` at the desired locations. The benefits of this method are more apparent if the piece is longer than the following example.

```

<<
\new Staff = ossia \with {
  \remove "Time_signature_engraver"
  \override Clef #'transparent = ##t
  fontSize = #-3
  \override StaffSymbol #'staff-space = #(magstep -3)
  \override StaffSymbol #'thickness = #(magstep -3)
}
{ \stopStaff s1*6 }

\new Staff \relative c' {
  c4 b c2
  <<
    { e4 f e2 }
    \context Staff = ossia {
      \startStaff e4 g8 f e2 \stopStaff
    }
  >>
  g4 a g2 \break
  c4 b c2
  <<
    { g4 a g2 }
    \context Staff = ossia {
      \startStaff g4 e8 f g2 \stopStaff
    }
  >>
  e4 d c2
}
>>

```



4



Using the `\RemoveEmptyStaffContext` command to create ossia staves may be used as an alternative. This method is most convenient when ossia staves occur immediately following a line break. For more information about `\RemoveEmptyStaffContext`, see [\[Hiding staves\]](#), page 148.

```

<<
\new Staff = ossia \with {
  \remove "Time_signature_engraver"
  \override Clef #'transparent = ##t
  fontSize = #-3

```

```

\override StaffSymbol #'staff-space = #(magstep -3)
\override StaffSymbol #'thickness = #(magstep -3)
} \relative c' {
  R1*3
  c4 e8 d c2
}
\new Staff \relative c' {
  c4 b c2
  e4 f e2
  g4 a g2 \break
  c4 b c2
  g4 a g2
  e4 d c2
}
>>

\layout {
  \context {
    \RemoveEmptyStaffContext
    \override VerticalAxisGroup #'remove-first = ##t
  }
}

```



Selected Snippets

Vertically aligning ossia and lyrics

This snippet demonstrates the use of the context properties `alignBelowContext` and `alignAboveContext` to control the positioning of lyrics and ossia.

```

\paper {
  ragged-right = ##t
}

\relative c' <<
  \new Staff = "1" { c4 c s2 }
  \new Staff = "2" { c4 c s2 }
  \new Staff = "3" { c4 c s2 }
  { \skip 2
    <<
      \lyrics {
        \set alignBelowContext = #"1"

```

```

    lyrics4 below
  }
  \new Staff \with {
    alignAboveContext = #"3"
    fontSize = #-2
    \override StaffSymbol #'staff-space = #(magstep -2)
    \remove "Time_signature_engraver"
  } {
    \times 4/6 {
      \override TextScript #'padding = #3
      c8[~"ossia above" d e d e f]
    }
  }
  >>
}
>>

```



See also

Music Glossary: [Section “ossia”](#) in *Music Glossary*, [Section “staff”](#) in *Music Glossary*, [Section “Frenched staff”](#) in *Music Glossary*.

Learning Manual: [Section “Nesting music expressions”](#) in *Learning Manual*, [Section “Size of objects”](#) in *Learning Manual*, [Section “Length and thickness of objects”](#) in *Learning Manual*.

Notation Reference: [\[Hiding staves\]](#), page 148.

Snippets: [Section “Staff notation”](#) in *Snippets*.

Internals Reference: [Section “StaffSymbol”](#) in *Internals Reference*.

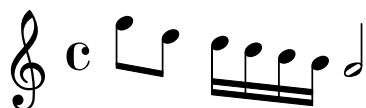
Hiding staves

Staff lines can be hidden by removing the `Staff_symbol_engraver` from the `Staff` context. As an alternative, `\stopStaff` may be used.

```

\new Staff \with {
  \remove "Staff_symbol_engraver"
}
\relative c''' { a8 f e16 d c b a2 }

```

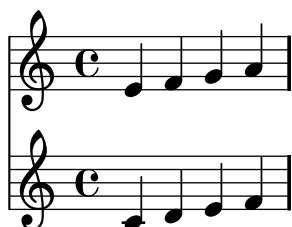


Empty staves can be hidden by setting the `\RemoveEmptyStaffContext` command in the `\layout` block. In orchestral scores, this style is known as ‘Frenched Score’. By default, this command hides and removes all empty staves in a score except for those in the first system.

Note: A staff is considered empty when it contains only multi-measure rests, skips, spacer rests, or a combination of these elements.

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

```
\relative c' <<
  \new Staff {
    e4 f g a \break
    b1 \break
    a4 b c2
  }
  \new Staff {
    c,4 d e f \break
    R1 \break
    f4 g c,2
  }
>>
```



`\RemoveEmptyStaffContext` can also be used to create ossia sections for a staff. For details, see [\[Ossia staves\]](#), page 145.

The `\AncientRemoveEmptyStaffContext` command may be used to hide empty staves in ancient music contexts. Similarly, `\RemoveEmptyRhythmicStaffContext` may be used to hide empty `RhythmicStaff` contexts.

Predefined commands

`\RemoveEmptyStaffContext,` `\AncientRemoveEmptyStaffContext,`
`\RemoveEmptyRhythmicStaffContext.`

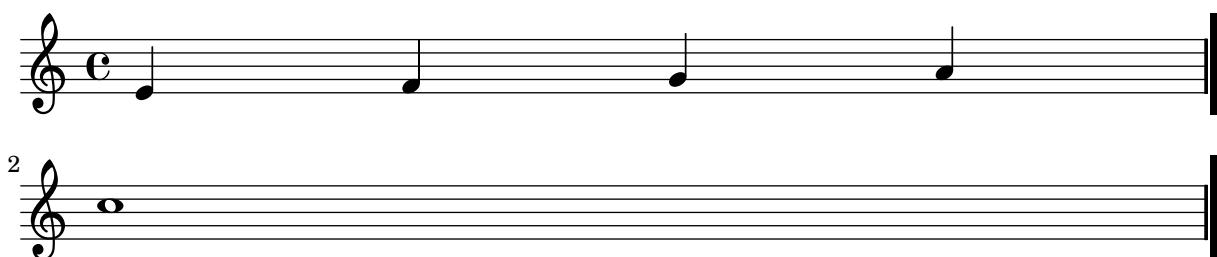
Selected Snippets

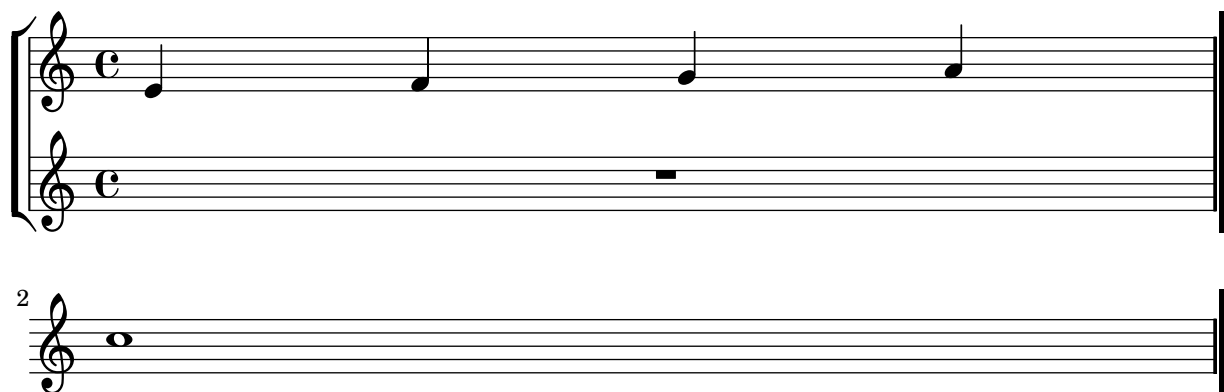
Removing the first empty line

The first empty staff can also be removed from the score by setting the `VerticalAxisGroup` property `remove-first`. This can be done globally inside the `\layout` block, or locally inside the specific staff that should be removed. In the latter case, you have to specify the context (`Staff` applies only to the current staff) in front of the property.

The lower staff of the second staff group is not removed, because the setting applies only to the specific staff inside of which it is written.

```
\layout {
  \context {
    \Staff \RemoveEmptyStaves
    % To use the setting globally, uncomment the following line:
    % \override VerticalAxisGroup #'remove-first = ##t
  }
}
\new StaffGroup <<
  \new Staff \relative c' {
    e4 f g a \break
    c1
  }
  \new Staff {
    % To use the setting globally, comment this line,
    % uncomment the line in the \layout block above
    \override Staff.VerticalAxisGroup #'remove-first = ##t
    R1 \break
    R
  }
>>
\new StaffGroup <<
  \new Staff \relative c' {
    e4 f g a \break
    c1
  }
  \new Staff {
    R1 \break
    R
  }
>>
```





See also

Music Glossary: Section “Frenched staff” in *Music Glossary*.

Learning Manual: Section “Visibility and color of objects” in *Learning Manual*.

Notation Reference: Section 5.1.5 [Changing context default settings], page 405, [Staff symbol], page 142, [Ossia staves], page 145, [Hidden notes], page 168, [Invisible rests], page 47, Section 5.4.7 [Visibility of objects], page 426.

Snippets: Section “Staff notation” in *Snippets*.

Internals Reference: Section “ChordNames” in *Internals Reference*, Section “FiguredBass” in *Internals Reference*, Section “Lyrics” in *Internals Reference*, Section “Staff” in *Internals Reference*, Section “VerticalAxisGroup” in *Internals Reference*, Section “Staff_symbol_engraver” in *Internals Reference*.

Known issues and warnings

Removing `Staff_symbol_engraver` also hides bar lines. If bar line visibility is forced, formatting errors may occur. In this case, use the following overrides instead of removing the engraver:

```
\override StaffSymbol #'stencil = ##f
\override NoteHead #'no-ledgers = ##t
```

For the Known issues and warnings associated with `\RemoveEmptyStaffContext` see Section 5.1.5 [Changing context default settings], page 405.

1.6.3 Writing parts

This section explains how to insert tempo indications and instrument names into a score. Methods to quote other voices and format cue notes are also described.

Metronome marks

A basic metronome mark is simple to write:

```
\tempo 4 = 120
c2 d
e4. d8 c2
```



Tempo indications with text can be used instead:

```
\tempo "Allegretto"
c4 e d c
b4. a16 b c4 r4
```



Combining a metronome mark and text will automatically place the metronome mark within parentheses:

```
\tempo "Allegro" 4 = 160
g4 c d e
d4 b g2
```



In general, the text can be any markup object:

```
\tempo \markup { \italic Faster } 4 = 132
a8-. r8 b-. r gis-. r a-. r
```



A parenthesized metronome mark with no textual indication may be written by including an empty string in the input:

```
\tempo "" 8 = 96
d4 g e c
```



Selected Snippets

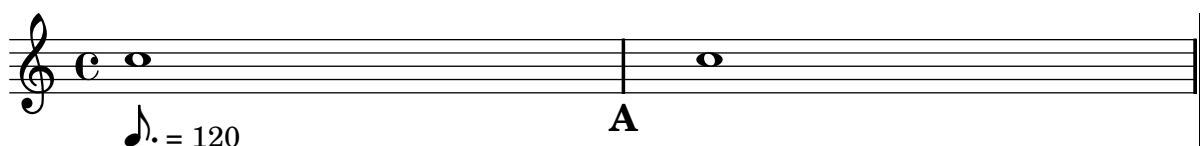
Printing metronome and rehearsal marks below the staff

By default, metronome and rehearsal marks are printed above the staff. To place them below the staff simply set the `direction` property of `MetronomeMark` or `RehearsalMark` appropriately.

```
\layout { ragged-right = ##f }

{
  % Metronome marks below the staff
  \override Score.MetronomeMark #'direction = #DOWN
  \tempo 8. = 120
  c''1

  % Rehearsal marks below the staff
  \override Score.RehearsalMark #'direction = #DOWN
  \mark \default
  c''1
}
```



Changing the tempo without a metronome mark

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

```
\score {
  \new Staff \relative c' {
    \tempo 4 = 160
    c4 e g b
    c4 b d c
    \set Score.tempoHideNote = ##t
    \tempo 4 = 96
    d,4 fis a cis
    d4 cis e d
  }
  \layout { }
  \midi { }
}
```



Creating metronome marks in markup mode

New metronome marks can be created in markup mode, but they will not change the tempo in MIDI output.

```
\relative c' {
  \tempo \markup {
    \concat {
      (
        \smaller \general-align #Y #DOWN \note #"16." #1
        " = "
        \smaller \general-align #Y #DOWN \note #"8" #1
      )
    }
  }
  c1
  c4 c' c,2
}
```



For more details, see [Section 1.8.2 \[Formatting text\]](#), page 182.

See also

Music Glossary: [Section “metronome”](#) in *Music Glossary*, [Section “metronomic indication”](#) in *Music Glossary*, [Section “tempo indication”](#) in *Music Glossary*, [Section “metronome mark”](#) in *Music Glossary*.

Notation Reference: [Section 1.8.2 \[Formatting text\]](#), page 182, [Section 3.5 \[MIDI output\]](#), page 347.

Snippets: [Section “Staff notation”](#) in *Snippets*.

Internals Reference: [Section “MetronomeMark”](#) in *Internals Reference*.

Instrument names

Instrument names can be printed on the left side of staves in the `Staff` and `PianoStaff` contexts. The value of `instrumentName` is used for the first staff, and the value of `shortInstrumentName` is used for all succeeding staves.

```
\set Staff.instrumentName = #"Violin "
\set Staff.shortInstrumentName = #"Vln "
c4.. g'16 c4.. g'16
\break
c1
```



Markup mode can be used to create more complicated instrument names:

```
\set Staff.instrumentName = \markup {
  \column { "Clarineti"
    \line { "in B" \smaller \flat } } }
c4 c,16 d e f g2
```



When two or more staff contexts are grouped together, the instrument names and short instrument names are centered by default. To center multi-line instrument names, `\center-column` must be used:

```
<<
\new Staff {
  \set Staff.instrumentName = #"Flute"
  f2 g4 f
}
\new Staff {
  \set Staff.instrumentName = \markup \center-column {
    Clarinet
    \line { "in B" \smaller \flat }
  }
}
```

```

    c4 b c2
  }
>>

```



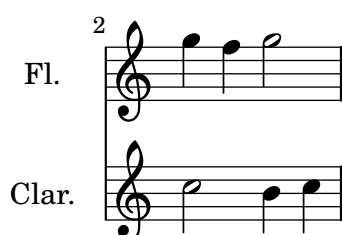
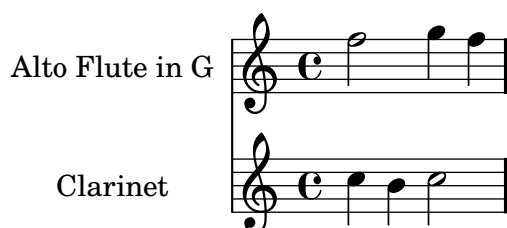
However, if the instrument names are longer, the instrument names in a staff group may not be centered unless the `indent` and `short-indent` settings are increased. For details about these settings, see [\[Horizontal dimensions\]](#), page 360.

```

\layout {
  indent = 3.0\cm
  short-indent = 1.5\cm
}

\relative c' ' <<
  \new Staff {
    \set Staff.instrumentName = #"Alto Flute in G"
    \set Staff.shortInstrumentName = #"Fl."
    f2 g4 f \break
    g4 f g2
  }
  \new Staff {
    \set Staff.instrumentName = #"Clarinet"
    \set Staff.shortInstrumentName = #"Clar."
    c,4 b c2 \break
    c2 b4 c
  }
>>

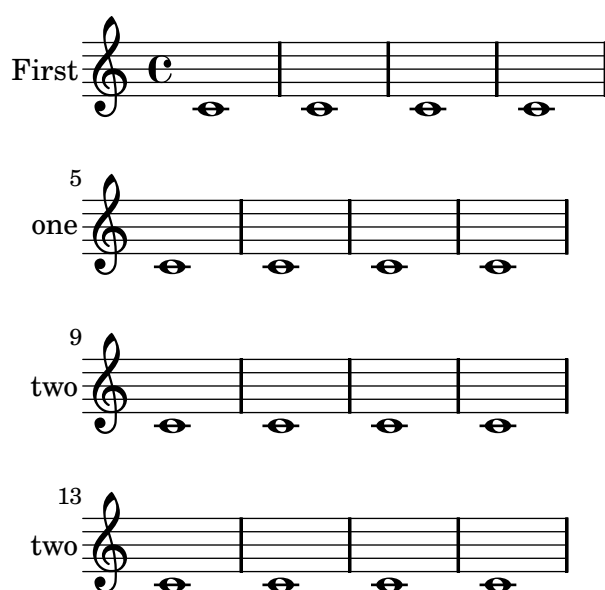
```



To add instrument names to other contexts (such as `GrandStaff`, `ChoirStaff`, or `StaffGroup`), `Instrument_name_engraver` must be added to that context. For details, see [Section 5.1.4 \[Modifying context plug-ins\]](#), page 403.

Instrument names may be changed in the middle of a piece. However, remember that `instrumentName` will not be displayed in the middle of the piece, as it only appears on the first staff:

```
\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
```



If an instrument *switch* is needed, `\addInstrumentDefinition` may be used in combination with `\instrumentSwitch` to create a detailed list of the necessary changes for the switch. The `\addInstrumentDefinition` command has two arguments: an identifying string, and an association list of context properties and values to be used for the instrument. It must be placed in the toplevel scope. `\instrumentSwitch` is used in the music expression to declare the instrument switch:

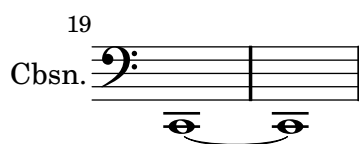
```
\addInstrumentDefinition #"contrabassoon"
#`((instrumentTransposition . ,(ly:make-pitch -1 0 0))
  (shortInstrumentName . "Cbsn.")
  (clefGlyph . "clefs.F")
  (middleCPosition . 6)
  (clefPosition . 2)
  (instrumentCueName . ,(make-bold-markup "cbsn."))
  (midiInstrument . "bassoon"))

\new Staff \with {
  instrumentName = #"Bassoon"
}
\relative c' {
  \clef tenor
  \compressFullBarRests
  c2 g'
```

```

R1*16
\instrumentSwitch "contrabassoon"
c,,2 g \break
c,1 ~ | c1
}

```



See also

Notation Reference: [\[Horizontal dimensions\]](#), page 360, [Section 5.1.4 \[Modifying context plug-ins\]](#), page 403.

Snippets: [Section “Staff notation” in *Snippets*](#).

Internals Reference: [Section “InstrumentName” in *Internals Reference*](#), [Section “PianoStaff” in *Internals Reference*](#), [Section “Staff” in *Internals Reference*](#).

Quoting other voices

It is very common for one voice to double some of the music from another voice. For example, the first and second violins may play the same notes during a passage of music. In LilyPond this is accomplished by letting one voice *quote* the other voice without having to re-enter it.

Before a part can be quoted, the `\addQuote` command must be used to initialize the quoted fragment. This command must be used in the toplevel scope. The first argument is an identifying string, and the second is a music expression:

```

flute = \relative c'' {
  a4 gis g gis
}
\addQuote "flute" { \flute }

```

The `\quoteDuring` command is used to indicate the point where the quotation begins. It is followed by two arguments: the name of the quoted voice, as defined with `\addQuote`, and a music expression that indicates the duration of the quote, usually spacer rests or multi-measure rests. The corresponding music (including all articulations, dynamics, markup, etc.) from the quoted voice is inserted into the music expression:

```

flute = \relative c'' {
  a4 gis g->\f gis^\markup{quoted}
}
\addQuote "flute" { \flute }

\relative c' {
  c4 cis \quoteDuring #"flute" { s2 }
}

```



If the music expression used for `\quoteDuring` contains anything but a spacer rest or multi-measure rest, a polyphonic situation is created, which is often not desirable:

```
flute = \relative c'' {
  a4 gis g gis
}
\addQuote "flute" { \flute }

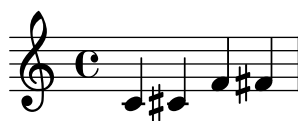
\relative c' {
  c4 cis \quoteDuring #"flute" { c4 b }
}
```



Quotations recognize instrument transposition settings for both the source and target instruments if the `\transposition` command is used. For details about `\transposition`, see [\[Instrument transpositions\]](#), page 19.

```
clarinet = \relative c'' {
  \transposition bes
  a4 gis g gis
}
\addQuote "clarinet" { \clarinet }

\relative c' {
  c4 cis \quoteDuring #"clarinet" { s2 }
}
```



It is possible to tag quotations with unique names in order to process them in different ways. For details about this procedure, see [\[Using tags\]](#), page 341.

It is also possible to adjust which objects from the original voice are quoted by changing the `quotedEventTypes` property. By default, its value is `#'(StreamEvent)`, which means that everything is quoted. Setting it to e.g. `#'(note-event rest-event tie-event)` causes lilypond to quote only notes, rests and ties, but no articulations, markup or dynamics.

```
clarinet = \relative c'' {
  a4 gis g->\f gis~\markup{quoted}
}
\addQuote "clarinet" { \clarinet }

\relative c' {
  \set Score.quotedEventTypes = #'(note-event rest-event tie-event)
  c4 cis \quoteDuring #"clarinet" { s2 }
}
```



Selected Snippets

Quoting another voice with transposition

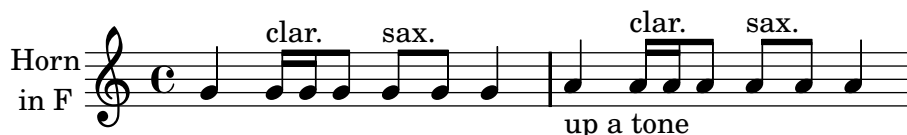
Quotations take into account the transposition of both source and target. In this example, all instruments play sounding middle C; the target is an instrument in F. The target part may be transposed using `\transpose`. In this case, all the pitches (including the quoted ones) are transposed.

```
\addQuote clarinet {
  \transpose bes
  \repeat unfold 8 { d'16 d' d'8 }
}

\addQuote sax {
  \transpose es'
  \repeat unfold 16 { a8 }
}

quoteTest = {
  % french horn
  \transpose f
  g'4
  << \quoteDuring #"clarinet" { \skip 4 } s4^"clar." >>
  << \quoteDuring #"sax" { \skip 4 } s4^"sax." >>
  g'4
}

{
  \set Staff.instrumentName =
    \markup {
      \center-column { Horn \line { in F } }
    }
  \quoteTest
  \transpose c' d' << \quoteTest s4_"up a tone" >>
}
```



Quoting another voice

The `quotedEventTypes` property determines the music event types which should be quoted. The default value is `(note-event rest-event tie-event beam-event tuplet-span-event)`, which means that only the notes, rests, ties, beams and tuplets of the quoted voice will appear in the `\quoteDuring` expression. In the following example, a 16th rest is not quoted since `rest-event` is not in `quotedEventTypes`.

For a list of event types, consult the “Music classes” section of the Internals Reference.

```
quoteMe = \relative c' {
  fis4 r16 a8.-> b4\ff c
}
```

```

\addQuote quoteMe \quoteMe

original = \relative c'' {
  c8 d s2
  \once \override NoteColumn #'ignore-collision = ##t
  es8 gis8
}

<<
  \new Staff {
    \set Staff.instrumentName = #"quoteMe"
    \quoteMe
  }
  \new Staff {
    \set Staff.instrumentName = #"orig"
    \original
  }
  \new Staff \relative c'' <<
    \set Staff.instrumentName = #"orig+quote"
    \set Staff.quotedEventTypes =
      #'(note-event articulation-event)
    \original
    \new Voice {
      s4
      \set fontSize = #-4
      \override Stem #'length-fraction = #(magstep -4)
      \quoteDuring #"quoteMe" { \skip 2. }
    }
  }
>>
>>

```

quoteMe

orig

orig+quote

See also

Notation Reference: [\[Instrument transpositions\]](#), page 19, [\[Using tags\]](#), page 341.

Snippets: [Section “Staff notation” in *Snippets*](#).

Internals Reference: [Section “QuoteMusic” in *Internals Reference*](#), [Section “Voice” in *Internals Reference*](#).

Known issues and warnings

Only the contents of the first **Voice** occurring in an `\addQuote` command will be considered for quotation, so *music* cannot 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`.

Formatting cue notes

The previous section explains how to create quotations. The `\cueDuring` command is a more specialized form of `\quoteDuring`, being particularly useful for inserting cue notes into a part. The syntax is as follows:

```
\cueDuring #partname #voice music
```

This command copies only the notes and rests from the corresponding measures from *part-name* into a **CueVoice** context. The **CueVoice** is created implicitly, and occurs simultaneously with *music*, which creates a polyphonic situation. The *voice* argument determines whether the cue notes should be notated as a first or second voice; **UP** corresponds to the first voice, and **DOWN** corresponds to the second.

```

oboe = \relative c'' {
    r2 r8 d16(\f f e g f a)
    g8 g16 g g2.
}
\addQuote "oboe" { \oboe }

```

```
\new Voice \relative c'' {
  \cueDuring #"oboe" #UP { R1 }
  g2 c,
}
```



In the above example, the **Voice** context had to be explicitly declared, or else the entire music expression would belong to the **CueVoice** context.

It is possible to adjust which aspects of the music are quoted with `\cueDuring` by setting the `quotedCueEventTypes` property. Its default value is `#'(note-event rest-event tie-event beam-event tuplet-span-event)`, which means that only notes, rest, ties, beams and tuplets are quoted, but not articulations, dynamic marks, markup etc.

```

oboe = \relative c'' {
  r2 r8 d16(\f f e g f a)
  g8 g16 g g2.
}
\addQuote "oboe" { \oboe }

```

[illegible]

```

                                dynamic-event slur-event)
\cueDuring #"oboe" #UP { R1 }
g2 c,
}

```



The name of the cued instrument can be printed by setting the `instrumentCueName` property in the `CueVoice` context.

```

oboe = \relative c''' {
  g4 r8 e16 f e4 d
}
\addQuote "oboe" { \oboe }

\new Staff \relative c'' <<
  \new CueVoice \with {
    instrumentCueName = "ob."
  }
  \new Voice {
    \cueDuring #"oboe" #UP { R1 }
    g4. b8 d2
  }
}
>>

```



In addition to printing the name of the cued instrument, when cue notes end, the name of the original instrument should be printed, and any other changes introduced by the cued part should be undone. This can be accomplished by using `\addInstrumentDefinition` and `\instrumentSwitch`. For an example and explanation, see [\[Instrument names\]](#), page 154.

The `\killCues` command removes cue notes from a music expression. This can be useful if cue notes need to be removed from a part but may be restored at a later time.

```

flute = \relative c''' {
  r2 cis2 r2 dis2
}
\addQuote "flute" { \flute }

\new Voice \relative c'' {
  \killCues {
    \cueDuring #"flute" #UP { R1 }
    g4. b8 d2
  }
}

```



The `\transposedCueDuring` command is useful for adding instrumental cues from a completely different register. The syntax is similar to `\cueDuring`, but it requires one extra argument to specify the transposition of the cued instrument. For more information about transposition, see [\[Instrument transpositions\]](#), page 19.

```
piccolo = \relative c''' {
  \clef "treble^8"
  R1
  c8 c c e g2
  a4 g g2
}
\addQuote "piccolo" { \piccolo }

cbassoon = \relative c, {
  \clef "bass_8"
  c4 r g r
  \transposedCueDuring #"piccolo" #UP c,, { R1 }
  c4 r g r
}

<<
  \new Staff = "piccolo" \piccolo
  \new Staff = "cbassoon" \cbassoon
>>
```



It is possible to tag cued parts with unique names in order to process them in different ways. For details about this procedure, see [\[Using tags\]](#), page 341.

A `CueVoice` context may be created explicitly if notes of a smaller size are required, for example to set an alternative sequence of notes more suitable for a higher or lower voice.

```
\time 12/8
\key ees \major
g4 ees8 f4 g8
\stemDown
<<
  { d4. bes4 c8 }
  \new CueVoice
  { g'4. f4 ees8 }
>>
\stemUp
d2. d2.
```



See also

Notation Reference: [\[Instrument transpositions\]](#), page 19, [\[Instrument names\]](#), page 154, [\[Using tags\]](#), page 341.

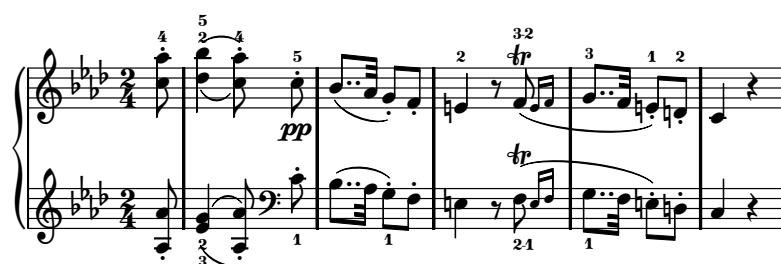
Snippets: [Section “Staff notation” in *Snippets*](#).

Internals Reference: [Section “CueVoice” in *Internals Reference*](#), [Section “Voice” in *Internals Reference*](#).

Known issues and warnings

Collisions can occur with rests, when using `\cueDuring`, between `Voice` and `CueVoice` contexts.

1.7 Editorial annotations



This section discusses the various ways to change the appearance of notes and add analysis or educational emphasis.

1.7.1 Inside the staff

This section discusses how to add emphasis to elements that are inside the staff.

Selecting notation font size

The font size of notation elements may be altered. It does not change the size of variable symbols, such as beams or slurs.

Note: For font sizes of text, see [\[Selecting font and font size\]](#), page 184.

```
\huge
c4.-> d8---3
\large
c4.-> d8---3
\normalsize
c4.-> d8---3
\small
c4.-> d8---3
\tiny
c4.-> d8---3
\teeny
c4.-> d8---3
```



Internally, this sets the `fontSize` property. This in turn causes 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 of 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.

```
\set fontSize = #3
c4.-> d8---3
\override NoteHead #'font-size = #-4
c4.-> d8---3
\override Script #'font-size = #2
c4.-> d8---3
\override Stem #'font-size = #-5
c4.-> d8---3
```



Font size changes are achieved by scaling the design size that is closest to the desired size. The standard font size (for `font-size = #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

`\teeny`, `\tiny`, `\small`, `\normalsize`, `\large`, `\huge`.

See also

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Internals Reference: [Section “font-interface” in *Internals Reference*](#).

Fingering instructions

Fingering instructions can be entered using *note-digit*:

```
c4-1 d-2 f-4 e-3
```



Markup texts may be used for finger changes.

```
c4-1 d-2 f-4 c^\markup { \finger "2 - 3" }
```



A thumb-script can be added (e.g., in cello music) to indicate that a note should be played with the thumb.

```
<a_\thumb a'-3>2 <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>
```



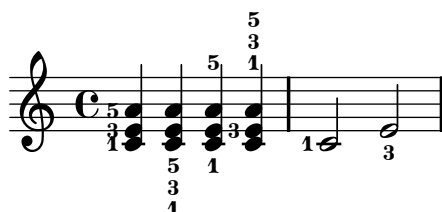
Fingering instructions may be manually placed above or below the staff, see [Section 5.4.2 \[Direction and placement\]](#), page 419.

Selected Snippets

Controlling the placement of chord fingerings

The placement of fingering numbers can be controlled precisely. For fingering orientation to apply, you must use a chord construct `<>` even if it is a single note.

```
\relative c' {
  \set fingeringOrientations = #'(left)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(down)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(down right up)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(up)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(left)
  <c-1>2
  \set fingeringOrientations = #'(down)
  <e-3>2
}
```



Allowing fingerings to be printed inside the staff

By default, vertically oriented fingerings are positioned outside the staff. However, this behavior can be canceled. Note: you must use a chord construct `<>`, even if it is only a single note.

```
\relative c' {
  <c-1 e-2 g-3 b-5>2
  \override Fingering #'staff-padding = #'()
  <c-1 e-2 g-3 b-5>4 <g'-0>
}
```



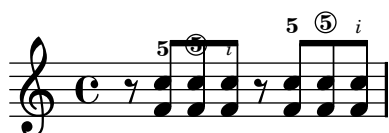
Avoiding collisions with chord fingerings

Fingerings and string numbers applied to individual notes will automatically avoid beams and stems, but this is not true by default for fingerings and string numbers applied to the individual notes of chords. The following example shows how this default behavior can be overridden.

```
\relative c' {
  \set fingeringOrientations = #'(up)
  \set stringNumberOrientations = #'(up)
  \set strokeFingerOrientations = #'(up)

  % Default behavior
  r8
  <f c'-5>8
  <f c'\5>8
  <f c'-\rightHandFinger #2 >8

  % Corrected to avoid collisions
  r8
  \override Fingering #'add-stem-support = ##t
  <f c'-5>8
  \override StringNumber #'add-stem-support = ##t
  <f c'\5>8
  \override StrokeFinger #'add-stem-support = ##t
  <f c'-\rightHandFinger #2 >8
}
```



See also

Notation Reference: [Section 5.4.2 \[Direction and placement\]](#), page 419

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Internals Reference: [Section “FingeringEvent” in *Internals Reference*](#), [Section “fingering-event” in *Internals Reference*](#), [Section “Fingering-engraver” in *Internals Reference*](#), [Section “New_fingering-engraver” in *Internals Reference*](#), [Section “Fingering” in *Internals Reference*](#).

Hidden notes

Hidden (or invisible or transparent) notes can be useful in preparing theory or composition exercises.

```
c4 d
\hideNotes
e4 f
\unHideNotes
g a
\hideNotes
b
\unHideNotes
c
```



Notation objects which are attached to invisible notes are still visible.

```
c4( d)
\hideNotes
e4(\p f)--
```



Predefined commands

`\hideNotes`, `\unHideNotes`.

See also

Learning Manual: [Section “Visibility and color of objects”](#) in *Learning Manual*.

Notation Reference: [\[Invisible rests\]](#), page 47, [Section 5.4.7 \[Visibility of objects\]](#), page 426, [\[Hiding staves\]](#), page 148.

Snippets: [Section “Editorial annotations”](#) in *Snippets*.

Internals Reference: [Section “Note-spacing-engraver”](#) in *Internals Reference*, [Section “NoteSpacing”](#) in *Internals Reference*.

Coloring objects

Individual objects may be assigned colors. Valid color names are listed in the [Section A.5 \[List of colors\]](#), page 455.

```
\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; this can be a symbol in the form `'FooBar` or a string in the form `"FooBar"`. 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.

If `x11-color` cannot make sense of the parameter then the color returned defaults to black.

```
\override Staff.StaffSymbol #'color = #(x11-color 'SlateBlue2)
\set Staff.instrumentName = \markup {
  \with-color #(x11-color 'navy) "Clarinet"
}
```

```
gis8 a
\override Beam #'color = #(x11-color "medium turquoise")
gis a
\override Accidental #'color = #(x11-color 'DarkRed)
gis a
\override NoteHead #'color = #(x11-color "LimeGreen")
gis a
% this is deliberate nonsense; note that the stems remain black
\override Stem #'color = #(x11-color 'Boggle)
b2 cis
```



Exact RGB colors can be specified using the Scheme function `rgb-color`.

```
\override Staff.StaffSymbol #'color = #(x11-color 'SlateBlue2)
\set Staff.instrumentName = \markup {
  \with-color #(x11-color 'navy) "Clarinet"
}
```

```
\override Stem #'color = #(rgb-color 0 0 0)
gis8 a
\override Stem #'color = #(rgb-color 1 1 1)
gis8 a
\override Stem #'color = #(rgb-color 0 0 0.5)
gis4 a
```



See also

Notation Reference: [Section A.5 \[List of colors\]](#), page 455, [Section 5.3.4 \[The tweak command\]](#), page 416.

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Known issues and warnings

An X11 color is not necessarily exactly the same shade as a similarly named normal color.

Not all X11 colors are distinguishable in a web browser, i.e., a web browser might not display a difference between 'LimeGreen and 'ForestGreen. For web use normal colors are recommended (i.e., #blue, #green, #red).

Notes in a chord cannot be colored with `\override`; use `\tweak` instead, see [Section 5.3.4 \[The `tweak` command\]](#), page 416.

Parentheses

Objects may be parenthesized by prefixing `\parenthesize` to the music event. When prefixed to a chord, it parenthesizes every note. Individual notes inside a chord may also be parenthesized.

```
c2 \parenthesize d
c2 \parenthesize <c e g>
c2 <c \parenthesize e g>
```



Non-note objects may be parenthesized as well. For articulations, a hyphen is needed before the `\parenthesize` command.

```
c2-\parenthesize -. d
c2 \parenthesize r
```



See also

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Internals Reference: [Section “Parenthesis_engraver” in *Internals Reference*](#), [Section “ParenthesesItem” in *Internals Reference*](#), [Section “parentheses-interface” in *Internals Reference*](#).

Known issues and warnings

Parenthesizing a chord prints parentheses around each individual note, instead of a single large parenthesis around the entire chord.

Stems

Whenever a note is found, a `Stem` object is created automatically. For whole notes and rests, they are also created but made invisible.

Stems may be manually placed to point up or down; see [Section 5.4.2 \[Direction and placement\]](#), page 419.

Predefined commands

`\stemUp`, `\stemDown`, `\stemNeutral`.

Selected Snippets

Default direction of stems on the center line of the staff

The default direction of stems on the center line of the staff is set by the `Stem` property `neutral-direction`.

```
\relative c' {
  a4 b c b
  \override Stem #'neutral-direction = #up
  a4 b c b
  \override Stem #'neutral-direction = #down
  a4 b c b
}
```



See also

Notation Reference: [Section 5.4.2 \[Direction and placement\]](#), page 419.

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Internals Reference: [Section “Stem_engraver” in *Internals Reference*](#), [Section “Stem” in *Internals Reference*](#), [Section “stem-interface” in *Internals Reference*](#).

1.7.2 Outside the staff

This section discusses how to add emphasis to elements in the staff from outside of the staff.

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.

```
\new Voice \with { \consists "Balloon_engraver" }
{
  \balloonGrobText #'Stem #'(3 . 4) \markup { "I'm a Stem" }
  a8
  \balloonGrobText #'Rest #'(-4 . -4) \markup { "I'm a rest" }
  r
  <c, g'-\balloonText #'(-2 . -2) \markup { "I'm a note head" } c>2.
}
```



There are two music functions, `balloonGrobText` and `balloonText`; the former is used like `\once \override` to attach text to any grob, and the latter is used like `\tweak`, typically within chords, to attach text to an individual note.

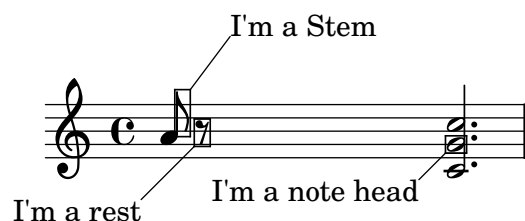
Balloon text normally influences note spacing, but this can be altered:

```
\new Voice \with { \consists "Balloon_engraver" }
{
  \balloonLengthOff
  \balloonGrobText #'Stem #'(3 . 4) \markup { "I'm a Stem" }
  a8
}
```

```

\balloonGrobText #'Rest #'(-4 . -4) \markup { "I'm a rest" }
r
\balloonLengthOn
<c, g'-\balloonText #'(-2 . -2) \markup { "I'm a note head" } c>2.
}

```



Predefined commands

`\balloonLengthOn`, `\balloonLengthOff`.

See also

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Internals Reference: [Section “Balloon-engraver” in *Internals Reference*](#), [Section “Balloon-TextItem” in *Internals Reference*](#), [Section “balloon-interface” in *Internals Reference*](#).

Grid lines

Vertical lines can be drawn between staves synchronized with the notes.

The `Grid_point_engraver` must be used to create the end points of the lines, while the `Grid_line_span_engraver` must be used to actually draw the lines. By default this centers grid lines horizontally below and to the left side of each note head. Grid lines extend from the middle lines of each staff. The `gridInterval` must specify the duration between the grid lines.

```

\layout {
  \context {
    \Staff
    \consists "Grid_point_engraver"
    gridInterval = #(ly:make-moment 1 4)
  }
  \context {
    \Score
    \consists "Grid_line_span_engraver"
  }
}

\score {
  \new ChoirStaff <<
    \new Staff \relative c'' {
      \stemUp
      c4. d8 e8 f g4
    }
    \new Staff \relative c {
      \clef bass
      \stemDown
      c4 g' f e
    }
  }
}

```

```
>>
}
```



Selected Snippets

Grid lines: changing their appearance

The appearance of grid lines can be changed by overriding some of their properties.

```
\score {
  \new ChoirStaff <<
    \new Staff {
      \relative c'' {
        \stemUp
        c'4. d8 e8 f g4
      }
    }
  \new Staff {
    \relative c {
      % this moves them up one staff space from the default position
      \override Score.GridLine #'extra-offset = #'(0.0 . 1.0)
      \stemDown
      \clef bass
      \once \override Score.GridLine #'thickness = #5.0
      c4
      \once \override Score.GridLine #'thickness = #1.0
      g'4
      \once \override Score.GridLine #'thickness = #3.0
      f4
      \once \override Score.GridLine #'thickness = #5.0
      e4
    }
  }
}
>>
\layout {
  \context {
    \Staff
    % set up grids
    \consists "Grid_point_engraver"
    % set the grid interval to one quarter note
    gridInterval = #(ly:make-moment 1 4)
  }
  \context {
    \Score
    \consists "Grid_line_span_engraver"
    % this moves them to the right half a staff space
```

```

\override NoteColumn #'X-offset = #-0.5
}
}
}

```



See also

Snippets: [Section “Editorial annotations” in *Snippets*](#).

Internals Reference: [Section “Grid_line_span_engraver” in *Internals Reference*](#), [Section “Grid_point_engraver” in *Internals Reference*](#), [Section “GridLine” in *Internals Reference*](#), [Section “GridPoint” in *Internals Reference*](#), [Section “grid-line-interface” in *Internals Reference*](#), [Section “grid-point-interface” in *Internals Reference*](#).

Analysis brackets

Brackets are used in musical analysis to indicate structure in musical pieces. Simple horizontal brackets are supported.

```

\layout {
  \context {
    \Voice
    \consists "Horizontal_bracket_engraver"
  }
}
\relative c'' {
  c2\startGroup
  d\stopGroup
}

```



Analysis brackets may be nested.

```

\layout {
  \context {
    \Voice
    \consists "Horizontal_bracket_engraver"
  }
}
\relative c'' {
  c4\startGroup\startGroup
  d4\stopGroup
}

```

```
e4\startGroup
d4\stopGroup\stopGroup
}
```



See also

Snippets: [Section “Editorial annotations”](#) in *Snippets*.

Internals Reference: [Section “Horizontal_bracket_engraver”](#) in *Internals Reference*, [Section “HorizontalBracket”](#) in *Internals Reference*, [Section “horizontal-bracket-interface”](#) in *Internals Reference*, [Section “Staff”](#) in *Internals Reference*.

1.8 Text

This section explains how to include text (with various formatting) in music scores.

Some text elements that are not dealt with here are discussed in other specific sections: [Section 2.1 \[Vocal music\]](#), page 198, [Section 3.2 \[Titles and headers\]](#), page 332.

1.8.1 Writing text

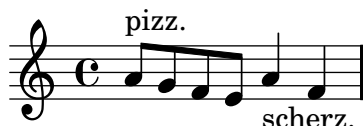
This section introduces different ways of adding text to a score.

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 3.3.3 \[Text encoding\]](#), page 344.

Text scripts

Simple “quoted text” indications may be added to a score, as demonstrated in the following example. Such indications may be manually placed above or below the staff, using the syntax described in [Section 5.4.2 \[Direction and placement\]](#), page 419.

```
a8~"pizz." g f e a4-"scherz." f
```



This syntax is actually a shorthand; more complex text formatting may be added to a note by explicitly using a `\markup` block, as described in [Section 1.8.2 \[Formatting text\]](#), page 182.

```
a8~\markup { \italic pizz. } g f e
a4_\markup { \tiny scherz. \bold molto } f
```



By default, text indications do not influence the note spacing. However, their widths can be taken into account: in the following example, the first text string does not affect spacing, whereas the second one does.

```
a8~"pizz." g f e
\textLengthOn
a4_"scherzando" f
```



In addition to text scripts, articulations can be attached to notes. For more information, see [\[Articulations and ornamentations\]](#), page 90.

For more information about the relative ordering of text scripts and articulations, see [Section “Placement of objects” in *Learning Manual*](#).

Predefined commands

```
\textLengthOn, \textLengthOff.
```

See also

Learning Manual: [Section “Placement of objects” in *Learning Manual*](#).

Notation Reference: [Section 1.8.2 \[Formatting text\]](#), page 182, [Section 5.4.2 \[Direction and placement\]](#), page 419.

Snippets: [Section “Text” in *Snippets*](#).

Internals Reference: [Section “TextScript” in *Internals Reference*](#).

Known issues and warnings

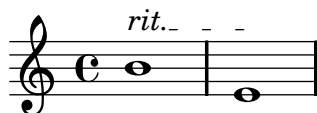
Checking to make sure that text scripts and lyrics are within the margins is a relatively large computational task. To speed up processing, LilyPond does not perform such calculations by default; to enable it, use

```
\override Score.PaperColumn #'keep-inside-line = ##t
```

Text spanners

Some performance indications, e.g., *rallentando* or *accelerando*, are written as text and are extended over multiple notes with dotted lines. Such objects, called “spanners”, may be created from one note to another using the following syntax:

```
\override TextSpanner #'(bound-details left text) = "rit."
b1\startTextSpan
e,\stopTextSpan
```



The string to be printed is set through object properties. By default it is printed in italic characters, but different formatting can be obtained using `\markup` blocks, as described in [Section 1.8.2 \[Formatting text\]](#), page 182.

```
\override TextSpanner #'(bound-details left text) =
  \markup { \upright "rit." }
b1\startTextSpan c
e,\stopTextSpan
```



The line style, as well as the text string, can be defined as an object property. This syntax is described in [Section 5.4.8 \[Line styles\]](#), page 431.

Predefined commands

```
\textSpannerUp, \textSpannerDown, \textSpannerNeutral.
```

Known issues and warnings

LilyPond is only able to handle one text spanner per voice.

See also

Notation Reference: [Section 5.4.8 \[Line styles\]](#), page 431, [\[Dynamics\]](#), page 93.

Snippets: [Section “Text” in Snippets](#).

Internals Reference: [Section “TextSpanner” in Internals Reference](#).

Text marks

Various text elements may be added to a score using the syntax described in [\[Rehearsal marks\]](#), page 81:

```
c4
\mark "Allegro"
c c c
```



This syntax makes it possible to put any text on a bar line; more complex text formatting may be added using a `\markup` block, as described in [Section 1.8.2 \[Formatting text\]](#), page 182:

```
<c e>1
\mark \markup { \italic { colla parte } }
<d f>2 <e g>
<c f aes>1
```



This syntax also allows to print special signs, like coda, segno or fermata, by specifying the appropriate symbol name as explained in [\[Music notation inside markup\]](#), page 192:

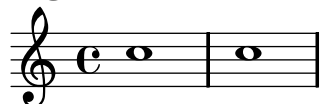
```
<bes f>2 <aes d>
\mark \markup { \musicglyph #"scripts.ufermata" }
<e g>1
```



Such objects are only typeset above the top staff of the score; depending on whether they are specified at the end or the middle of a bar, they can be placed above the bar line or between notes. When specified at a line break, the mark will be printed at the beginning of the next line.

```
\mark "Allegro"
c1 c
\mark "assai" \break
c c
```

Allegro



Selected Snippets

Printing marks at the end of a line

Marks can be printed at the end of the current line, instead of the beginning of the following line. In such cases, it might be preferable to align the right end of the mark with the bar line.

```
\relative c'' {
  g2 c
  d,2 a'
  \once \override Score.RehearsalMark #'break-visibility = #end-of-line-visible
  \once \override Score.RehearsalMark #'self-alignment-X = #RIGHT
  \mark "D.C. al Fine"
  \break
  g2 b,
```

```
c1 \bar "||"
}
```



Aligning marks with various notation objects

If specified, text marks may be aligned with notation objects other than bar lines. These objects include `ambitus`, `breathing-sign`, `clef`, `custos`, `staff-bar`, `left-edge`, `key-cancellation`, `key-signature`, and `time-signature`.

In such cases, text marks will be horizontally centered above the object. However this can be changed, as demonstrated on the second line of this example (in a score with multiple staves, this setting should be done for all the staves).

```
\relative c' {
  e1

  % the RehearsalMark will be centered above the Clef
  \override Score.RehearsalMark #'break-align-symbols = #'(clef)
  \key a \major
  \clef treble
  \mark ""
  e1

  % the RehearsalMark will be centered above the TimeSignature
  \override Score.RehearsalMark #'break-align-symbols = #'(time-signature)
  \key a \major
  \clef treble
  \time 3/4
  \mark \markup { \char ##x2193 }
  e2.

  % the RehearsalMark will be centered above the KeySignature
  \override Score.RehearsalMark #'break-align-symbols = #'(key-signature)
  \key a \major
  \clef treble
  \time 4/4
  \mark \markup { \char ##x2193 }
  e1

  \break
  e1

  % the RehearsalMark will be aligned with the left edge of the KeySignature
  \once \override Score.KeySignature #'break-align-anchor-alignment = #LEFT
```

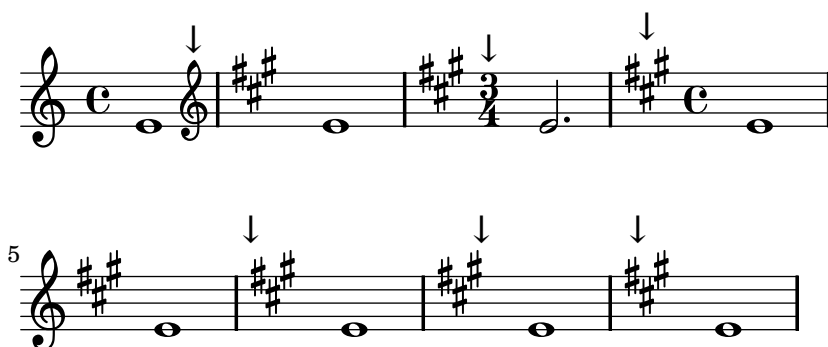
```

\mark \markup { \char ##x2193 }
\key a \major
e1

% the RehearsalMark will be aligned with the right edge of the KeySignature
\once \override Score.KeySignature #'break-align-anchor-alignment = #RIGHT
\key a \major
\mark \markup { \char ##x2193 }
e1

% the RehearsalMark will be aligned with the left edge of the KeySignature
% and then shifted right by one unit.
\once \override Score.KeySignature #'break-align-anchor = #1
\key a \major
\mark \markup { \char ##x2193 }
e1
}

```



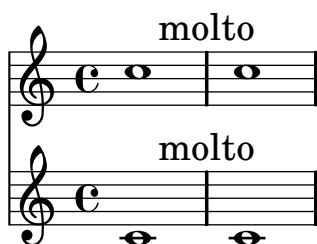
Printing marks on every staff

Although text marks are normally only printed above the topmost staff, they may also be printed on every staff.

```

\score {
  <<
    \new Staff { c''1 \mark "molto" c'' }
    \new Staff { c'1 \mark "molto" c' }
  >>
  \layout {
    \context {
      \Score
      \remove "Mark_engraver"
      \remove "Staff_collecting_engraver"
    }
    \context {
      \Staff
      \consists "Mark_engraver"
      \consists "Staff_collecting_engraver"
    }
  }
}

```



See also

Notation Reference: [Rehearsal marks], page 81, Section 1.8.2 [Formatting text], page 182, [Music notation inside markup], page 192, Section A.6 [The Feta font], page 457.

Snippets: Section “Text” in *Snippets*.

Internals Reference: Section “MarkEvent” in *Internals Reference*, Section “Mark_engraver” in *Internals Reference*, Section “RehearsalMark” in *Internals Reference*.

Separate text

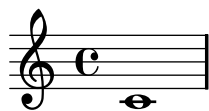
A `\markup` block can exist by itself, outside of any `\score` block, as a “top-level expression”. This syntax is described in Section 3.1.3 [File structure], page 330.

```
\markup {
  Tomorrow, and tomorrow, and tomorrow...
}
```

Tomorrow, and tomorrow, and tomorrow...

This allows printing text separately from the music, which is particularly useful when the input file contains several music pieces, as described in Section 3.1.2 [Multiple scores in a book], page 329.

```
\score {
  c'1
}
\markup {
  Tomorrow, and tomorrow, and tomorrow...
}
\score {
  c'1
}
```



Tomorrow, and tomorrow, and tomorrow...



Separate text blocks can be spread over multiple pages, making it possible to print text documents or books entirely within LilyPond. This feature, and the specific syntax it requires, are described in [Multi-page markup], page 194.

Predefined commands

`\markup`, `\markuplines`.

Selected Snippets

Stand-alone two-column markup

Stand-alone text may be arranged in several columns using `\markup` commands:

```
\markup {
  \fill-line {
    \hspace #1
    \column {
      \line { 0 sacrum convivium }
      \line { in quo Christus sumitur, }
      \line { recolitur memoria passionis ejus, }
      \line { mens impletur gratia, }
      \line { futurae gloriae nobis pignus datur. }
      \line { Amen. }
    }
    \hspace #2
    \column {
      \line { \italic { 0 sacred feast } }
      \line { \italic { in which Christ is received, } }
      \line { \italic { the memory of His Passion is renewed, } }
      \line { \italic { the mind is filled with grace, } }
      \line { \italic { and a pledge of future glory is given to us. } }
      \line { \italic { Amen. } }
    }
  }
  \hspace #1
}
```

O sacrum convivium	<i>O sacred feast</i>
in quo Christus sumitur,	<i>in which Christ is received,</i>
recolitur memoria passionis ejus,	<i>the memory of His Passion is renewed,</i>
mens impletur gratia,	<i>the mind is filled with grace,</i>
futurae gloriae nobis pignus datur.	<i>and a pledge of future glory is given to us.</i>
Amen.	<i>Amen.</i>

See also

Notation Reference: [Section 1.8.2 \[Formatting text\]](#), page 182, [Section 3.1.3 \[File structure\]](#), page 330, [Section 3.1.2 \[Multiple scores in a book\]](#), page 329, [\[Multi-page markup\]](#), page 194.

Snippets: [Section “Text” in *Snippets*](#).

Internals Reference: [Section “TextScript” in *Internals Reference*](#).

1.8.2 Formatting text

This section presents basic and advanced text formatting, using the `\markup` mode specific syntax.

Text markup introduction

A `\markup` block is used to typeset text with an extensible syntax called “markup mode”.

The markup syntax is similar to LilyPond’s usual syntax: a `\markup` expression is enclosed in curly braces `{ ... }`. A single word is regarded as a minimal expression, and therefore does not need to be enclosed with braces.

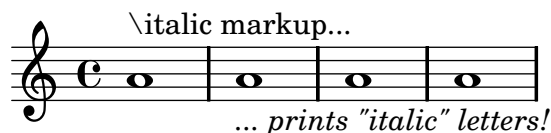
Unlike simple “quoted text” indications, `\markup` blocks may contain nested expressions or markup commands, entered using the backslash `\` character. Such commands only affect the first following expression.

```
a1-\markup intenso
a2^\markup { poco \italic più forte }
c e1
d2_\markup { \italic "string. assai" }
e
b1^\markup { \bold { molto \italic agitato } }
c
```



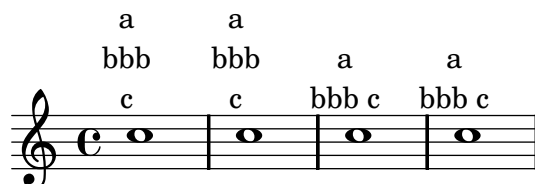
A `\markup` block may also contain quoted text strings. Such strings are treated as minimal text expressions, and therefore any markup command or special character (such as `\` and `#`) will be printed verbatim without affecting the formatting of the text. Double quotation marks themselves may be printed by preceding them with backslashes.

```
a1^\markup { \italic markup... }
a_\markup { \italic "... prints \"italic\" letters!" }
a a
```



To be treated as a distinct expression, a list of words needs to be enclosed with double quotes or preceded by a command. The way markup expressions are defined affects how these expressions will be stacked, centered and aligned; in the following example, the second `\markup` expression is treated the same as the first one:

```
c1^\markup { \center-column { a bbb c } }
c1^\markup { \center-column { a { bbb c } } }
c1^\markup { \center-column { a \line { bbb c } } }
c1^\markup { \center-column { a "bbb c" } }
```



Markups can be stored in variables. Such variables may be directly attached to notes:

```
allegro = \markup { \bold \large Allegro }
```

```
{
  d''8.\allegro
  d'16 d'4 r2
}
```



An exhaustive list of \markup-specific commands can be found in [Section A.8 \[Text markup commands\]](#), page 474.

See also

Notation Reference: [Section A.8 \[Text markup commands\]](#), page 474.

Snippets: [Section “Text” in Snippets](#).

Installed files: ‘scm/markup.scm’.

Known issues and warnings

Syntax errors for markup mode can be confusing.

Selecting font and font size

Basic font switching is supported in markup mode:

```
d1^\markup {
  \bold { Più mosso }
  \italic { non troppo \underline Vivo }
}
r2 r4 r8
d,_\markup { \italic quasi \smallCaps Tromba }
f1 d2 r
```

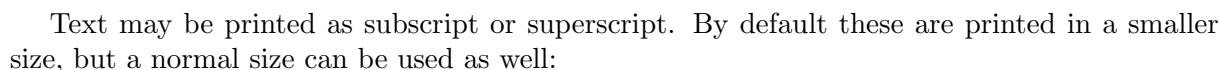


The size of the characters can also be altered in different ways:

- the font size can be set to predefined standard sizes,
- the font size can be set to an absolute value,
- the font size can also be changed relatively to its previous value.

The following example demonstrates these three methods:

```
f1_\markup {
  \tiny espressivo
  \large e
  \normalsize intenso
}
a^\markup {
  \fontsize #5 Sinfonia
  \fontsize #2 da
  \fontsize #3 camera
}
bes^\markup { (con
  \larger grande
  \smaller emozione
  \magnify #0.6 { e sentimento } )
}
d c2 r8 c bes a g1
```



1st movement
1st movement (part two)

```
\markup {
  \column {
    \line { Act \number 1 }
    \line { \sans { Scene I. } }
    \line { \typewriter { Verona. An open place. } }
    \line { Enter \roman Valentine and Proteus. }
  }
}
```

Some of these font families, used for specific items such as numbers or dynamics, do not provide all characters, as mentioned in [New dynamic marks], page 97 and [Manual repeat marks], page 114.

```
\markup {
  \column {
    \line {
      \concat { 1 \super st }
      movement
    }
    \line {
      \concat { \dynamic p , }
      \italic { con dolce espressione }
    }
  }
}
```

1st movement

***p**, con dolce espressione*

An exhaustive list of font switching commands and custom font usage commands can be found in [Section A.8.1 \[Font\]](#), page 474.

Defining custom font sets is also possible, as explained in [Section 1.8.3 \[Fonts\]](#), page 195.

Predefined commands

`\teeny`, `\tiny`, `\small`, `\normalsize`, `\large`, `\huge`, `\smaller`, `\larger`.

See also

Notation Reference: [Section A.8.1 \[Font\]](#), page 474, [\[New dynamic marks\]](#), page 97, [\[Manual repeat marks\]](#), page 114, [Section 1.8.3 \[Fonts\]](#), page 195.

Snippets: [Section “Text” in Snippets](#).

Internals Reference: [Section “TextScript” in Internals Reference](#).

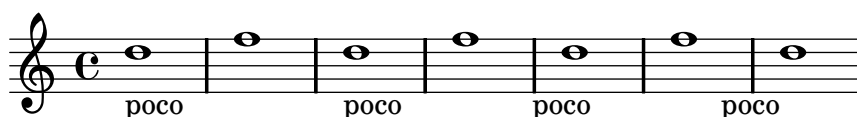
Installed files: ‘`scm/define-markup-commands.scm`’.

Text alignment

This subsection discusses how to place text in markup mode. Markup objects can also be moved as a whole, using the syntax described in [Section “Moving objects” in Learning Manual](#).

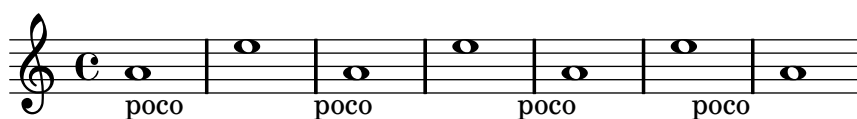
Markup objects may be aligned in different ways. By default, a text indication is aligned on its left edge: in the following example, there is no difference between the first and the second markup.

```
d1-\markup { poco }
f
d-\markup { \left-align poco }
f
d-\markup { \center-align { poco } }
f
d-\markup { \right-align poco }
```



Horizontal alignment may be fine-tuned using a numeric value:

```
a1-\markup { \halign #-1 poco }
e'
a,-\markup { \halign #0 poco }
e'
a,-\markup { \halign #0.5 poco }
e'
a,-\markup { \halign #2 poco }
```



Some objects may have alignment procedures of their own, and therefore are not affected by these commands. It is possible to move such markup objects as a whole, as shown for instance in [\[Text marks\]](#), page 177.

Vertical alignment is a bit more complex. As stated above, markup objects can be moved as a whole; however, it is also possible to move specific elements inside a markup block. In this case, the element to be moved needs to be preceded with an *anchor point*, that can be another markup element or an invisible object. The following example demonstrates these two possibilities; the last markup in this example has no anchor point, and therefore is not moved.

```
d2^\markup {
  Acte I
  \raise #2 { Scène 1 }
}
a'
g_\markup {
  \null
  \lower #4 \bold { Très modéré }
}
a
d,^\markup {
  \raise #4 \italic { Une forêt. }
}
a'4 a g2 a
```



Très modéré

Some commands can affect both the horizontal and vertical alignment of text objects in markup mode. Any object affected by these commands must be preceded with an anchor point:

```
d2^\markup {
  Acte I
  \translate #'(-1 . 2) "Scène 1"
}
a'
g_\markup {
  \null
  \general-align #Y #3.2 \bold "Très modéré"
}
a
d,^\markup {
  \null
  \translate-scaled #'(-1 . 2) \teeny "Une forêt."
}
a'4 a g2 a
```



Très modéré

A markup object may include several lines of text. In the following example, each element or expression is placed on its own line, either left-aligned or centered:

```
\markup {
  \column {
    a
    "b c"
    \line { d e f }
  }
  \hspace #10
  \center-column {
    a
    "b c"
    \line { d e f }
  }
}
```

a	a
b c	b c
d e f	d e f

Similarly, a list of elements or expressions may be spread to fill the entire horizontal line width (if there is only one element, it will be centered on the page). These expressions can, in turn, include multi-line text or any other markup expression:

```
\markup {
  \fill-line {
    \line { William S. Gilbert }
    \center-column {
      \huge \smallCaps "The Mikado"
      or
      \smallCaps "The Town of Titipu"
    }
    \line { Sir Arthur Sullivan }
  }
}
\markup {
  \fill-line { 1885 }
}
```

William S. Gilbert

THE MIKADO
or
THE TOWN OF TITIPU

Sir Arthur Sullivan

1885

Long text indications can also be automatically wrapped accordingly to the given line width. These will be either left-aligned or justified, as shown in the following example.

```
\markup {
  \column {
    \line \smallCaps { La vida breve }
    \line \bold { Acto I }
    \wordwrap \italic {
```

```

    (La escena representa el corral de una casa de
    gitanos en el Albaicín de Granada. Al fondo una
    puerta por la que se ve el negro interior de
    una Fragua, iluminado por los rojos resplandores
    del fuego.)
}
\hspace #0

\line \bold { Acto II }
\override #'(line-width . 50)
\justify \italic {
    (Calle de Granada. Fachada de la casa de Carmela
    y su hermano Manuel con grandes ventanas abiertas
    a través de las que se ve el patio
    donde se celebra una alegre fiesta)
}
}
}

```

LA VIDA BREVE

Acto I

(La escena representa el corral de una casa de gitanos en el Albaicín de Granada. Al fondo una puerta por la que se ve el negro interior de una Fragua, iluminado por los rojos resplandores del fuego.)

Acto II

(Calle de Granada. Fachada de la casa de Carmela y su hermano Manuel con grandes ventanas abiertas a través de las que se ve el patio donde se celebra una alegre fiesta)

An exhaustive list of text alignment commands can be found in [Section A.8.2 \[Align\]](#), [page 482](#).

See also

Learning Manual: [Section “Moving objects” in Learning Manual](#).

Notation Reference: [Section A.8.2 \[Align\]](#), [page 482](#), [\[Text marks\]](#), [page 177](#).

Snippets: [Section “Text” in Snippets](#).

Internals Reference: [Section “TextScript” in Internals Reference](#).

Installed files: ‘scm/define-markup-commands.scm’.

Graphic notation inside markup

Various graphic objects may be added to a score, using markup commands.

Some markup commands allow decoration of text elements with graphics, as demonstrated in the following example.

```

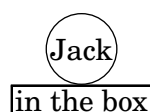
\markup \fill-line {
  \center-column {
    \circle Jack
    \box "in the box"
    \null
  }
}

```

```

\line {
  Erik Satie
  \hspace #3
  \bracket "1866 - 1925"
}
\null
\rounded-box \bold Prelude
}
}

```



Erik Satie [1866 - 1925]

Prelude

Some commands may require an increase in the padding around the text; this is achieved with some markup commands exhaustively described in [Section A.8.2 \[Align\]](#), page 482.

```

\markup \fill-line {
  \center-column {
    \box "Charles Ives (1874 - 1954)"
    \null
    \box \pad-markup #2 "THE UNANSWERED QUESTION"
    \box \pad-x #8 "A Cosmic Landscape"
    \null
  }
}
\markup \column {
  \line {
    \hspace #10
    \box \pad-to-box #'(-5 . 20) #'(0 . 5)
    \bold "Largo to Presto"
  }
  \pad-around #3
  "String quartet keeps very even time,
  Flute quartet keeps very uneven time."
}

```

[Charles Ives (1874 - 1954)]

THE UNANSWERED QUESTION

A Cosmic Landscape

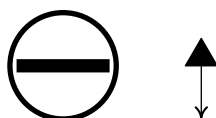
Largo to Presto

String quartet keeps very even time, Flute quartet keeps very uneven time.

Other graphic elements or symbols may be printed without requiring any text. As with any markup expression, such objects can be combined.

```
\markup {
  \combine
    \draw-circle #4 #0.4 ##f
    \filled-box #'(-4 . 4) #'(-0.5 . 0.5) #1
  \hspace #5

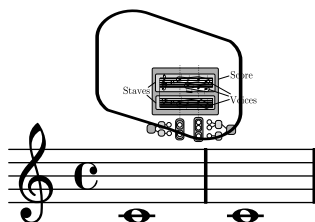
  \center-column {
    \triangle ##t
    \combine
      \draw-line #'(0 . 4)
      \arrow-head #Y #DOWN ##f
  }
}
```



Advanced graphic features include the ability to include external image files converted to the Encapsulated PostScript format (*eps*), or to directly embed graphics into the input file, using native PostScript code. In such a case, it may be useful to explicitly specify the size of the drawing, as demonstrated below:

```
c1^\markup {
  \combine
    \epsfile #X #10 #"./context-example.eps"
    \with-dimensions #'(0 . 6) #'(0 . 10)
    \postscript #"
      -2 3 translate
      2.7 2 scale
      newpath
      2 -1 moveto
      4 -2 4 1 1 arct
      4 2 3 3 1 arct
      0 4 0 3 1 arct
      0 0 1 -1 1 arct
      closepath
      stroke"
}
```

c



An exhaustive list of graphics-specific commands can be found in [Section A.8.3 \[Graphic\]](#), [page 496](#).

See also

Notation Reference: [Section A.8.3 \[Graphic\]](#), [page 496](#), [Section 1.7 \[Editorial annotations\]](#), [page 164](#).

Snippets: [Section “Text” in *Snippets*](#).

Internals Reference: [Section “TextScript” in *Internals Reference*](#).

Installed files: ‘scm/define-markup-commands.scm’, ‘scm/stencil.scm’.

Music notation inside markup

Various musical notation elements may be added to a score, inside a markup object.

Notes and accidentals can be entered using markup commands:

```
a2 a^\markup {
  \note #"4" #1
  =
  \note-by-number #1 #1 #1.5
}
b1_\markup {
  \natural \semiflat \flat
  \sesquiflat \doubleflat
}
\glissando
a1_\markup {
  \natural \semisharp \sharp
  \sesquisharp \doublesharp
}
\glissando b
```



Other notation objects may also be printed in markup mode:

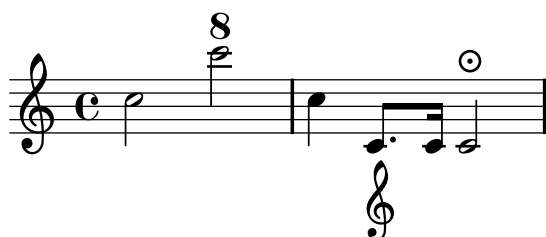
```
g1 bes
ees-\markup {
  \finger 4
  \tied-lyric #"~"
  \finger 1
}
fis_\markup { \dynamic rf }
bes^\markup {
  \beam #8 #0.1 #0.5
}
cis
d-\markup {
  \markalphabet #8
  \markletter #8
}
```

}



More generally, any available musical symbol may be included separately in a markup object, as demonstrated below; an exhaustive list of these symbols and their names can be found in [Section A.6 \[The Feta font\]](#), page 457.

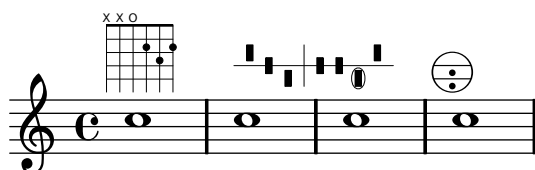
```
c2
c'\markup { \musicglyph #"eight" }
c,4
c,8.\markup { \musicglyph #"clefs.G_change" }
c16
c2^\markup { \musicglyph #"timesig.neomensural94" }
```



Another way of printing non-text glyphs is described in [\[Fonts explained\]](#), page 195. This is useful for printing braces of various sizes.

The markup mode also supports diagrams for specific instruments:

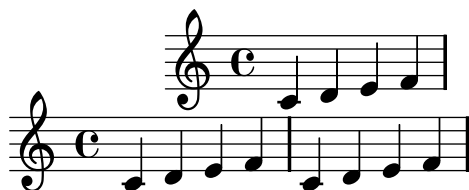
```
c1^\markup {
  \fret-diagram-terse #"x;x;o;2;3;2;"
}
c^\markup {
  \harp-pedal #"^~v|---ov^"
}
c
c^\markup {
  \combine
  \musicglyph #"accordion.accDiscant"
  \combine
  \raise #0.5 \musicglyph #"accordion.accDot"
  \raise #1.5 \musicglyph #"accordion.accDot"
}
```



Such diagrams are documented in [Section A.8.5 \[Instrument Specific Markup\]](#), page 504.

A whole score can even be nested inside a markup object. In such a case, the nested `\score` block must contain a `\layout` block, as demonstrated here:

```
c4 d~\markup {
  \score {
    \relative c' { c4 d e f }
    \layout { }
  }
}
e f |
c d e f
```



An exhaustive list of music notation related commands can be found in [Section A.8.4 \[Music\]](#), [page 501](#).

See also

Notation Reference: [Section A.8.4 \[Music\]](#), [page 501](#), [Section A.6 \[The Feta font\]](#), [page 457](#), [\[Fonts explained\]](#), [page 195](#).

Snippets: [Section “Text” in *Snippets*](#).

Internals Reference: [Section “TextScript” in *Internals Reference*](#).

Installed files: ‘scm/define-markup-commands.scm’, ‘scm/fret-diagrams.scm’, ‘scm/harp-pedals.scm’.

Multi-page markup

Although standard markup objects are not breakable, a specific syntax makes it possible to enter lines of text that can spread over multiple pages:

```
\markuplines {
  \justified-lines {
    A very long text of justified lines.
    ...
  }
  \wordwrap-lines {
    Another very long paragraph.
    ...
  }
  ...
}
```

A very long text of justified lines. ...

Another very long paragraph. ...

...

This syntax accepts a list of markups, that can be

- the result of a markup list command,

- a list of markups,
- a list of markup lists.

An exhaustive list of markup list commands can be found in [Section A.9 \[Text markup list commands\]](#), page 512.

See also

Notation Reference: [Section A.9 \[Text markup list commands\]](#), page 512,

Snippets: [Section “Text” in *Snippets*](#).

Extending: [Section “New markup list command definition” in *Extending*](#).

Internals Reference: [Section “TextScript” in *Internals Reference*](#).

Installed files: ‘scm/define-markup-commands.scm’.

Predefined commands

`\markuplines`.

1.8.3 Fonts

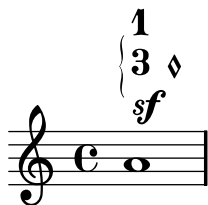
This section presents the way fonts are handled, and how they may be changed in scores.

Fonts explained

Fonts are handled through several libraries. FontConfig is used to detect available fonts on the system; the selected fonts are rendered using Pango.

Music notation fonts can be described as a set of specific glyphs, ordered in several families. The following syntax allows various LilyPond **feta** non-text fonts to be used directly in markup mode:

```
a1^\markup {
  \vcenter {
    \override #'(font-encoding . fetaBraces)
    \lookup #"brace120"
    \override #'(font-encoding . fetaText)
    \column { 1 3 sf }
    \override #'(font-encoding . fetaMusic)
    \lookup #"noteheads.s0petrucci"
  }
}
```



However, all these glyphs except the braces of various sizes contained in **fetaBraces** are available using the simpler syntax described in [\[Music notation inside markup\]](#), page 192.

When using the glyphs contained in **fetaBraces**, the size of the brace is specified by the numerical part of the glyph name, in arbitrary units. Any integer from 0 to 575 inclusive may be specified, 0 giving the smallest brace. The optimum value must be determined by trial and error. These glyphs are all left braces; right braces may be obtained by rotation, see [Section 5.4.9 \[Rotating objects\]](#), page 432.

Three families of text fonts are made available: the *roman* (serif) font, that defaults to New Century Schoolbook, the *sans* font and the monospaced *typewriter* font – these last two families are determined by the Pango installation.

Each family may include different shapes and series. The following example demonstrates the ability to select alternate families, shapes, series and sizes. The value supplied to **font-size** is the required change from the default size.

```
\override Score.RehearsalMark #'font-family = #'typewriter
\mark \markup "Ouverture"
\override Voice.TextScript #'font-shape = #'italic
\override Voice.TextScript #'font-series = #'bold
d2.^ \markup "Allegro"
\override Voice.TextScript #'font-size = #-3
c4^smaller
```



A similar syntax may be used in markup mode; however in this case it is preferable to use the simpler syntax explained in [\[Selecting font and font size\]](#), page 184:

```
\markup {
  \column {
    \line {
      \override #'(font-shape . italic)
      \override #'(font-size . 4)
      Idomeneo,
    }
    \line {
      \override #'(font-family . typewriter)
      {
        \override #'(font-series . bold)
        re
        di
      }
      \override #'(font-family . sans)
      Creta
    }
  }
}
```

Idomeneo,
re di Creta

Although it is easy to switch between preconfigured fonts, it is also possible to use other fonts, as explained in the following sections: [\[Single entry fonts\]](#), page 197 and [\[Entire document fonts\]](#), page 197.

See also

Notation Reference: [Section A.6 \[The Feta font\]](#), page 457, [\[Music notation inside markup\]](#), page 192, [\[Selecting font and font size\]](#), page 184, [Section A.8.1 \[Font\]](#), page 474.

Single entry fonts

Any font that is installed on the operating system and recognized by FontConfig may be used in a score, using the following syntax:

```
\override Staff.TimeSignature #'font-name = #"Bitstream Charter"
\override Staff.TimeSignature #'font-size = #2
\time 3/4
```

```
a1_\markup {
  \override #'(font-name . "Vera Bold")
  { Vera Bold }
}
```



The following command displays a list of all available fonts on the operating system:

```
lilypond -dshow-available-fonts x
```

See also

Notation Reference: [\[Fonts explained\]](#), page 195, [\[Entire document fonts\]](#), page 197.

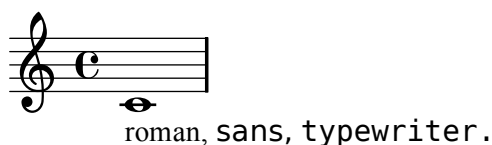
Snippets: [Section “Text” in *Snippets*](#).

Entire document fonts

It is possible to change the fonts to be used as the default fonts in the *roman*, *sans* and *typewriter* font families by specifying them, in that order, as shown in the example below. For an explanation of fonts, see [\[Fonts explained\]](#), page 195.

```
\paper {
  myStaffSize = #20
  #(define fonts
    (make-pango-font-tree "Times New Roman"
                          "Nimbus Sans"
                          "Luxi Mono"
                          (/ myStaffSize 20)))
}
```

```
\relative c'{
  c1-\markup {
    roman,
    \sans sans,
    \typewriter typewriter. }
}
```



See also

Notation Reference: [\[Fonts explained\]](#), page 195, [\[Single entry fonts\]](#), page 197, [\[Selecting font and font size\]](#), page 184, [Section A.8.1 \[Font\]](#), page 474.

2 Specialist notation

This chapter explains how to create musical notation for specific types of instrument or in specific styles.

2.1 Vocal music

This section explains how to typeset vocal music, and make sure that the lyrics will be aligned with the notes of their melody.

2.1.1 Common notation for vocal music

This section discusses issues related to vocal music in general, and to some particular styles of vocal music.

References for vocal music and lyrics

Various issues may arise when engraving vocal music. Some of these are discussed in this section, while others are explained elsewhere:

- Most styles of vocal music use written text as lyrics. An introduction to this notation is to be found in [Section “Setting simple songs” in *Learning Manual*](#).
- Vocal music is likely to require the use of `markup` mode, either for lyrics or for other text elements (character’s names, etc.). This syntax is described in [\[Text markup introduction\]](#), [page 182](#).
- Lead sheets may be printed by combining vocal parts and ‘chord mode’; this syntax is explained in [Section 2.7 \[Chord notation\]](#), [page 278](#).
- ‘Ambitus’ may be added at the beginning of vocal staves, as explained in [\[Ambitus\]](#), [page 27](#).
- Vocal parts may be printed using traditional clefs, as shown in [\[Clef\]](#), [page 13](#).
- Ancient vocal music is supported, as explained in [Section 2.9 \[Ancient notation\]](#), [page 298](#).

Opera

TBC

Song books

TBC

Selected Snippets

Simple lead sheet

When put together, chord names, a melody, and lyrics 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

Notation Reference: [Section 2.7 \[Chord notation\]](#), page 278.

Spoken music

Such effects as ‘parlato’ or ‘Sprechgesang’ require performers to speak without pitch but still with rhythm; these are notated by cross note heads, as demonstrated in [\[Special note heads\]](#), page 29.

Chants

TBC

Ancient vocal music

TBC

See also

Notation Reference: [Section 2.9 \[Ancient notation\]](#), page 298.

2.1.2 Entering lyrics

Lyrics explained

Since LilyPond input files are text, there is at least one issue to consider when working with vocal music: song texts must be interpreted as text, not notes. For example, the input `d` should be interpreted as a one letter syllable, not the note D. Therefore, a special lyric mode has to be used, either explicitly or using some abbreviated methods.

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

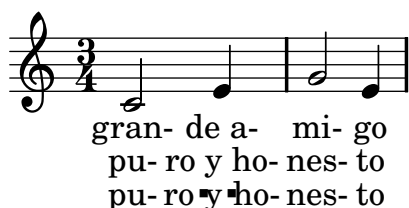
```
\override Score.LyricText #'font-shape = #'italic
```

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 c' { 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; therefore a font that includes this glyph (such as DejaVuLGC) has to be used. More explanations about text and non-text fonts can be found in [Section 1.8.3 \[Fonts\]](#), page 195.

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 3.3.3 \[Text encoding\]](#), page 344, for more info.

```
\relative c' { e4 f e d e f e2 }
\addlyrics { He said: \Let my peo ple go". }
```



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

Notation Reference: [Section 1.8.3 \[Fonts\]](#), page 195.

Internals Reference: [Section “LyricText”](#) in *Internals Reference*, [Section “LyricSpace”](#) in *Internals Reference*.

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 c' { c2 e4 g2. }
\addlyrics { play the game }

```



More stanzas can be added by adding more `\addlyrics` sections

```

\time 3/4
\relative c' { 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 [\[Lyrics explained\]](#), page 199.

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 set up 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: [Section “LyricCombineMusic”](#) in *Internals Reference*, [Section “Lyrics”](#) in *Internals Reference*.

2.1.3 Aligning lyrics to a melody

Aligning of text with melodies can be made automatically, but if you specify the durations of the syllables it can also be made manually. Lyrics aligning and typesetting are prepared with the help of skips, hyphens and extender lines.

Lyrics are printed by interpreting them in the context called **Lyrics**; see [Section “Lyrics”](#) in *Internals Reference*, for more.

```

\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`

The **Voice** context containing the melody to which the lyrics are being aligned must not have “died”, or the lyrics after that point will be lost. This can happen if there are periods when that voice has nothing to do. For methods of keeping contexts alive, see [Section 5.1.3 \[Keeping contexts alive\]](#), page 401.

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` key-word may be omitted.

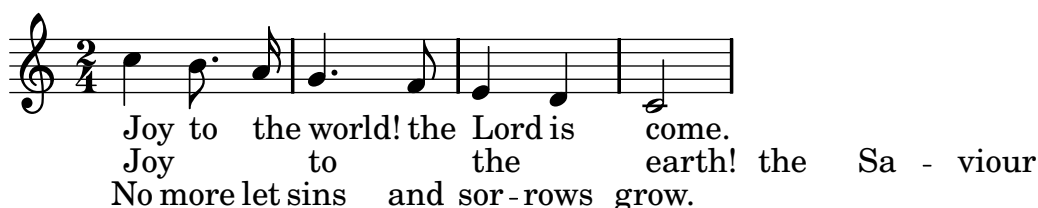
The following example uses different commands for entering lyrics.

```
<<
\new Voice = "one" \relative c'' {
  \autoBeamOff
  \time 2/4
  c4 b8. a16 g4. f8 e4 d c2
}

% not recommended: left-aligned syllables
\new Lyrics \lyricmode { Joy4 to8. the16 world!4. the8 Lord4 is come.2 }

% wrong: durations needed
\new Lyrics \lyricmode { Joy to the earth! the Sa -- viour reigns. }

%correct
\new Lyrics \lyricsto "one" { No more let sins and sor -- rows grow. }
>>
```



⁸reigns.

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 }
```

Manual syllable durations

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

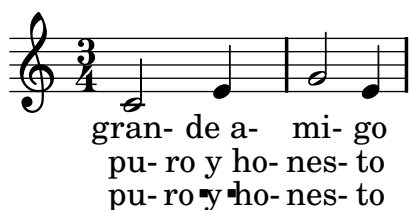
Notation Reference: [Section 5.1.3 \[Keeping contexts alive\]](#), page 401.

Internals Reference: [Section “Lyrics” in *Internals Reference*](#), [Section “Voice” in *Internals Reference*](#).

Multiple syllables to one 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 c' { c2 e4 g2 e4 }
\addlyrics { gran- de_a- mi- go }
\addlyrics { pu- "ro y ho-" nes- to }
\addlyrics { pu- ro~y~ho- nes- to }
```



See also

Internals Reference: [Section “LyricCombineMusic” in *Internals Reference*](#).

Multiple notes to one 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 extra note that has to be added to the melisma.

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.

Additionally, you can make an extender line to be typeset to indicate the melisma in the score, writing a double underscore next to the first syllable of the melisma. This example shows the three elements that are used for this purpose (all of them surrounded by spaces): double hyphens to separate syllables in a word, underscores to add notes to a melisma, and a double underscore to put an extender line.

```
{ \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 [\[Setting automatic beam behavior\]](#), [page 64](#)) is switched off.

```
<<
\new Voice = "lala" {
  \time 3/4
  \autoBeamOff
  f4 g8[ f e f]
  e2
}
\new Lyrics \lyricsto "lala" {
  la di _ _ daah
}
```

```
}
>>
```



A complete example of a SATB score setup is in [Section “Vocal ensembles”](#) in *Learning Manual*.

Predefined commands

`\melisma`, `\melismaEnd`.

Known issues and warnings

Melismata are not detected automatically, and extender lines must be inserted by hand.

Skipping notes

Making a lyric line run slower than the melody can be achieved by inserting `\skips` into the lyrics. For every `\skip`, the text will be delayed another note. The `\skip` command must be followed by a valid duration, but this is ignored when `\skip` is used in lyrics.

```
\relative c' { c c g' }
\addlyrics {
  twin -- \skip 4
  kle
}
```



Extenders and hyphens

In the last syllable of a word, melismata are sometimes indicated with a long horizontal line starting in the melisma syllable, and ending in 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).

Note: Melismata are indicated in the score with extender lines, which are entered as one double underscore; but short melismata can also be entered by skipping individual notes, which are entered as single underscore characters; these do not make an extender line to be typeset by default.

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 tightly 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: [Section “LyricExtender”](#) in *Internals Reference*, [Section “LyricHyphen”](#) in *Internals Reference*.

Lyrics and repeats

TBC

2.1.4 Specific uses of lyrics

Often, different stanzas of one song are put to one melody in slightly differing ways. Such variations can still be captured with `\lyricsto`.

Divisi lyrics

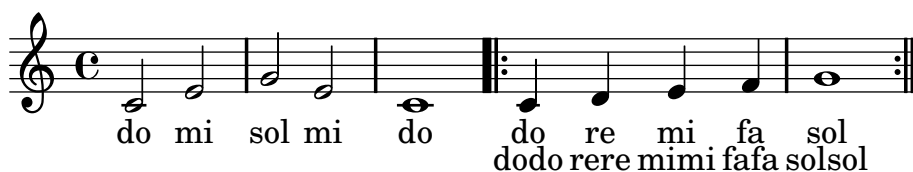
You can display alternate (or divisi) lyrics by naming voice contexts and attaching lyrics to those specific contexts.

```
\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 }
>>
}
```





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' }
>>
```



This method is recommended only if the music in the `Devnull` context does not contain melismata. Melismata are defined by the `Voice` context. Connecting lyrics to a `Devnull` context makes the voice/lyrics links to get lost, and so does the info on melismata. Therefore, if you link lyrics to a `Devnull` context, the implicit melismata get ignored.

Spacing out syllables

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
    }
  }
}
```



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
  }
}
```

Centering lyrics between staves

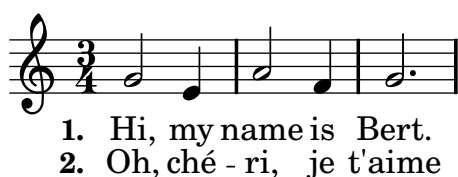
TBC

2.1.5 Stanzas

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, ché -- ri, je t'aime
}
```



These numbers are put just before the start of the first syllable.

Adding dynamics marks to stanzas

Stanzas differing in loudness may be indicated by putting a dynamics mark before each stanza. In LilyPond, everything 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
>>
```



Adding singers' names to stanzas

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, ché -- ri, je t'aime
}

```



Bert Hi, my name is Bert.
Ernie Oh, ché - ri, je t'aime

Stanzas with different rhythms


Ignoring melismata

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.

```

<<
\relative c' \new Voice = "lahlah" {
  \set Staff.autoBeaming = ##f
  c4
  \slurDotted
  f8.[( g16)]
  a4
}
\new Lyrics \lyricsto "lahlah" {
  more slow -- ly
}
\new Lyrics \lyricsto "lahlah" {
  go
  \set ignoreMelismata = ##t
  fas -- ter
  \unset ignoreMelismata
  still
}
>>

```



more slow - ly
go fas-ter still

Known issues and warnings

Unlike most `\set` commands, `\set ignoreMelismata` does not work if prefixed with `\once`. It is necessary to use `\set` and `\unset` to bracket the lyrics where melismata are to be ignored.

Adding syllables to grace notes

By default, grace notes (e.g. via `\grace`) do not get assigned syllables when using `\lyricsto`, but this behavior can be changed:

```
\relative c' {
  f4 \appoggiatura a32 b4
  \grace { f16[ a16] } b2
  \afterGrace b2 { f16[ a16] }
  \appoggiatura a32 b4
  \acciaccatura a8 b4
}
\addlyrics {
  normal
  \set includeGraceNotes = ##t
  case,
  gra -- ce case,
  after -- grace case,
  \set ignoreMelismata = ##t
  app. case,
  acc. case.
}
```



Known issues and warnings

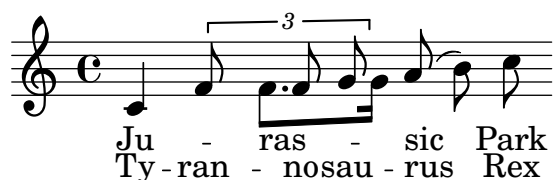
Like for `associatedVoice`, `includeGraceNotes` needs to be set at latest one syllable before the one which is to be put under a grace note. For the case of a grace note at the very beginning of a piece of music, consider using a `\with` or `\context` block:

```
<<
  \new Voice = melody \relative c' {
    \grace { c16[( d e f] }
    g1) f
  }
  \new Lyrics \with { includeGraceNotes = ##t }
  \lyricsto melody {
    Ah __ fa
  }
>>
```



Switching to an alternative melody

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.

This command must be one syllable too early, before 'Ty' in this case. In other words, changing the `associatedVoice` happens one step later than expected. This is for technical reasons, and it is not a bug.

```
\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`.

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. }
}
```

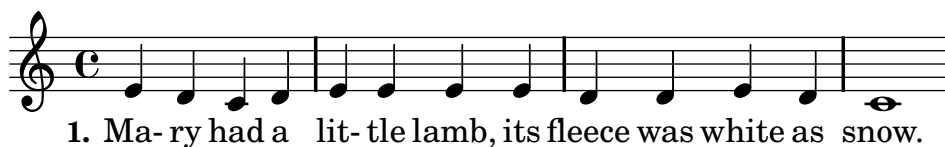
```

}
\markup{
  \wordwrap-string #"
  Verse 3.

  Mary took it home again,

  It was against the rule."
}

```



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.

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
}
}

```



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: [Section “LyricText”](#) in *Internals Reference*, [Section “StanzaNumber”](#) in *Internals Reference*.

2.2 Keyboard and other multi-staff instruments

The image displays two musical staves with various notation techniques. The top staff features a piano part with a treble and bass clef. It includes markings such as "Un peu retenu très expressif", "ppp", "Rall.", "long", "a Tempo", and "ped.". The bottom staff also features a piano part with a treble and bass clef. It includes markings such as "Rallentando", "Lent", "ppp", and "8va".

This section discusses several aspects of music notation that are unique to keyboard instruments and other instruments notated on many staves, such as harps and vibraphones. For the purposes of this section this entire group of multi-staff instruments is called “keyboards” for short, even though some of them do not have a keyboard.

2.2.1 Common notation for keyboards

This section discusses notation issues that may arise for most keyboard instruments.

References for keyboards

Keyboard instruments are usually notated with Piano staves. These are two or more normal staves coupled with a brace. The same notation is also used for other keyed instruments. Organ music is normally written with two staves inside a `PianoStaff` group and third, normal staff for the pedals.

The staves in keyboard music are largely independent, but sometimes voices can cross between the two staves. This section discusses notation techniques particular to keyboard music.

Several common issues in keyboard music are covered elsewhere:

- Keyboard music usually contains multiple voices and the number of voices may change regularly; this is described in [\[Collision resolution\]](#), page 126.
- Keyboard music can be written in parallel, as described in [\[Writing music in parallel\]](#), page 133.
- Fingerings are indicated with [\[Fingering instructions\]](#), page 165.
- Organ pedal indications are inserted as articulations, see [Section A.10 \[List of articulations\]](#), page 513.
- Vertical grid lines can be shown with [\[Grid lines\]](#), page 172.
- Keyboard music often contains *Laissez vibrer* ties as well as ties on arpeggios and tremolos, described in [\[Ties\]](#), page 42.

- Placing arpeggios across multiple voices and staves is covered in [Arpeggio], page 105.
- Tremolo marks are described in [Tremolo repeats], page 119.
- Several of the tweaks that can occur in keyboard music are demonstrated in Section “Real music example” in *Learning Manual*.
- Hidden notes can be used to produce ties that cross voices, as shown in Section “Other uses for tweaks” in *Learning Manual*.

See also

Learning Manual: Section “Real music example” in *Learning Manual*, Section “Other uses for tweaks” in *Learning Manual*.

Notation Reference: [Grouping staves], page 137, [Instrument names], page 154, [Collision resolution], page 126, [Writing music in parallel], page 133, [Fingering instructions], page 165, Section A.10 [List of articulations], page 513, [Grid lines], page 172, [Ties], page 42, [Arpeggio], page 105, [Tremolo repeats], page 119.

Internals Reference: Section “PianoStaff” in *Internals Reference*.

Snippets: Section “Keyboards” in *Snippets*.

Known issues and warnings

Dynamics are not automatically centered, but workarounds do exist. One option is the ‘piano centered dynamics’ template under Section “Piano templates” in *Learning Manual*; another option is to increase the staff-padding of dynamics as discussed in Section “Moving objects” in *Learning Manual*.

Changing staff manually

Voices can be switched between staves manually, using the command

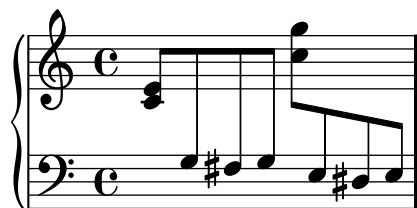
```
\change Staff = staffname
```

The string *staffname* is the name of the staff. It switches the current voice from its current staff to the staff called *staffname*. Typical values for *staffname* are "up" and "down", or "RH" and "LH".

The staff to which the voice is being switched must exist at the time of the switch. If necessary, staves should be “kept alive”, see Section 5.1.3 [Keeping contexts alive], page 401.

Cross-staff notes are beamed automatically:

```
\new PianoStaff <<
  \new Staff = "up" {
    <e' c'>8
    \change Staff = "down"
    g8 fis g
    \change Staff = "up"
    <g'' c''>8
    \change Staff = "down"
    e8 dis e
    \change Staff = "up"
  }
  \new Staff = "down" {
    \clef bass
    % keep staff alive
    s1
  }
>>
```



If the beaming needs to be tweaked, make any changes to the stem directions first. The beam positions are then measured from the center of the staff that is closest to the beam. For a simple example of beam tweaking, see [Section “Fixing overlapping notation” in *Learning Manual*](#).

See also

Learning Manual: [Section “Fixing overlapping notation” in *Learning Manual*](#).

Notation Reference: [\[Stems\]](#), page 170, [\[Automatic beams\]](#), page 63, [Section 5.1.3 \[Keeping contexts alive\]](#), page 401.

Snippets: [Section “Keyboards” in *Snippets*](#).

Internals Reference: [Section “Beam” in *Internals Reference*](#), [Section “ContextChange” in *Internals Reference*](#).

Changing staff automatically

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 staff group (usually a `PianoStaff`), called "up" and "down". The lower staff will be in the bass clef by default. The autochanger switches on the basis of the pitch (middle C is the turning point), and it looks ahead skipping over rests to switch in advance.

```
\new PianoStaff {
  \autochange {
    g4 a b c'
    d'4 r a g
  }
}
```



A `\relative` section that is outside of `\autochange` has no effect on the pitches of the music, so if necessary, put `\relative` inside `\autochange`.

If additional control is needed over the individual staves, they can be created manually with the names "up" and "down". The `\autochange` command will then switch its voice between the existing staves.

Note: If staves are created manually, they *must* be named "up" and "down".

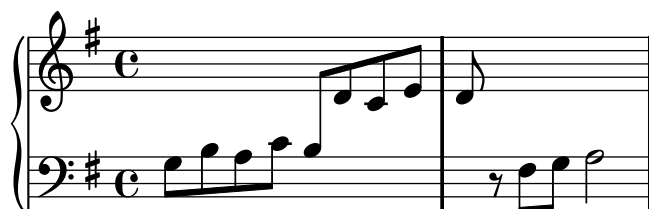
For example, staves must be created manually in order to place a key signature in the lower staff:

```
\new PianoStaff <<
  \new Staff = "up" {
```

```

\new Voice = "melOne" {
  \key g \major
  \autochange \relative c' {
    g8 b a c b d c e
    d8 r fis, g a2
  }
}
}
\new Staff = "down" {
  \key g \major
  \clef bass
}
>>

```



See also

Notation Reference: [\[Changing staff manually\]](#), page 217.

Snippets: [Section “Keyboards” in *Snippets*](#).

Internals Reference: [Section “AutoChangeMusic” in *Internals Reference*](#).

Known issues and warnings

The staff switches may not end up in optimal places. For high quality output, staff switches should be specified manually.

Chords will not be split across the staves; they will be assigned to a staff based on the first note named in the chord construct.

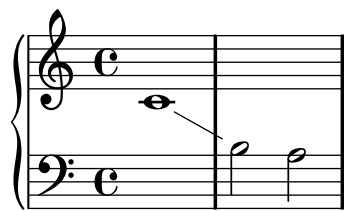
Staff-change lines

Whenever a voice switches to another staff, a line connecting the notes can be printed automatically:

```

\new PianoStaff <<
  \new Staff = "one" {
    \showStaffSwitch
    c1
    \change Staff = "two"
    b2 a
  }
  \new Staff = "two" {
    \clef bass
    s1*2
  }
}
>>

```



Predefined commands

`\showStaffSwitch`, `\hideStaffSwitch`.

See also

Snippets: [Section “Keyboards” in *Snippets*](#).

Internals Reference: [Section “Note_head_line_engraver” in *Internals Reference*](#), [Section “VoiceFollower” in *Internals Reference*](#).

Cross-staff stems

Chords that cross staves may be produced:

```
\new PianoStaff <<
  \new Staff {
    \relative c' {
      f8 e4 d8 d f e4
    }
  }
  \new Staff {
    \relative c' {
      << {
        \clef bass
        % stems may overlap the other staff
        \override Stem #'cross-staff = ##t
        % extend the stems to reach the other staff
        \override Stem #'length = #12
        % do not print extra flags
        \override Stem #'flag-style = #'no-flag
        % prevent beaming as needed
        a8 g4 f8 f bes\noBeam g4
      }
    }
  }
>>
```



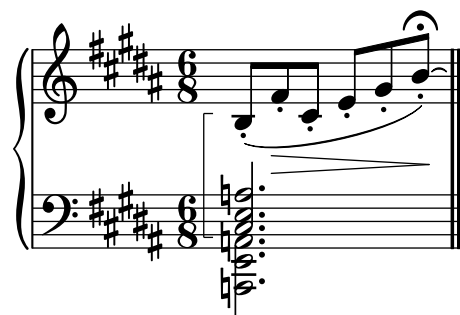
Selected Snippets

Indicating cross-staff chords with arpeggio bracket

An arpeggio bracket can indicate that notes on two different staves are to be played with the same hand. In order to do this, the `PianoStaff` must be set to accept cross-staff arpeggios and the arpeggios must be set to the bracket shape in the `PianoStaff` context.

(Debussy, *Les collines d'Anacapri*, m. 65)

```
\new PianoStaff <<
  \set PianoStaff.connectArpeggios = ##t
  \override PianoStaff.Arpeggio #'stencil = #ly:arpeggio::brew-chord-bracket
  \new Staff {
    \relative c' {
      \key b \major
      \time 6/8
      b8-.(\arpeggio fis'-.> cis-. e-. gis-. b-.)\!\fermata^\laissezVibrer
      \bar "||"
    }
  }
  \new Staff {
    \relative c' {
      \clef bass
      \key b \major
      <<
        {
          <a e cis>2.\arpeggio
        }
        \\
        {
          <a, e a,>2.
        }
      >>
    }
  }
>>
```



See also

Snippets: [Section “Keyboards” in *Snippets*](#).

Internals Reference: [Section “Stem” in *Internals Reference*](#).

2.2.2 Piano

This section discusses notation issues that relate most directly to the piano.

Piano pedals

Pianos generally have three pedals that alter the way sound is produced: *sustain*, *sostenuto* (sos.), and *una corda* (U.C.). Sustain pedals are also found on vibraphones and celestas.

```
c4\sustainOn d e g
<c, f a>1\sustainOff
c4\sostenutoOn e g c,
<bes d f>1\sostenutoOff
c4\unaCorda d e g
<d fis a>1\treCorde
```



There are three styles of pedal indications: text, bracket, and mixed. The sustain pedal and the una corda pedal use the text style by default while the sostenuto pedal uses mixed by default.

```
c4\sustainOn g c2\sustainOff
\set Staff.pedalSustainStyle = #'mixed
c4\sustainOn g c d
d\sustainOff\sustainOn g, c2\sustainOff
\set Staff.pedalSustainStyle = #'bracket
c4\sustainOn g c d
d\sustainOff\sustainOn g, c2
\bar "|."
```



The placement of the pedal commands matches the physical movement of the sustain pedal during piano performance. Pedalling to the final bar line is indicated by omitting the final pedal off command.

See also

Notation Reference: [\[Ties\]](#), page 42.

Snippets: [Section “Keyboards” in Snippets](#).

Internals Reference: [Section “SustainPedal” in Internals Reference](#), [Section “SustainPedalLineSpanner” in Internals Reference](#), [Section “SustainEvent” in Internals Reference](#), [Section “SostenutoPedal” in Internals Reference](#), [Section “SostenutoPedalLineSpanner” in Internals Reference](#), [Section “SostenutoEvent” in Internals Reference](#), [Section “UnaCordaPedal” in Internals Reference](#), [Section “UnaCordaPedalLineSpanner” in Internals Reference](#), [Section “UnaCordaEvent” in Internals Reference](#), [Section “PianoPedalBracket” in Internals Reference](#), [Section “Piano_pedal_engraver” in Internals Reference](#).

2.2.3 Accordion

This section discusses notation that is unique to the accordion.

Discant symbols

Accordions are often built with more than one set of reeds that may be in unison with, an octave above, or an octave below the written pitch. Each accordion maker has different names for the *shifts* that select the various reed combinations, such as *oboe*, *musette*, or *bandonium*, so a system of symbols has come into use to simplify the performance instructions.

Selected Snippets

Accordion-discant symbols

Accordion discant-specific symbols are added using `\markup`. The vertical placement of the symbols can be tweaked by changing the `\raise` arguments.

```
discant = \markup {
  \musicglyph #"accordion.accDiscant"
}
dot = \markup {
  \musicglyph #"accordion.accDot"
}

\layout { ragged-right = ##t }

% 16 voets register
accBasson = ^\markup {
  \combine
  \discant
  \raise #0.5 \dot
}

% een korig 8 en 16 voets register
accBandon = ^\markup {
  \combine
  \discant
  \combine
  \raise #0.5 \dot
  \raise #1.5 \dot
}

accVCello = ^\markup {
  \combine
  \discant
  \combine
  \raise #0.5 \dot
  \combine
  \raise #1.5 \dot
  \translate #'(1 . 0) \raise #1.5 \dot
}

% 4-8-16 voets register
accHarmon = ^\markup {
  \combine
  \discant
  \combine
  \raise #0.5 \dot
```

```

        \combine
        \raise #1.5 \dot
        \raise #2.5 \dot
    }

accTrombon = ^\markup {
    \combine
    \discant
    \combine
    \raise #0.5 \dot
    \combine
    \raise #1.5 \dot
    \combine
    \translate #'(1 . 0) \raise #1.5 \dot
    \translate #'(-1 . 0) \raise #1.5 \dot
}

% eenkorig 4 en 16 voets register
accOrgan = ^\markup {
    \combine
    \discant
    \combine
    \raise #0.5 \dot
    \raise #2.5 \dot
}

accMaster = ^\markup {
    \combine
    \discant
    \combine
    \raise #0.5 \dot
    \combine
    \raise #1.5 \dot
    \combine
    \translate #'(1 . 0) \raise #1.5 \dot
    \combine
    \translate #'(-1 . 0) \raise #1.5 \dot
    \raise #2.5 \dot
}

accAccord = ^\markup {
    \combine
    \discant
    \combine
    \raise #1.5 \dot
    \combine
    \translate #'(1 . 0) \raise #1.5 \dot
    \combine
    \translate #'(-1 . 0) \raise #1.5 \dot
    \raise #2.5 \dot
}

```

```
accMusette = ^\markup {
  \combine
  \discant
  \combine
  \raise #1.5 \dot
  \combine
  \translate #'(1 . 0) \raise #1.5 \dot
  \translate #'(-1 . 0) \raise #1.5 \dot
}
```

```
accCeleste = ^\markup {
  \combine
  \discant
  \combine
  \raise #1.5 \dot
  \translate #'(-1 . 0) \raise #1.5 \dot
}
```

```
accOboe = ^\markup {
  \combine
  \discant
  \combine
  \raise #1.5 \dot
  \raise #2.5 \dot
}
```

```
accClarin = ^\markup {
  \combine
  \discant
  \raise #1.5 \dot
}
```

```
accPiccolo = ^\markup {
  \combine
  \discant
  \raise #2.5 \dot
}
```

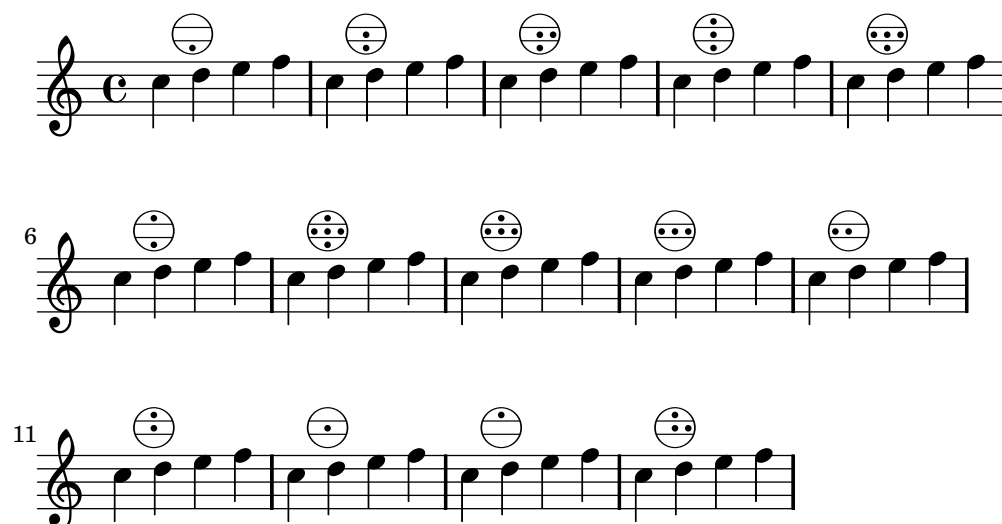
```
accViolin = ^\markup {
  \combine
  \discant
  \combine
  \raise #1.5 \dot
  \combine
  \translate #'(1 . 0) \raise #1.5 \dot
  \raise #2.5 \dot
}
```

```
\relative c'' {
  c4 d\accBasson e f
  c4 d\accBandon e f
  c4 d\accVCello e f
}
```

```

c4 d\accHarmon e f
c4 d\accTrombon e f
\break
c4 d\accOrgan e f
c4 d\accMaster e f
c4 d\accAccord e f
c4 d\accMusette e f
c4 d\accCeleste e f
\break
c4 d\accOboe e f
c4 d\accClarinet e f
c4 d\accPiccolo e f
c4 d\accViolin e f
}

```



See also

Snippets: [Section “Keyboards” in *Snippets*](#).

2.2.4 Harp

This section discusses notation issues that are unique to the harp.

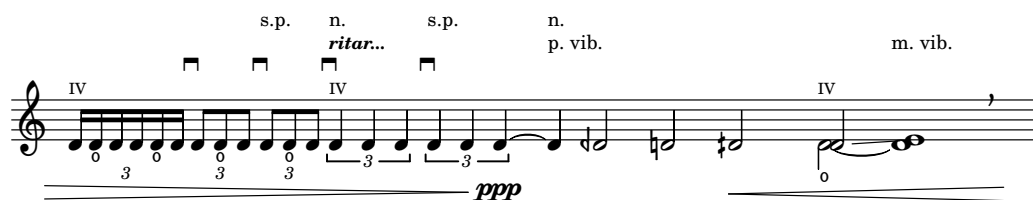
References for harps

Some common characteristics of harp music are covered elsewhere:

- The glissando is the most characteristic harp technique, [\[Glissando\]](#), page 104.
- A *bisbigliando* is written as a tremelo [\[Tremolo repeats\]](#), page 119.
- Natural harmonics are covered under [\[Harmonics\]](#), page 229.
- For directional arpeggios and non-arpeggios, see [\[Arpeggio\]](#), page 105.

See also

Notation Reference: [\[Tremolo repeats\]](#), page 119, [\[Glissando\]](#), page 104, [\[Arpeggio\]](#), page 105, [\[Harmonics\]](#), page 229.



This section provides information and references which are helpful when writing for unfretted string instruments, principally orchestral strings.

2.3.1 Common notation for unfretted strings

There is little specialist notation for unfretted string instruments. The music is notated on a single staff, and usually only a single voice is required. Two voices might be required for some double-stopped or divisi passages.

References for unfretted strings

Most of the notation which is useful for orchestral strings and other bowed instruments is covered elsewhere:

- Textual indications such as “pizz.” and “arco” are added as simple text – see [Text scripts], page 176.
- Fingerings, including the thumb indication, are described in [Fingering instructions], page 165.
- Double stopping is normally indicated by writing a chord, see [Chorded notes], page 121. Directives for playing chords may be added, see [Arpeggio], page 105.
- A template for a string quartet can be found in Section “String quartet” in *Learning Manual*. Others are shown in the snippets.

See also

Learning Manual: Section “String quartet” in *Learning Manual*.

Notation Reference: [Text scripts], page 176, [Fingering instructions], page 165, [Chorded notes], page 121, [Arpeggio], page 105.

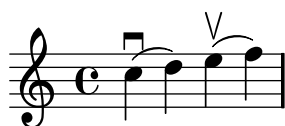
Snippets: Section “Unfretted strings” in *Snippets*.

Bowing indications

Bowing indications are created as articulations, which are described in [Articulations and ornamentations], page 90.

The bowing commands, `\upbow` and `\downbow`, are used with slurs as follows:

`c4(\downbow d) e(\upbow f)`

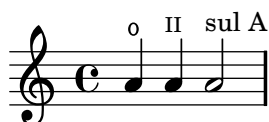


and the following example shows three ways in which an open A string on a violin might be indicated:

`a4 \open`

`a^\markup { \teeny "II" }`

`a2^\markup { \small "sul A" }`



Predefined commands

`\downbow`, `\upbow`, `\open`.

See also

Notation Reference: [\[Articulations and ornamentations\]](#), page 90, [\[Slurs\]](#), page 99.

Harmonics

Natural harmonics

Natural harmonics can be notated in several ways. A diamond-shaped note head generally means to touch the string where you would stop the note if it were not a diamond.

```
d4 e4.
\harmonicsOn
d8 e e
d4 e4.
\harmonicsOff
d8 e e
```



Alternatively a normal note head is shown at the pitch to be sounded together with a small circle to indicate it should be played as a harmonic:

```
d2^\flageolet d_\flageolet
```



A smaller circle may be created, see the snippet list in [\[References for unfretted strings\]](#), page 228.

Artificial harmonics

Artificial harmonics are notated with two notes, one with a normal note head indicating the stopped position and one with an open diamond note head to indicate the harmonic position.

Artificial harmonics indicated with `\harmonic` do not show the dots. The context property `harmonicDots` should be set if dots are required.

```
<e a\harmonic>2. <c g'\harmonic>4
\set harmonicDots = ##t
<e a\harmonic>2. <c g'\harmonic>4
```



Note: `\harmonic` **must** be placed inside a chord construct even if there is only a single note. Normally `\harmonicsOn` would be used in this situation.

See also

Music Glossary: [Section “harmonics” in Music Glossary](#).

Notation Reference: [\[Special note heads\]](#), page 29, [\[References for unfretted strings\]](#), page 228.

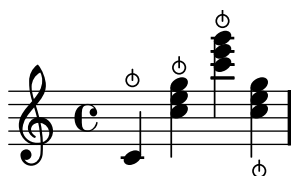
Snap (Bartók) pizzicato

Selected Snippets

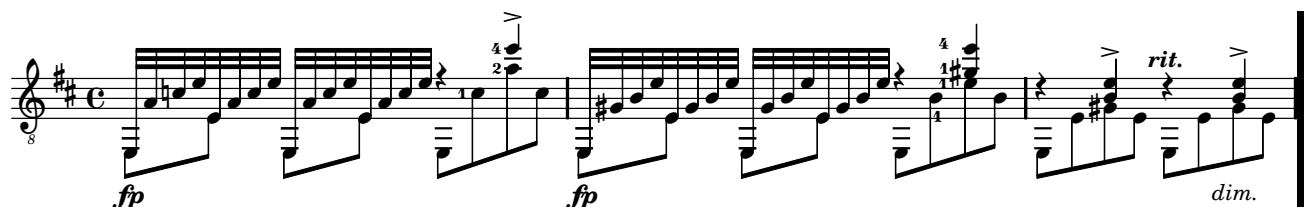
Snap-pizzicato (“Bartok pizzicato”)

A snap-pizzicato (also known as “Bartok pizzicato”) is a “strong pizzicato where the string is plucked vertically by snapping and rebounds off the fingerboard of the instrument” (Wikipedia). It is denoted by a circle with a vertical line going from the center upwards outside the circle.

```
\relative c' {
  c4\snappizzicato
  <c' e g>4\snappizzicato
  <c' e g>4^\snappizzicato
  <c, e g>4_\snappizzicato
}
```



2.4 Fretted string instruments



This section discusses several aspects of music notation that are unique to fretted string instruments.

2.4.1 Common notation for fretted strings

This section discusses common notation that is unique to fretted string instruments.

References for fretted strings

Music for fretted string instruments is normally notated on a single staff, either in traditional music notation or in tablature. Sometimes the two types are combined, and it is especially common in popular music to use chord diagrams above a staff of traditional notation. The guitar and the banjo are transposing instruments, sounding an octave lower than written. Scores for these instruments should use the "treble_8" clef (or `\transposition c` to get correct MIDI output). Some other elements pertinent to fretted string instruments are covered elsewhere:

- Fingerings are indicated as shown in [Fingering instructions], page 165.
- Instructions for *Laissez vibrer* ties as well as ties on arpeggios and tremolos can be found in [Ties], page 42.
- Instructions for handling multiple voices can be found in [Collision resolution], page 126.
- Instructions for indicating harmonics can be found in [Harmonics], page 229.

See also

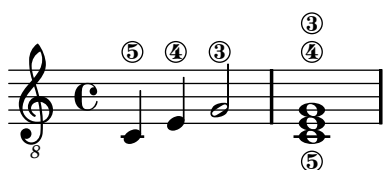
Notation Reference: [Fingering instructions], page 165, [Ties], page 42, [Collision resolution], page 126, [Instrument names], page 154, [Writing music in parallel], page 133, [Arpeggio], page 105, Section A.10 [List of articulations], page 513, [Clef], page 13, [Instrument transpositions], page 19.

String number indications

The string on which a note should be played may be indicated by appending `\number` to a note inside a chord construct `<>`.

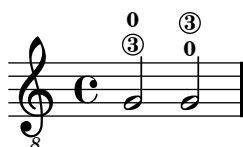
Note: String numbers **must** be defined inside a chord construct even if there is only a single note.

```
\clef "treble_8"
<c\5>4 <e\4> <g\3>2
<c,\5 e\4 g\3>1
```



When fingerings and string indications are used together, their placement is controlled by the order in which the two items appear in the code:

```
\clef "treble_8"
<g\3-0>2
<g-0\3>
```

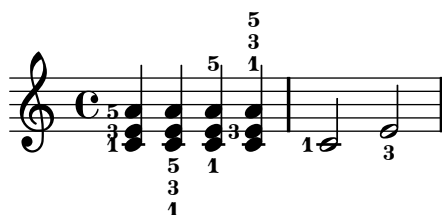


Selected Snippets

Controlling the placement of chord fingerings

The placement of fingering numbers can be controlled precisely. For fingering orientation to apply, you must use a chord construct `<>` even if it is a single note.

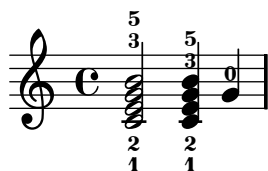
```
\relative c' {
  \set fingeringOrientations = #'(left)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(down)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(down right up)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(up)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(left)
  <c-1>2
  \set fingeringOrientations = #'(down)
  <e-3>2
}
```



Allowing fingerings to be printed inside the staff

By default, vertically oriented fingerings are positioned outside the staff. However, this behavior can be canceled. Note: you must use a chord construct `<>`, even if it is only a single note.

```
\relative c' {
  <c-1 e-2 g-3 b-5>2
  \override Fingering #'staff-padding = #'()
  <c-1 e-2 g-3 b-5>4 <g'-0>
}
```



See also

Notation Reference: [\[Fingering instructions\]](#), page 165.

Snippets: [Section “Fretted strings”](#) in *Snippets*.

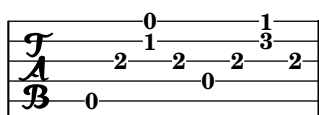
Internals Reference: [Section “StringNumber”](#) in *Internals Reference*, [Section “Fingering”](#) in *Internals Reference*.

Default tablatures

Music for plucked string instruments is frequently notated using a finger/touch notation or tablature. In contrast to traditional notation pitches are not denoted with note heads, but by numbers (or letterlike symbols in historical intavolatura). The numbers indicate on which string and fret a note must be played. The numbers are printed on top of each other if they are to be played simultaneously.

By default, string 1 is the highest, and the tuning defaults to the standard guitar tuning (with 6 strings). The notes are printed as tablature, by using `TabStaff` and `TabVoice` contexts. A calligraphic tablature clef is added automatically.

```
\new TabStaff \relative c' {
  a,8 a' <c e> a
  d, a' <d f> a
}
```



Default tablatures do not contain any symbols for tone duration nor any other musical symbols such as e.g. expressive marks.

```
symbols = {
  \time 3/4
  c4-.^"Allegro" d( e)
  f4-. \f g a^ \fermata
  \mark \default
  c8_. \<\( c16 c ~ c2\!
  c'2. \prall\}
}
```

```
\score {
  <<
    \new Staff { \clef "G_8" \symbols }
    \new TabStaff { \symbols }
  >>
}
```

If all musical symbols used in traditional notation should also show up in tablature one has to apply the command `\tabFullNotation` in a `TabStaff`-context. Please bear in mind that half notes are double-stemmed in tablature in order to distinguish them from quarter notes.

```
\new StaffGroup <<
  \new Staff \relative c {
    \clef "treble_8"
    \time 2/4
    c16 d e f g4
    c,16\5 d\5 e\4 f\4 g4\4
    c,16 d e f g4
  }
  \new TabStaff \relative c {
    c16 d e f g4
    c,16\5 d\5 e\4 f\4 g4\4
    \set TabStaff.minimumFret = #5
    c,16 d e f g4
  }
>>
```

The first system of the musical score for 'The Little Boat' is shown. It consists of a treble clef staff with a 2/4 time signature and a bass clef staff. The treble staff contains a melody of eighth notes: G4, A4, B4, C5, B4, A4, G4. The bass staff contains a bass line of eighth notes: E3, F3, G3, A3, G3, F3, E3. The system is divided into three measures by vertical bar lines.

Chord constructs can be repeated by the chord repetition symbol `q`. To use this feature in combination with tablature, `\tabChordRepetition` is provided. It preserves the string information explicitly given within chord constructs so repeated chords get identical tablature representations.

```
\tabChordRepetition
```

```
guitar = \relative c' {
  r8 < gis\4 cis\3 b\2 > ~ q4 q8 ~ q q4
}

\new StaffGroup <<
  \new Staff {
    \clef "treble_8"
    \override Voice.StringNumber #'transparent = ##t
    \guitar
  }
  \new TabStaff {
    \guitar
  }
>>
```

Ties over a line break are parenthesized by default. The same holds for the second alternative of a repeat.

```
ties = \relative c' {
  \repeat volta 2 {
    e2. f4~
    f2 g2~
  }
  \alternative {
    { g4 f2. }
    { g4\repeatTie c,2. }
  }
  b1~
  \break
  b1
  \bar "|."
}
```

```
\score {
  <<
    \new StaffGroup <<
      \context Staff {
        \clef "treble_8"
```

```

        \ties
    }
    \context TabStaff {
        \ties
    }
    >>
>>
\layout {
indent = #0
ragged-right = ##t
}
}

```

The command `\hideSplitTiedTabNotes` cancels the behavior of engraving fret numbers in parentheses:

```

ties = \relative c' {
  \repeat volta 2 {
    e2. f4~
    f2 g2~ }
  \alternative {
    { g4 f2. }
    { g4\repeatTie c,2. }
  }
  b1~
  \break
  b1
  \bar "|"
}

```

```

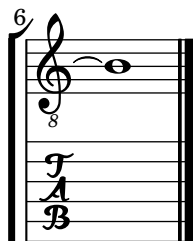
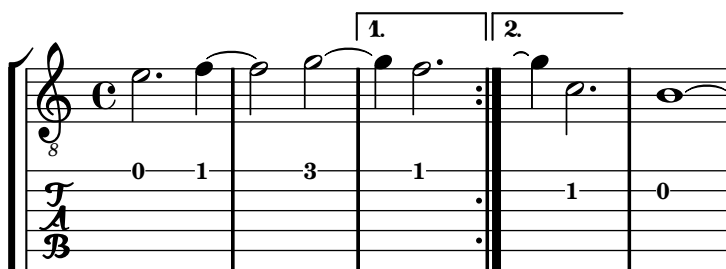
\score {
  <<
    \new StaffGroup <<
      \context Staff {
        \clef "treble_8"
        \ties

```

```

    }
    \context TabStaff {
      \hideSplitTiedTabNotes
      \ties
    }
  >>
  >>
  \layout {
    indent = #0
    ragged-right = ##t
  }
}

```

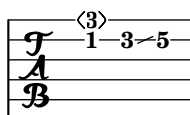


Harmonic indications and slides can be added to tablature notation.

```

\new TabStaff {
  \new TabVoice {
    <c g'\harmonic> d\2\glissando e\2
  }
}

```



Selected Snippets

Stem and beam behavior in tablature

The direction of stems is controlled the same way in tablature as in traditional notation. Beams can be made horizontal, as shown in this example.

```

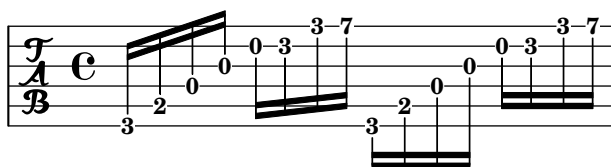
\new TabStaff {
  \relative c {
    \tabFullNotation
    g16 b d g b d g b
    \stemDown
    \override Beam #'damping = #+inf.0
  }
}

```

```

    g,,16 b d g b d g b
  }
}

```



Polyphony in tablature

Polyphony is created the same way in a TabStaff as in a regular staff.

```

upper = \relative c' {
  \time 12/8
  \key e \minor
  \voiceOne
  r4. r8 e, fis g16 b g e e' b c b a g fis e
}

```

```

lower = \relative c {
  \key e \minor
  \voiceTwo
  r16 e d c b a g4 fis8 e fis g a b c
}

```

```

\score {
  <<
    \new StaffGroup = "tab with traditional" <<
      \new Staff = "guitar traditional" <<
        \clef "treble_8"
        \context Voice = "upper" \upper
        \context Voice = "lower" \lower
      >>
      \new TabStaff = "guitar tab" <<
        \context TabVoice = "upper" \upper
        \context TabVoice = "lower" \lower
      >>
    >>
  >>
}

```

See also

Notation Reference: [\[Stems\]](#), page 170.

Snippets: [Section “Fretted strings” in *Snippets*](#).

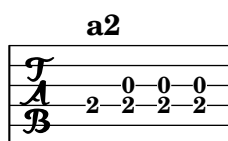
Internals Reference: [Section “TabNoteHead” in *Internals Reference*](#), [Section “TabStaff” in *Internals Reference*](#), [Section “TabVoice” in *Internals Reference*](#), [Section “Beam” in *Internals Reference*](#).

Known issues and warnings

Chords are not handled in a special way, and hence the automatic string selector may easily select the same string for 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 }
  >>
>>
```



Guitar special effects are limited to harmonics and slides.

Custom tablatures

LilyPond tablature automatically calculates the fret for a note based on the string to which the note is assigned. In order to do this, the tuning of the strings must be specified. The tuning of the strings is given in the `stringTunings` property.

LilyPond comes with predefined string tunings for banjo, mandolin, guitar, bass guitar and ukulele. LilyPond automatically sets the correct transposition for predefined tunings. The following example is for bass guitar, which sounds an octave lower than written.

```
<<
  \new Staff {
    \clef "bass_8"
    \relative c, {
      c4 d e f
    }
  }
  \new TabStaff {
    \set TabStaff.stringTunings = #bass-tuning
    \relative c, {
      c4 d e f
    }
  }
>>
```



The default string tuning is `guitar-tuning`, which is the standard EADGBE tuning. Some other predefined tunings are `guitar-open-g-tuning`, `mandolin-tuning` and `banjo-open-g-tuning`. The predefined string tunings are found in `scm/tablature.scm`.

A string tuning is a Scheme list of string pitches, one for each string, ordered by string number from 1 to N, where string 1 is at the top of the tablature staff and string N is at the bottom. This ordinarily results in ordering from highest pitch to lowest pitch, but some instruments (e.g. ukulele) do not have strings ordered by pitch.

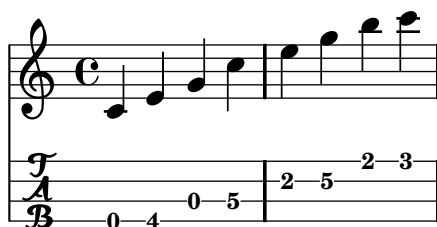
A string pitch in a string tuning list is the pitch difference of the open string from middle C measured in semitones. The string pitch must be an integer. LilyPond calculates the actual pitch of the string by adding the string tuning pitch to the actual pitch for middle C.

LilyPond automatically calculates the number of strings in the `TabStaff` as the number of elements in `stringTunings`.

Any desired string tuning can be created. For example, we can define a string tuning for a four-string instrument with pitches of `a''`, `d''`, `g'`, and `c'`:

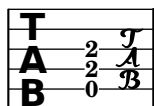
```
mynotes = {
  c'4 e' g' c'' |
  e''4 g'' b'' c'''
}

<<
\new Staff {
  \clef treble
  \mynotes
}
\new TabStaff {
  \set TabStaff.stringTunings = #'(21 14 7 0)
  \mynotes
}
>>
```

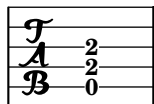


A modern tab clef can also be used.

```
\new TabStaff {
  \clef moderntab
  <a, e a>1
  \break
  \clef tab
  <a, e a>1
}
```



2



The modern tab clef supports tablatures from 4 to 7 strings.

See also

Installed Files: ‘`scm/tablature.scm`’.

Snippets: [Section “Fretted strings” in *Snippets*](#).

Internals Reference: [Section “Tab_note_heads_engraver” in *Internals Reference*](#).

Known issues and warnings

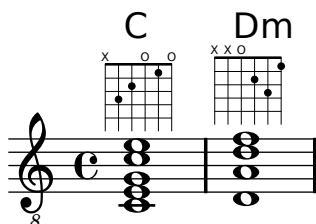
Automatic tablature calculations do not work properly in most cases for instruments where string pitches do not vary monotonically with string number, such as ukuleles.

Fret diagram markups

Fret diagrams can be added to music as a markup to the desired note. The markup contains information about the desired fret diagram. 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 A.8 \[Text markup commands\], page 474](#).

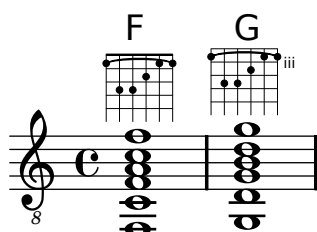
The standard fret diagram markup string indicates the string number and the fret number for each dot to be placed on the string. In addition, open and unplayed (muted) strings can be indicated.

```
<<
\context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context Staff {
  \clef "treble_8"
  < c e g c' e' > 1 ^\markup
    \fret-diagram #"6-x;5-3;4-2;3-o;2-1;1-o;"
  < d a d' f' > ^\markup
    \fret-diagram #"6-x;5-x;4-o;3-2;2-3;1-1;"
}
>>
```



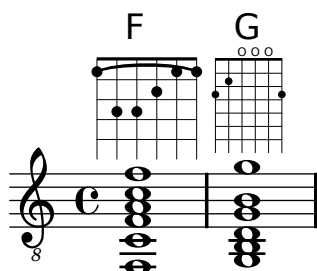
Barre indications can be added to the diagram from the fret-diagram markup string.

```
<<
\context ChordNames {
  \chordmode {
    f1 g
  }
}
\context Staff {
  \clef "treble_8"
  < f, c f a c' f'>1 ^\markup
    \fret-diagram #"c:6-1-1;6-1;5-3;4-3;3-2;2-1;1-1;"
  < g, d g b d' g'> ^\markup
    \fret-diagram #"c:6-1-3;6-3;5-5;4-5;3-4;2-3;1-3;"
}
>>
```



The size of the fret diagram, and the number of frets in the diagram can be changed in the fret-diagram markup string.

```
<<
\context ChordNames {
  \chordmode {
    f1 g
  }
}
\context Staff {
  \clef "treble_8"
  < f, c f a c' f'>1 ^\markup
    \fret-diagram #"s:1.5;c:6-1-1;6-1;5-3;4-3;3-2;2-1;1-1;"
  < g, b, d g b g'> ^\markup
    \fret-diagram #"h:6;6-3;5-2;4-o;3-o;2-o;1-3;"
}
>>
```



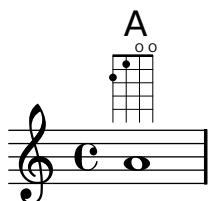
The number of strings in a fret diagram can be changed to accomodate different instruments such as banjos and ukuleles with the fret-diagram markup string.

```
<<
\context ChordNames {
```

```

\chordmode {
  a1
}
}
\context Staff {
  %% An A chord for ukulele
  a'1 ^\markup \fret-diagram #"w:4;4-2-2;3-1-1;2-o;1-o;"
}
>>

```

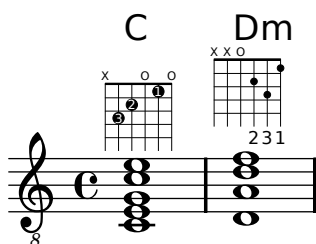


Fingering indications can be added, and the location of fingering labels can be controlled by the fret-diagram markup string.

```

<<
\context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context Staff {
  \clef "treble_8"
  < c e g c' e' > 1 ^\markup
    \fret-diagram #"f:1;6-x;5-3-3;4-2-2;3-o;2-1-1;1-o;"
  < d a d' f' > ^\markup
    \fret-diagram #"f:2;6-x;5-x;4-o;3-2-2;2-3-3;1-1-1;"
}
>>

```



Dot radius and dot position can be controlled with the fret-diagram markup string.

```

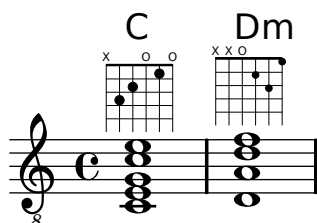
<<
\context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context Staff {
  \clef "treble_8"
  < c e g c' e' > 1 ^\markup

```

```

\fret-diagram #"d:0.35;6-x;5-3;4-2;3-o;2-1;1-o;"
< d a d' f' > ^\markup
\fret-diagram #"p:0.2;6-x;5-x;4-o;3-2;2-3;1-1;"
}
>>

```

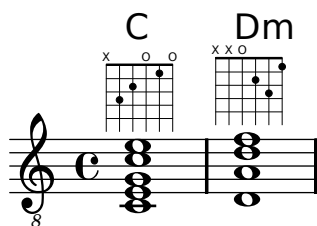


The fret-diagram-terse markup string omits string numbers; the string number is implied by the presence of semicolons. There is one semicolon for each string in the diagram. The first semicolon corresponds to the highest string number and the last semicolon corresponds to the first string. Mute strings, open strings, and fret numbers can be indicated.

```

<<
\context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context Staff {
  \clef "treble_8"
  < c e g c' e' > 1 ^\markup
  \fret-diagram-terse #"x;3;2;o;1;o;"
  < d a d' f' > ^\markup
  \fret-diagram-terse #"x;x;o;2;3;1;"
}
>>

```



Barre indicators can be included in the fret-diagram-terse markup string.

```

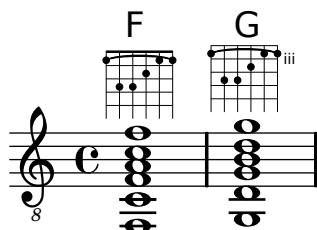
<<
\context ChordNames {
  \chordmode {
    f1 g
  }
}
\context Staff {
  \clef "treble_8"
  < f, c f a c' f' > 1 ^\markup
  \fret-diagram-terse #"1-(;3;3;2;1;1-);"
  < g, d g b d' g' > ^\markup
}
>>

```

```

\fret-diagram-terse #"3-(;5;5;4;3;3-);"
}
>>

```

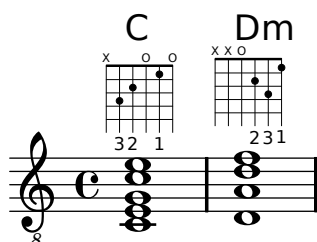


Fingering indications can be included in the fret-diagram-terse markup string.

```

<<
\context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context Staff {
  \override Voice.TextScript
    #'(fret-diagram-details finger-code) = #'below-string
  \clef "treble_8"
  < c e g c' e' > 1 ^\markup
    \fret-diagram-terse #"x;3-3;2-2;o;1-1;o;"
  < d a d' f' > ^\markup
    \fret-diagram-terse #"x;x;o;2-2;3-3;1-1;"
}
>>

```



Other fret diagram properties must be adjusted using `\override` when using the fret-diagram-terse markup.

The fret-diagram-verbose markup string is in the format of a Scheme list. Each element of the list indicates an item to be placed on the fret diagram.

```

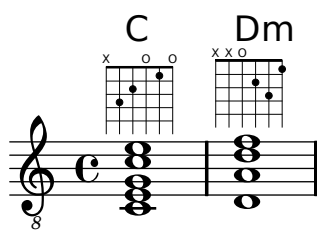
<< \context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context Staff {
  \clef "treble_8"
  < c e g c' e' > 1 ^\markup
    \fret-diagram-verbose #'(
      (mute 6)

```

```

        (place-fret 5 3)
        (place-fret 4 2)
        (open 3)
        (place-fret 2 1)
        (open 1)
    )
    < d a d' f' > ^\markup
    \fret-diagram-verbose #'(
        (mute 6)
        (mute 5)
        (open 4)
        (place-fret 3 2)
        (place-fret 2 3)
        (place-fret 1 1)
    )
}
>>

```



Fingering indications and barres can be included in a fret-diagram-verbose markup string. Unique to the fret-diagram-verbose interface is a capo indication that can be placed on the fret diagram. The capo indication is a thick bar that covers all strings. The fret with the capo will be the lowest fret in the fret diagram.

```

<<
\context ChordNames {
  \chordmode {
    f1 g c
  }
}
\context Staff {
  \clef "treble_8"
  \override Voice.TextScript
    #'(fret-diagram-details finger-code) = #'below-string

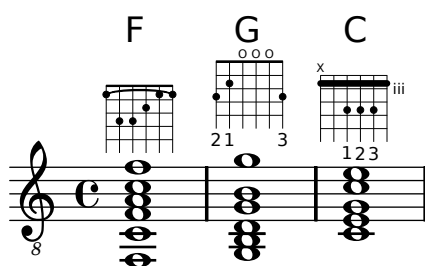
  < f, c f a c' f' >1 ^\markup
  \fret-diagram-verbose #'(
    (place-fret 6 1)
    (place-fret 5 3)
    (place-fret 4 3)
    (place-fret 3 2)
    (place-fret 2 1)
    (place-fret 1 1)
    (barre 6 1 1)
  )
  < g, b, d g b g' > ^\markup
  \fret-diagram-verbose #'(

```

```

        (place-fret 6 3 2)
        (place-fret 5 2 1)
        (open 4)
        (open 3)
        (open 2)
        (place-fret 1 3 3)
    )
    < c e g c' e' > ^\markup
      \fret-diagram-verbose #'(
        (capo 3)
        (mute 6)
        (place-fret 4 5 1)
        (place-fret 3 5 2)
        (place-fret 2 5 3)
      )
  }
>>

```



All other fret diagram properties must be adjusted using `\override` when using the `fret-diagram-verbose` markup.

The graphical layout of a fret diagram can be customized according to user preference through the properties of the `fret-diagram-interface`. Details are found at [Section “fret-diagram-interface” in *Internals Reference*](#). For a fret diagram markup, the interface properties belong to `Voice.TextScript`.

Selected Snippets

Customizing markup fret diagrams

Fret diagram properties can be set through `'fret-diagram-details`. For markup fret diagrams, overrides can be applied to the `Voice.TextScript` object or directly to the markup.

```

<<
\chords { c1 | c | c | d }

\new Voice = "mel" {
  \textLengthOn
  % Set global properties of fret diagram
  \override TextScript #'size = #'1.2
  \override TextScript
    #'(fret-diagram-details finger-code) = #'in-dot
  \override TextScript
    #'(fret-diagram-details dot-color) = #'white

  %% C major for guitar, no barre, using defaults
  % terse style

```

```

c'1^\markup { \fret-diagram-terse #"x;3-3;2-2;o;1-1;o;" }

%% C major for guitar, barred on third fret
% verbose style
% size 1.0
% roman fret label, finger labels below string, straight barre
c'1^\markup {
% standard size
\override #'(size . 1.0) {
  \override #'(fret-diagram-details . (
    (number-type . roman-lower)
    (finger-code . in-dot)
    (barre-type . straight))) {
    \fret-diagram-verbose #'((mute 6)
      (place-fret 5 3 1)
      (place-fret 4 5 2)
      (place-fret 3 5 3)
      (place-fret 2 5 4)
      (place-fret 1 3 1)
      (barre 5 1 3))
    }
  }
}

%% C major for guitar, barred on third fret
% verbose style
% landscape orientation, arabic numbers, M for mute string
% no barre, fret label down or left, small mute label font
c'1^\markup {
  \override #'(fret-diagram-details . (
    (finger-code . below-string)
    (number-type . arabic)
    (label-dir . -1)
    (mute-string . "M")
    (orientation . landscape)
    (barre-type . none)
    (xo-font-magnification . 0.4)
    (xo-padding . 0.3))) {
    \fret-diagram-verbose #'((mute 6)
      (place-fret 5 3 1)
      (place-fret 4 5 2)
      (place-fret 3 5 3)
      (place-fret 2 5 4)
      (place-fret 1 3 1)
      (barre 5 1 3))
    }
  }
}

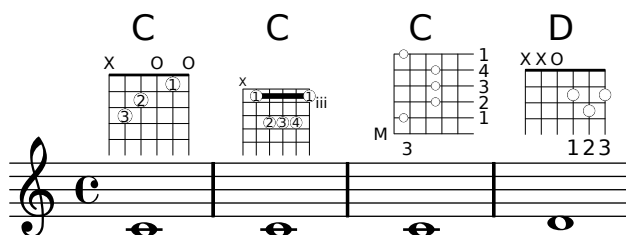
%% simple D chord
% terse style
% larger dots, centered dots, fewer frets
% label below string

```

```

d'1^\markup {
  \override #'(fret-diagram-details . (
    (finger-code . below-string)
    (dot-radius . 0.35)
    (dot-position . 0.5)
    (fret-count . 3))) {
    \fret-diagram-terse #"x;x;o;2-1;3-2;2-3;"
  }
}
}
}
>>

```



See also

Notation Reference: [Section A.8 \[Text markup commands\]](#), page 474.

Snippets: [Section “Fretted strings”](#) in *Snippets*.

Internals Reference: [Section “fret-diagram-interface”](#) in *Internals Reference*.

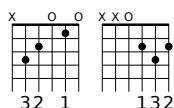
Predefined fret diagrams

Fret diagrams can be displayed using the `FretBoards` context. By default, the `FretBoards` context will display fret diagrams that are stored in a lookup table:

```

\include "predefined-guitar-fretboards.ly"
\context FretBoards {
  \chordmode {
    c1 d
  }
}

```



The default predefined fret diagrams are contained in the file `predefined-guitar-fretboards.ly`. Fret diagrams are stored based on the pitches of a chord and the value of `stringTunings` that is currently in use. `predefined-guitar-fretboards.ly` contains predefined fret diagrams only for `guitar-tuning`. Predefined fret diagrams can be added for other instruments or other tunings by following the examples found in `predefined-guitar-fretboards.ly`.

Fret diagrams for the ukulele are contained in the file `predefined-ukulele-fretboards.ly`.
`\include "predefined-ukulele-fretboards.ly"`

```
myChords = \chordmode { a1 a:m a:aug }
```

```
\new ChordNames {
```

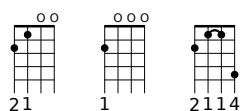
```

\myChords
}

\new FretBoards {
  \set stringTunings = #ukulele-tuning
  \myChords
}

```

A Am A+

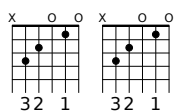


Chord pitches can be entered either as simultaneous music or using chord mode (see [\[Chord mode overview\]](#), page 279).

```

\include "predefined-guitar-fretboards.ly"
\context FretBoards {
  \chordmode {c1}
  <c' e' g'>1
}

```



It is common that both chord names and fret diagrams are displayed together. This is achieved by putting a **ChordNames** context in parallel with a **FretBoards** context and giving both contexts the same music.

```

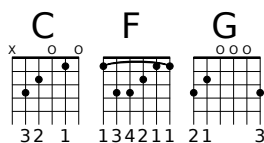
\include "predefined-guitar-fretboards.ly"
mychords = \chordmode{
  c1 f g
}

```

```

<<
  \context ChordNames {
    \mychords
  }
  \context FretBoards {
    \mychords
  }
>>

```



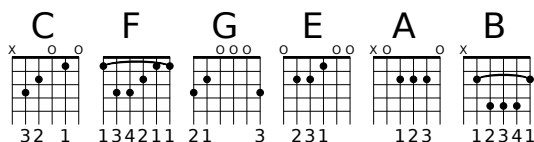
Predefined fret diagrams are transposable, as long as a diagram for the transposed chord is stored in the fret diagram table.

```

\include "predefined-guitar-fretboards.ly"
mychords = \chordmode{
  c1 f g
}

mychordlist = {
  \mychords
  \transpose c e { \mychords}
}
<<
  \context ChordNames {
    \mychordlist
  }
  \context FretBoards {
    \mychordlist
  }
>>

```



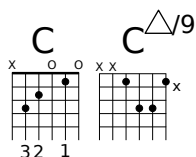
The predefined fret diagram table for guitar contains eight chords (major, minor, augmented, diminished, dominant seventh, major seventh, minor seventh, dominant ninth) for each of 17 keys. The predefined fret diagram table for ukulele contains these chords plus an additional three chords (major sixth, suspended second, and suspended fourth). A complete list of the predefined fret diagrams is shown in [Section A.3 \[Predefined fretboard diagrams\]](#), page 447. If there is no entry in the table for a chord, the FretBoards engraver will calculate a fret-diagram using the automatic fret diagram functionality described in [\[Automatic fret diagrams\]](#), page 258.

```

\include "predefined-guitar-fretboards.ly"
mychords = \chordmode{
  c1 c:maj9
}

<<
  \context ChordNames {
    \mychords
  }
  \context FretBoards {
    \mychords
  }
>>

```



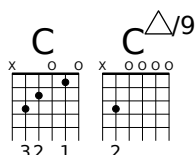
Fret diagrams can be added to the fret diagram table. To add a diagram, you must specify the chord for the diagram, the tuning to be used, and a definition for the diagram. The diagram definition can be either a fret-diagram-terse definition string or a fret-diagram-verbose marking list.

```
\include "predefined-guitar-fretboards.ly"
```

```
\storePredefinedDiagram \chordmode {c:maj9}
    #guitar-tuning
    #"x;3-2;o;o;o;o;"
```

```
mychords = \chordmode{
  c1 c:maj9
}
```

```
<<
  \context ChordNames {
    \mychords
  }
  \context FretBoards {
    \mychords
  }
>>
```



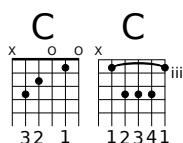
Different fret diagrams for the same chord name can be stored using different octaves of pitches. The different octave should be at least two octaves above or below the default octave, because the octaves above and below the default octave are used for transposing fretboards.

```
\include "predefined-guitar-fretboards.ly"
```

```
\storePredefinedDiagram \chordmode {c''}
    #guitar-tuning
    #(offset-fret 2 (chord-shape 'bes guitar-tuning))
```

```
mychords = \chordmode{
  c1 c''
}
```

```
<<
  \context ChordNames {
    \mychords
  }
  \context FretBoards {
    \mychords
  }
>>
```



In addition to fret diagrams, LilyPond stores an internal list of chord shapes. The chord shapes are fret diagrams that can be shifted along the neck to different positions to provide

different chords. Chord shapes can be added to the internal list and then used to define predefined fret diagrams. Because they can be moved to various positions on the neck, chord shapes will normally not contain any open strings. Like fret diagrams, chord shapes can be entered as either fret-diagram-terse strings or fret-diagram-verbose marking lists.

```
\include "predefined-guitar-fretboards.ly"

% add a new chord shape

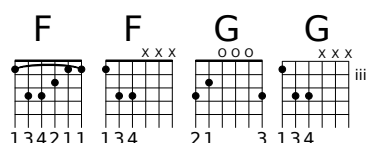
\addChordShape #'powerf #guitar-tuning #"1-1;3-3;3-4;x;x;x;"

% add some new chords based on the power chord shape

\storePredefinedDiagram \chordmode {f''}
    #guitar-tuning
    #(chord-shape 'powerf guitar-tuning)
\storePredefinedDiagram \chordmode {g''}
    #guitar-tuning
    #(offset-fret 2 (chord-shape 'powerf guitar-tuning))

mychords = \chordmode{
  f1 f'' g g''
}

<<
  \context ChordNames {
    \mychords
  }
  \context FretBoards {
    \mychords
  }
>>
```



The graphical layout of a fret diagram can be customized according to user preference through the properties of the `fret-diagram-interface`. Details are found at [Section “fret-diagram-interface” in *Internals Reference*](#). For a predefined fret diagram, the interface properties belong to `FretBoards.FretBoard`.

Selected Snippets

Customizing fretboard fret diagrams

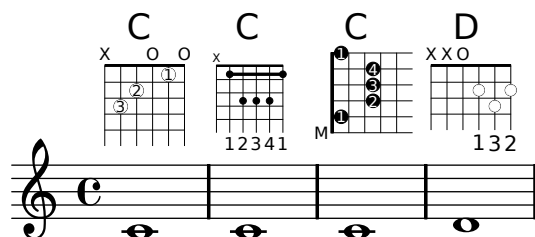
Fret diagram properties can be set through `'fret-diagram-details`. For `FretBoard` fret diagrams, overrides are applied to the `FretBoards.FretBoard` object. Like `Voice`, `FretBoards` is a bottom level context, therefore can be omitted in property overrides.

```
\include "predefined-guitar-fretboards.ly"
\storePredefinedDiagram \chordmode { c' }
    #guitar-tuning
    #"x;1-1-(;3-2;3-3;3-4;1-1-);"
```

```

<<
\new ChordNames {
  \chordmode { c1 | c | c | d }
}
\new FretBoards {
  % Set global properties of fret diagram
  \override FretBoards.FretBoard #'size = #'1.2
  \override FretBoard
    #'(fret-diagram-details finger-code) = #'in-dot
  \override FretBoard
    #'(fret-diagram-details dot-color) = #'white
  \chordmode {
    c
    \once \override FretBoard #'size = #'1.0
    \once \override FretBoard
      #'(fret-diagram-details barre-type) = #'straight
    \once \override FretBoard
      #'(fret-diagram-details dot-color) = #'black
    \once \override FretBoard
      #'(fret-diagram-details finger-code) = #'below-string
    c'
    \once \override FretBoard
      #'(fret-diagram-details barre-type) = #'none
    \once \override FretBoard
      #'(fret-diagram-details number-type) = #'arabic
    \once \override FretBoard
      #'(fret-diagram-details orientation) = #'landscape
    \once \override FretBoard
      #'(fret-diagram-details mute-string) = #'"M"
    \once \override FretBoard
      #'(fret-diagram-details label-dir) = #LEFT
    \once \override FretBoard
      #'(fret-diagram-details dot-color) = #'black
    c'
    \once \override FretBoard
      #'(fret-diagram-details finger-code) = #'below-string
    \once \override FretBoard
      #'(fret-diagram-details dot-radius) = #0.35
    \once \override FretBoard
      #'(fret-diagram-details dot-position) = #0.5
    \once \override FretBoard
      #'(fret-diagram-details fret-count) = #3
    d
  }
}
\new Voice {
  c'1 | c' | c' | d'
}
>>

```



Defining predefined fretboards for other instruments

Predefined fret diagrams can be added for new instruments in addition to the standards used for guitar. This file shows how this is done by defining a new string-tuning and a few predefined fretboards for the Venezuelan cuatro.

This file also shows how fingerings can be included in the chords used as reference points for the chord lookup, and displayed in the fret diagram and the `TabStaff`, but not the music.

These fretboards are not transposable because they contain string information. This is planned to be corrected in the future.

```
% add FretBoards for the Cuatro
% Note: This section could be put into a separate file
%     predefined-cuatro-fretboards.ly
%     and \included into each of your compositions

cuatroTuning = #'(11 18 14 9)

dSix = { <a\4 b\1 d\3 fis\2> }
dMajor = { <a\4 d\1 d\3 fis \2> }
aMajSeven = { <a\4 cis\1 e\3 g\2> }
dMajSeven = { <a\4 c\1 d\3 fis\2> }
gMajor = { <b\4 b\1 d\3 g\2> }

\storePredefinedDiagram \dSix
    #cuatroTuning
    #"o;o;o;o;"
\storePredefinedDiagram \dMajor
    #cuatroTuning
    #"o;o;o;3-3;"
\storePredefinedDiagram \aMajSeven
    #cuatroTuning
    #"o;2-2;1-1;2-3;"
\storePredefinedDiagram \dMajSeven
    #cuatroTuning
    #"o;o;o;1-1;"
\storePredefinedDiagram \gMajor
    #cuatroTuning
    #"2-2;o;1-1;o;"

% end of potential include file /predefined-cuatro-fretboards.ly

#(set-global-staff-size 16)

primerosNames = \chordmode {
  d:6 d a:maj7 d:maj7
  g
}
```

```

primeros = {
  \dSix \dMajor \aMajSeven \dMajSeven
  \gMajor
}

\score {
  <<
    \new ChordNames {
      \set chordChanges = ##t
      \primerosNames
    }

    \new Staff {
      \new Voice \with {
        \remove "New_fingering_engraver"
      }
      \relative c'' {
        \primeros
      }
    }

    \new FretBoards {
      \set stringTunings = #cuatroTuning
      \override FretBoard
        #'(fret-diagram-details string-count) = #'4
      \override FretBoard
        #'(fret-diagram-details finger-code) = #'in-dot
      \primeros
    }

    \new TabStaff \relative c'' {
      \set TabStaff.stringTunings = #cuatroTuning
      \primeros
    }

  >>

  \layout {
    \context {
      \Score
      \override SpacingSpanner
        #'base-shortest-duration = #(ly:make-moment 1 16)
    }
  }
  \midi { }
}

```

chordChanges for FretBoards

FretBoards can be set to display only when the chord changes or at the beginning of a new line.

```
\include "predefined-guitar-fretboards.ly"
```

```
myChords = \chordmode {
  c1 c1 \break
  \set chordChanges = ##t
  c1 c1 \break
  c1 c1 \break
}
```

```
<<
  \new ChordNames { \myChords }
  \new FretBoards { \myChords }
  \new Staff { \myChords }
>>
```

See also

Notation Reference: [Custom tablatures], page 239, [Automatic fret diagrams], page 258, [Chord mode overview], page 279, Section A.3 [Predefined fretboard diagrams], page 447.

Installed Files: ‘ly/predefined-guitar-fretboards.ly’, ‘ly/predefined-guitar-ninth-fretboards.ly’.

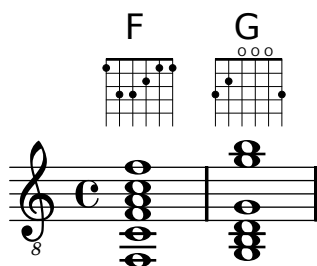
Snippets: Section “Fretted strings” in *Snippets*.

Internals Reference: Section “fret-diagram-interface” in *Internals Reference*.

Automatic fret diagrams

Fret diagrams can be automatically created from entered notes using the `FretBoards` context. If no predefined diagram is available for the entered notes in the active `stringTunings`, this context calculates strings and frets that can be used to play the notes.

```
<<
\context ChordNames {
  \chordmode {
    f1 g
  }
}
\context FretBoards {
  < f, c f a c' f'>1
  < g,\6 b, d g b g'>
}
\context Staff {
  \clef "treble_8"
  < f, c f a c' f'>1
  < g, b, d g b' g'>
}
>>
```



As no predefined diagrams are loaded by default, automatic calculation of fret diagrams is the default behavior. Once default diagrams are loaded, automatic calculation can be enabled and disabled with predefined commands:

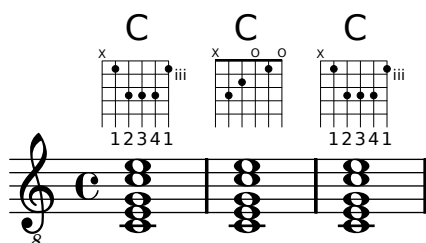
```
\storePredefinedDiagram <c e g c' e'>
                        #guitar-tuning
                        #"x;3-1-(;5-2;5-3;5-4;3-1-1);"

<<
\context ChordNames {
  \chordmode {
    c1 c c
  }
}
>>
```

```

\context FretBoards {
  <c e g c' e'>1
  \predefinedFretboardsOff
  <c e g c' e'>
  \predefinedFretboardsOn
  <c e g c' e'>
}
\context Staff {
  \clef "treble_8"
  <c e g c' e'>1
  <c e g c' e'>
  <c e g c' e'>
}
}
>>

```



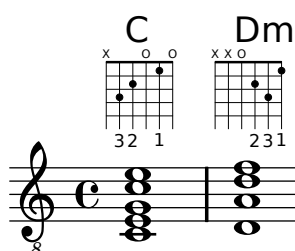
Sometimes the fretboard calculator will be unable to find an acceptable diagram. This can often be remedied by manually assigning a note to a string. In many cases, only one note need be manually placed on a string; the rest of the notes will then be placed appropriately by the `FretBoards` context.

Fingerings can be added to FretBoard fret diagrams.

```

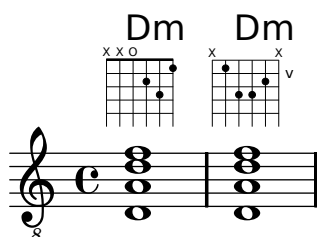
<<
\context ChordNames {
  \chordmode {
    c1 d:m
  }
}
\context FretBoards {
  < c-3 e-2 g c'-1 e' > 1
  < d a-2 d'-3 f'-1>
}
\context Staff {
  \clef "treble_8"
  < c e g c' e' > 1
  < d a d' f'>
}
}
>>

```



The minimum fret to be used in calculating strings and frets for the `FretBoard` context can be set with the `minimumFret` property.

```
<<
\context ChordNames {
  \chordmode {
    d1:m d:m
  }
}
\context FretBoards {
  < d a d' f' >
  \set FretBoards.minimumFret = #5
  < d a d' f' >
}
\context Staff {
  \clef "treble_8"
  < d a d' f' >
  < d a d' f' >
}
>>
```



The strings and frets for the `FretBoards` context depend on the `stringTunings` property, which has the same meaning as in the `TabStaff` context. See [\[Custom tablatures\]](#), page 239 for information on the `stringTunings` property.

The graphical layout of a fret diagram can be customized according to user preference through the properties of the `fret-diagram-interface`. Details are found at [Section “fret-diagram-interface” in *Internals Reference*](#). For a `FretBoards` fret diagram, the interface properties belong to `FretBoards.FretBoard`.

Predefined commands

`\predefinedFretboardsOff`, `\predefinedFretboardsOn`.

See also

Notation Reference: [\[Custom tablatures\]](#), page 239.

Snippets: [Section “Fretted strings” in *Snippets*](#).

Internals Reference: [Section “fret-diagram-interface” in *Internals Reference*](#).

Known issues and warnings

Automatic fretboard calculations do not work properly for instruments with non-monotonic tunings.

Right-hand fingerings

Right-hand fingerings *p-i-m-a* must be entered within a chord construct `<>` for them to be printed in the score, even when applied to a single note.

Note: There **must** be a hyphen before `\rightHandFinger` and a space before the closing `>`.

```
\clef "treble_8"
<c-\rightHandFinger #1 >4
<e-\rightHandFinger #2 >
<g-\rightHandFinger #3 >
<c-\rightHandFinger #4 >
<c,-\rightHandFinger #1 e-\rightHandFinger #2
  g-\rightHandFinger #3 c-\rightHandFinger #4 >1
```



For convenience, you can abbreviate `\rightHandFinger` to something short, for example `RH`,
`#(define RH rightHandFinger)`

Selected Snippets

Placement of right-hand fingerings

It is possible to exercise greater control over the placement of right-hand fingerings by setting a specific property, as demonstrated in the following example. Note: you must use a chord construct

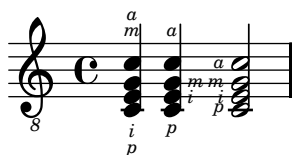
```
#(define RH rightHandFinger)

\relative c {
  \clef "treble_8"

  \set strokeFingerOrientations = #'(up down)
  <c-\RH #1 e-\RH #2 g-\RH #3 c-\RH #4 >4

  \set strokeFingerOrientations = #'(up right down)
  <c-\RH #1 e-\RH #2 g-\RH #3 c-\RH #4 >4

  \set strokeFingerOrientations = #'(left)
  <c-\RH #1 e-\RH #2 g-\RH #3 c-\RH #4 >2
}
```

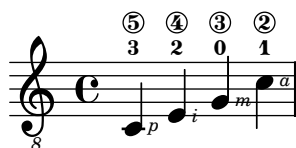


Fingerings string indications and right-hand fingerings

This example combines left-hand fingering, string indications, and right-hand fingering.
`#(define RH rightHandFinger)`

```
\relative c {
  \clef "treble_8"
  <c-3\5-\RH #1 >4
  <e-2\4-\RH #2 >4
```

```
<g-0\3-\RH #3 >4
<c-1\2-\RH #4 >4
}
```



See also

Snippets: [Section “Fretted strings” in *Snippets*](#).

Internals Reference: [Section “StrokeFinger” in *Internals Reference*](#).

2.4.2 Guitar

Most of the notational issues associated with guitar music are covered sufficiently in the general fretted strings section, but there are a few more worth covering here. Occasionally users want to create songbook-type documents having only lyrics with chord indications above them. Since LilyPond is a music typesetter, it is not recommended for documents that have no music notation in them. A better alternative is a word processor, text editor, or, for experienced users, a typesetter like GuitarTeX.

Indicating position and barring

This example demonstrates how to include guitar position and barring indications.

```
\clef "treble_8"
b16 d g b e
\textSpannerDown
\override TextSpanner #'(bound-details left text) = #"XII "
g16\startTextSpan
b16 e g e b g\stopTextSpan
e16 b g d
```



See also

Notation Reference: [\[Text spanners\]](#), page 177.

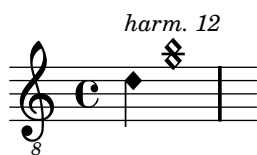
Snippets: [Section “Fretted strings” in *Snippets*](#), [Section “Expressive marks” in *Snippets*](#).

Indicating harmonics and dampened notes

Special note heads can be used to indicate dampened notes or harmonics. Harmonics are normally further explained with a text markup.

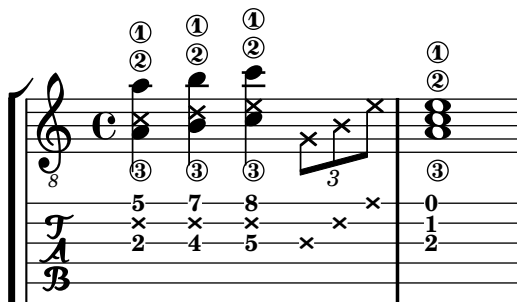
```
\relative c' {
\clef "treble_8"
\override Staff.NoteHead #'style = #'harmonic-mixed
d~\markup { \italic { \fontsize #-2 { "harm. 12" }}} <g b>1
```

}



Dampened notes (also called *dead notes*) are supported within normal and tablature staves:

```
music = \relative c' {
  < a\3 \deadNote c\2 a'\1 >4
  < b\3 \deadNote d\2 b'\1 >
  < c\3 \deadNote e\2 c'\1 >
  \deadNotesOn
  \times 2/3 { g8 b e }
  \deadNotesOff
  < a,\3 c\2 e\1 >1
}
\new StaffGroup <<
  \new Staff {
    \clef "treble_8"
    \music
  }
  \new TabStaff {
    \music
  }
>>
```



Another playing technique (especially used on electric guitars) is called *palm mute*. The string is hereby partly muted by the palm of the striking hand (hence the name). Lilypond supports the notation of palm mute-style notes by changing the note head to a triangle shape.

```
\new Voice { % Warning: explicit Voice instantiation is
              % required to have palmMuteOff work properly
              % when palmMuteOn comes at the beginning of
              % the piece.
  \relative c, {
    \clef "G_8"
    \palmMuteOn
    e8~\markup { \musicglyph #"noteheads.u2do" = palm mute }
    < e b' e > e
    \palmMuteOff
    e e \palmMute e e e |
    e8 \palmMute { e e e } e e e e |
```

```

    < \palmMute e b' e >8 \palmMute { e e e } < \palmMute e b' e >2
  }
}

```



See also

Snippets: [Section “Fretted strings” in *Snippets*](#).

Notation Reference: [\[Special note heads\]](#), page 29, [Section A.7 \[Note head styles\]](#), page 473.

2.4.3 Banjo

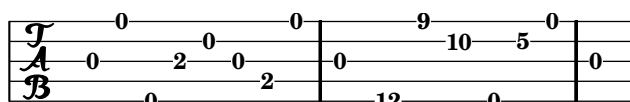
Banjo tablatures

LilyPond has basic support for the five-string banjo. When making tablatures for five-string 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

Snippets: [Section “Fretted strings” in *Snippets*](#).

Installed Files: ‘`scm/output-lib.scm`’ contains predefined banjo tunings.

2.5 Percussion

2.5.1 Common notation for percussion

Rhythmic music is primarily used for percussion and drum notation, but it can also be used to show the rhythms of melodies.

References for percussion

TODO add more.

- Some percussion may be notated on a rhythmic staff; this is discussed in [Showing melody rhythms], page 60, and [Instantiating new staves], page 136.
- MIDI output is discussed in a separate section; please see Section 3.5.6 [Percussion in MIDI], page 356.

See also

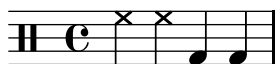
Notation Reference: [Showing melody rhythms], page 60, [Instantiating new staves], page 136. Section 3.5.6 [Percussion in MIDI], page 356.

Snippets: Section “Percussion” in *Snippets*.

Basic percussion notation

Percussion notes may be entered in `\drummode` mode, which is similar to the standard mode for entering notes. The simplest way to enter percussion notes is to use the `\drums` command, which creates the correct context and entry mode for percussion:

```
\drums {
  hihat4 hh bassdrum bd
}
```



This is shorthand for:

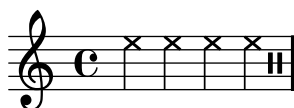
```
\new DrumStaff {
  \drummode {
    hihat4 hh bassdrum bd
  }
}
```



Each piece of percussion has a full name and an abbreviated name, and both can be used in input files. The full list of percussion note names may be found in Section A.11 [Percussion notes], page 515.

Note that the normal notation of pitches (such as `cis4`) in a `DrumStaff` context will cause an error message. Percussion clefs are added automatically to a `DrumStaff` context but they can also be set explicitly. Other clefs may be used as well.

```
\drums {
  \clef treble
  hh4 hh hh hh
  \break
  \clef percussion
  bd4 bd bd bd
}
```



2



There are a few issues concerning MIDI support for percussion instruments; for details please see [Section 3.5.6 \[Percussion in MIDI\]](#), page 356.

See also

Notation Reference: [Section 3.5.6 \[Percussion in MIDI\]](#), page 356, [Section A.11 \[Percussion notes\]](#), page 515.

File: 'ly/drumpitch-init.ly'

Snippets: [Section "Percussion" in Snippets](#).

Drum rolls

Drum rolls are indicated with three slashes across the stem. For quarter notes or longer the three slashes are shown explicitly, eighth notes are shown with two slashes (the beam being the third), and drum rolls shorter than eighths have one stem slash to supplement the beams. This is achieved with the tremolo notation, :32, as described in [\[Tremolo repeats\]](#), page 119. Here is an example of some snare rolls:

```
\drums {
  \time 2/4
  sn16 sn8 sn16 sn8 sn8:32 ~
  sn8 sn8 sn4:32 ~
  sn4 sn8 sn16 sn16
  sn4 r4
}
```



Sticking can be indicated by placing ^"R" or ^"L" after the note. The `staff-padding` property may be overridden to achieve a pleasing baseline.

```
\drums {
  \repeat unfold 2 {
    sn16 ^"L" sn^"R" sn^"L" sn^"L" sn^"R" sn^"L" sn^"R" sn^"R"
  }
}
```



See also

Snippets: [Section "Percussion" in Snippets](#).

Pitched percussion

Certain pitched percussion instruments (e.g. xylophone, vibraphone, and timpani) are written using normal staves. This is covered in other sections of the manual.

See also

Notation Reference: [Section 3.5.6 \[Percussion in MIDI\]](#), page 356.

Snippets: [Section "Percussion" in Snippets](#).

Percussion staves

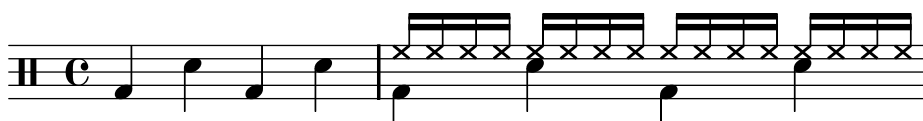
A percussion part for more than one instrument typically uses a multiline staff where each position in the staff refers to one piece of percussion. To typeset the music, the notes must be interpreted in `DrumStaff` and `DrumVoice` context.

```
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 [Section “I’m hearing Voices” in *Learning Manual*](#), can also be used. For example,

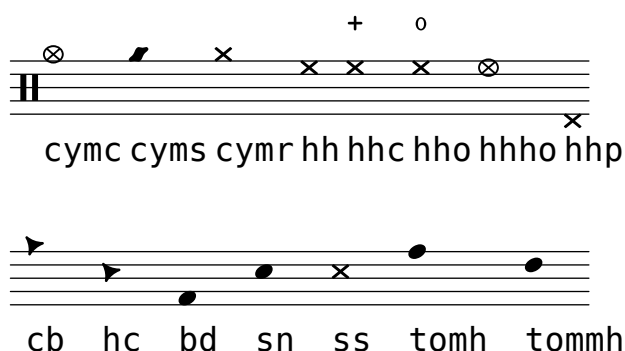
```
\new DrumStaff <<
  \drummode {
    bd4 sn4 bd4 sn4
    << {
      \repeat unfold 16 hh16
    } \\ {
      bd4 sn4 bd4 sn4
    } >>
  }
>>
```

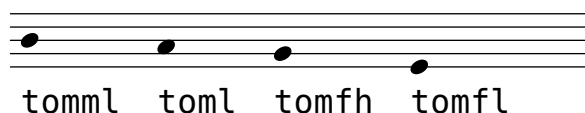


There are also other layout possibilities. To use these, set the property `drumStyleTable` in context `DrumVoice`. The following variables have been predefined:

`drums-style`

This is the default. It typesets a typical drum kit on a five-line staff:

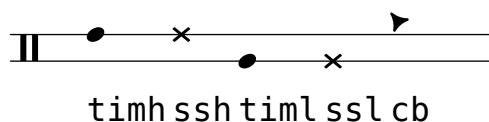




The drum scheme supports six different toms. When there are fewer toms, simply select the toms that produce the desired result. For example, to get toms on the three middle lines you use `tommh`, `tomm1`, and `tomfh`.

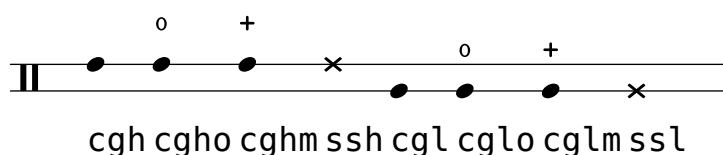
timbales-style

This typesets timbales on a two line staff:



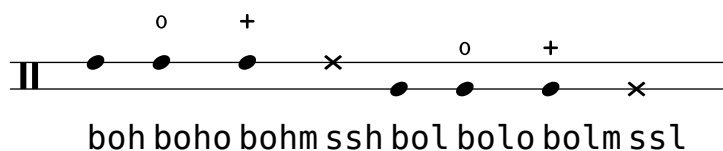
congas-style

This typesets congas on a two line staff:



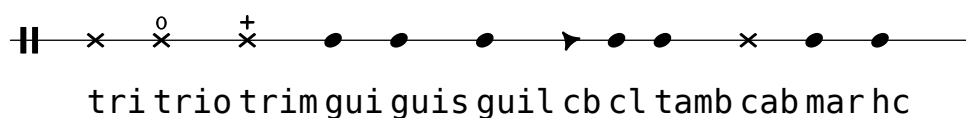
bongos-style

This typesets bongos on a two line staff:



percussion-style

To typeset all kinds of simple percussion on one line staves:



Custom percussion staves

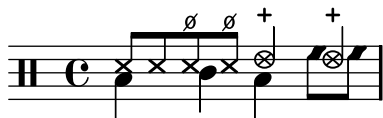
If you do not like any of the predefined lists you can define your own list at the top of your file.

```
#(define mydrums '(
  (bassdrum      default  #f      -1)
  (snare         default  #f      0)
  (hihat         cross    #f      1)
  (halfopenhihat cross    "halfopen" 1)
  (pedalhihat    xcircle  "stopped" 2)
  (lowtom        diamond  #f      3)))

up = \drummode { hh8 hh hhho hhho 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 }
>>
```



Selected Snippets

Here are some examples:

Two Woodblocks, entered with wbh (high woodblock) and wbl (low woodblock)

```
% These lines define the position of the woodblocks in the stave;
% if you like, you can change it or you can use special note heads
% for the woodblocks.
#(define mydrums '((hiwoodblock default #t 3)
                    (lowwoodblock default #t -2)))

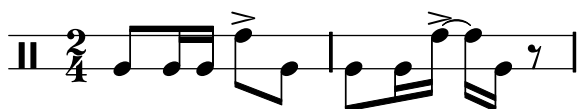
woodstaff = {
  % This defines a staff with only two lines.
  % It also defines the positions of the two lines.
  \override Staff.StaffSymbol #'line-positions = #'(-2 3)

  % This is neccessary; if not entered, the barline would be too short!
  \override Staff.BarLine #'bar-size = #3
}

\new DrumStaff {
  \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)

  % with this you load your new drum style table
  \woodstaff

  \drummode {
    \time 2/4
    wbl8 wbl16 wbl wbh8-> wbl |
    wbl8 wbl16 wbh-> ~ wbh wbl16 r8 |
  }
}
```



Note that in this special case the length of the barline must altered with `\override Staff.BarLine #'bar-size #number`. Otherwise it would be too short. And you have also to define the positions of the two stafflines. For more information about these delicate things have a look at [\[Staff symbol\]](#), page 142.

A tambourine, entered with ‘tamb’:

```
#(define mydrums '((tambourine default #t 0)))

tambustaff = {
```

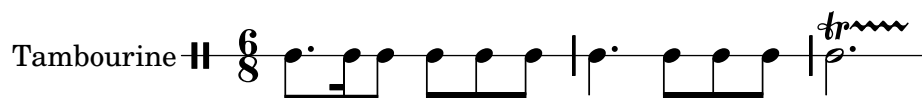
```

\override Staff.StaffSymbol #'line-positions = #'( 0 )
\override Staff.BarLine #'bar-size = #3
\set DrumStaff.instrumentName = #"Tambourine"
}

\new DrumStaff {
  \tambustaff
  \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)

  \drummode {
    \time 6/8
    tamb8. tamb16 tamb8 tamb tamb tamb |
    tamb4. tamb8 tamb tamb |
    % the trick with the scaled duration and the shorter rest
    % is necessary for the correct ending of the trill-span!
    tamb2.*5/6 \startTrillSpan s8 \stopTrillSpan |
  }
}

```



Music for Tam-Tam (entered with 'tt'):

```

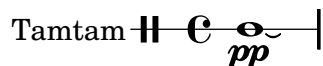
#(define mydrums '((tamtam default #t 0)))

tamtamstaff = {
  \override Staff.StaffSymbol #'line-positions = #'( 0 )
  \override Staff.BarLine #'bar-size = #3
  \set DrumStaff.instrumentName = #"Tamtam"
}

\new DrumStaff {
  \tamtamstaff
  \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)

  \drummode {
    tt 1 \pp \laissezVibrer
  }
}

```



Two different bells, entered with 'cb' (cowbell) and 'rb' (ridebell)

```

#(define mydrums '((ridebell default #t 3)
                  (cowbell default #t -2)))

bellstaff = {
  \override DrumStaff.StaffSymbol #'line-positions = #'(-2 3)
  \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)
  \override Staff.BarLine #'bar-size = #3
}

```

```

\set DrumStaff.instrumentName = #"Different Bells"
}

\new DrumStaff {
  \bellstaff
  \drummode {
    \time 2/4
    rb8 rb cb cb16 rb-> ~ |
    rb16 rb8 rb16 cb8 cb |
  }
}

```



Here a short example taken from Stravinsky's 'L'histoire du Soldat'.

```

#(define mydrums '((bassdrum default #t 4)
                   (snare default #t -4)
                   (tambourine default #t 0)))

global = {
  \time 3/8 s4.
  \time 2/4 s2*2
  \time 3/8 s4.
  \time 2/4 s2
}

drumsA = {
  \context DrumVoice <<
    { \global }
    { \drummode {
      \autoBeamOff
      \stemDown sn8 \stemUp tamb s8 |
      sn4 \stemDown sn4 |
      \stemUp tamb8 \stemDown sn8 \stemUp sn16 \stemDown sn \stemUp sn8 |
      \stemDown sn8 \stemUp tamb s8 |
      \stemUp sn4 s8 \stemUp tamb
    }
  }
  >>
}

drumsB = {
  \drummode {
    s4 bd8 s2*2 s4 bd8 s4 bd8 s8
  }
}

\layout {
  indent = #40
}

```

```

\score {
  \new StaffGroup <<
    \new DrumStaff {
      \set DrumStaff.instrumentName = \markup {
        \column {
          "Tambourine"
          "et"
          "caisse claire s. timbre"
        }
      }
      \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)
      \drumsA
    }

    \new DrumStaff {
      \set DrumStaff.instrumentName = #"Grosse Caisse"
      \set DrumStaff.drumStyleTable = #(alist->hash-table mydrums)
      \drumsB }
  >>
}

```

Tambourine
et
caisse claire s. timbre

Grosse Caisse



See also

Snippets: [Section “Percussion” in Snippets](#).

Internals Reference: [Section “DrumStaff” in Internals Reference](#), [Section “DrumVoice” in Internals Reference](#).

Ghost notes

Ghost notes for drums and percussion may be created using the `\parenthesize` command detailed in [\[Parentheses\]](#), [page 170](#). However, the default `\drummode` does not include the `Parenthesis_engraver` plugin which allows this.

```

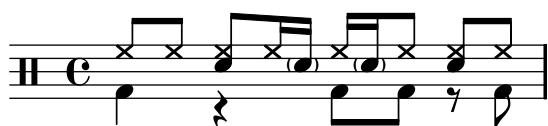
\new DrumStaff \with {
  \consists "Parenthesis_engraver"
}
<<
\context DrumVoice = "1" { s1 }
\context DrumVoice = "2" { s1 }
\drummode {
  <<
    {
      hh8[ hh] <hh sn> hh16
      < \parenthesize sn > hh
      < \parenthesize sn > hh8 <hh sn> hh
    } \
  }

```

```

{
  bd4 r4 bd8 bd r8 bd
}
>>
}
>>

```



Also note that you must add chords (< > brackets) around each `\parenthesize` statement.

See also

Snippets: [Section “Percussion” in *Snippets*](#).

2.6 Wind instruments

Moderato assai

This section includes some elements of music notation that arise when writing for winds.

2.6.1 Common notation for wind instruments

This section discusses some issues common to most wind instruments.

References for wind instruments

Many notation issues for wind instruments pertain to breathing and tonguing:

- Breathing can be specified by rests or [\[Breath marks\]](#), page 102.
- Legato playing is indicated by [\[Slurs\]](#), page 99.
- Different types of tonguings, ranging from legato to non-legato to staccato are usually shown by articulation marks, sometimes combined with slurs, see [\[Articulations and ornamentations\]](#), page 90 and [Section A.10 \[List of articulations\]](#), page 513.
- Flutter tonguing is usually indicated by placing a tremolo mark and a text markup on the note. See [\[Tremolo repeats\]](#), page 119.

There are also other aspects of musical notation that can apply to wind instruments:

- Many wind instruments are transposing instruments, see [\[Instrument transpositions\]](#), page 19.
- The slide glissando are characteristic of the trombone, but other winds may perform keyed or valved glissandi. See [\[Glissando\]](#), page 104.
- Harmonic series glissandi, which are possible on all brass instruments but common for French Horns, are usually written out as [\[Grace notes\]](#), page 84.

- Pitch inflections at the end of a note are discussed in [\[Falls and doits\]](#), page 104.
- Key slaps or valve slaps are often shown by the `cross` style of [\[Special note heads\]](#), page 29.
- Woodwinds can overblow low notes to sound harmonics. These are shown by the `flageolet` articulation. See [Section A.10 \[List of articulations\]](#), page 513.
- The use of brass mutes is usually indicated by a text markup, but where there are many rapid changes it is better to use the `stopped` and `open` articulations. See [\[Articulations and ornamentations\]](#), page 90 and [Section A.10 \[List of articulations\]](#), page 513.
- Stopped horns are indicated by the `stopped` articulation. See [\[Articulations and ornamentations\]](#), page 90.

Selected Snippets

Changing \flageolet mark size

To make the `\flageolet` circle smaller use the following Scheme function.

```
smallFlageolet =
#(let ((m (make-articulation "flageolet")))
  (set! (ly:music-property m 'tweaks)
    (acons 'font-size -3
      (ly:music-property m 'tweaks)))
  m)

\layout { ragged-right = ##f }

\relative c'' {
  d4~\flageolet_\markup { default size } d_\flageolet
  c4~\smallFlageolet_\markup { smaller } c_\smallFlageolet
}
```



See also

Notation Reference: [\[Breath marks\]](#), page 102, [\[Slurs\]](#), page 99, [\[Articulations and ornamentations\]](#), page 90, [Section A.10 \[List of articulations\]](#), page 513, [\[Tremolo repeats\]](#), page 119, [\[Instrument transpositions\]](#), page 19, [\[Glissando\]](#), page 104, [\[Grace notes\]](#), page 84, [\[Falls and doits\]](#), page 104, [\[Special note heads\]](#), page 29,

Snippets: [Section “Winds” in Snippets](#).

Fingerings

All wind instruments other than the trombone require the use of several fingers to produce each pitch.

Selected Snippets

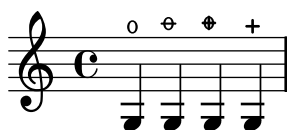
Fingering symbols for wind instruments

Special symbols can be achieved by combining existing glyphs, which is useful for wind instruments.

```

centermarkup = {
  \once \override TextScript #'self-alignment-X = #CENTER
  \once \override TextScript #'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))))
}
\score
{\relative c'
  {
    g\open
    \once \override TextScript #'staff-padding = #-1.0 \centermarkup
    g^\markup{\combine \musicglyph #"scripts.open" \musicglyph
      #"scripts.tenuto"}
    \centermarkup g^\markup{\combine \musicglyph #"scripts.open"
      \musicglyph #"scripts.stopped"}
    g\stopped
  }
}

```



Recorder fingering chart

The following example demonstrates how fingering charts for wind instruments can be realized.

% range chart for paetzold contrabass recorder

```

centermarkup = {
  \once \override TextScript #'self-alignment-X = #CENTER
  \once \override TextScript #'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))))
}

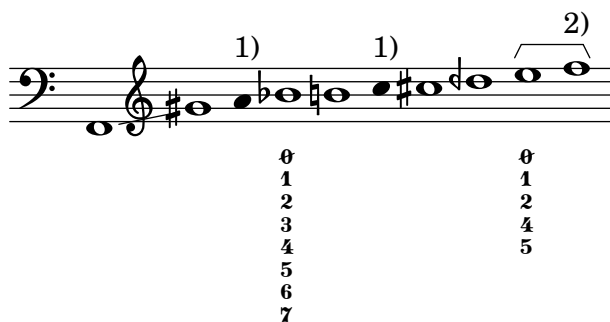
\score {
  \new Staff \with {
    \remove "Time_signature_engraver"
    \override Stem #'stencil = ##f
    \consists "Horizontal_bracket_engraver"
  }
  {
    \clef bass
    \set Score.timing = ##f
  }
}

```

```

f,1*1/4 \glissando
\clef violin
gis'1*1/4
\stemDown a'4^\markup{1)}
\centermarkup
\once \override TextScript #'padding = #2
bes'1*1/4_\markup{\override #'(baseline-skip . 1.7) \column
  { \fontsize #-5 \slashed-digit #0 \finger 1 \finger 2 \finger 3 \finger 4
    \finger 5 \finger 6 \finger 7} }
b'1*1/4
c''4^\markup{1)}
\centermarkup
\once \override TextScript #'padding = #2
cis''1*1/4
deh''1*1/4
\centermarkup
\once \override TextScript #'padding = #2
\once \override Staff.HorizontalBracket #'direction = #UP
e''1*1/4_\markup{\override #'(baseline-skip . 1.7) \column
  { \fontsize #-5 \slashed-digit #0 \finger 1 \finger 2 \finger 4
    \finger 5} }\startGroup
f''1*1/4^\markup{2)}\stopGroup
}
}

```



See also

Snippets: [Section “Winds”](#) in *Snippets*.

2.6.2 Bagpipes

This section includes extra information for writing for bagpipes.

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 grace notes 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.

See also

Section “Winds” in *Snippets*.

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 \hdbl 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
\thrw d2.
\slurd d2
\bar "|."
}

```

Amazing Grace

Hymn

Trad. arr.



See also

Section “Winds” in *Snippets*.

2.7 Chord notation

F C F F C F F B \flat F C⁷ F C

1. Fair is the sun - shine, Fair - er the moon - light And all the stars_in heav'n a - bove;
2. Fair are the mead - ows, Fair - er the wood - land, Robed in the flowers of blooming spring;

Chords can be entered either as normal notes or in chord mode and displayed using a variety of traditional European chord naming conventions. Chord names and figured bass notation can also be displayed.

2.7.1 Chord mode

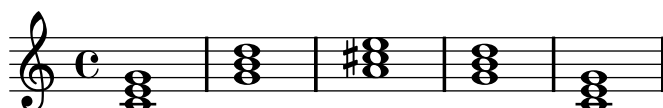
Chord mode is used to enter chords using an indicator of the chord structure, rather than the chord pitches.

Chord mode overview

Chords can be entered as simultaneous music, as discussed in [\[Chorded notes\]](#), page 121.

Chords can also be entered in “chord mode”, which is an input mode that focuses on the structures of chords in traditional European music, rather than on specific pitches. This is convenient for those who are familiar with using chord names to describe chords. More information on different input modes can be found at [Section 5.4.1 \[Input modes\]](#), page 418.

```
\chordmode { c1 g a g c }
```



Chords entered using chord mode are music elements, and can be transposed just like chords entered using simultaneous music. `\chordmode` is absolute, as `\relative` has no effect on `chordmode` blocks. However, in `\chordmode` the absolute pitches are one octave higher than in note mode.

Chord mode and note mode can be mixed in sequential music:

```
<c e g>2 <g b d>
\chordmode { c2 f }
<c e g>2 <g' b d>
\chordmode { f2 g }
```



See also

Music Glossary: [Section “chord” in Music Glossary](#).

Notation Reference: [\[Chorded notes\]](#), page 121, [Section 5.4.1 \[Input modes\]](#), page 418.

Snippets: [Section “Chords” in Snippets](#).

Known issues and warnings

When chord mode and note mode are mixed in sequential music, and chord mode comes first, the note mode will create a new `Staff` context.

```
\chordmode { c2 f }
<c e g>2 <g' b d>
```



To avoid this behavior, explicitly create the `Staff` context:

```
\new Staff {
  \chordmode { c2 f }
  <c e g>2 <g' b d>
}
```



Common chords

Major triads are entered by including the root and an optional duration:

```
\chordmode { c2 f4 g }
```



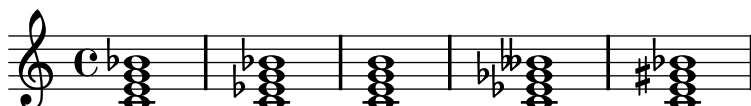
Minor, augmented, and diminished triads are entered by placing : and a quality modifier string after the duration:

```
\chordmode { c2:m f4:aug g:dim }
```



Seventh chords can be created:

```
\chordmode { c1:7 c:m7 c:maj7 c:dim7 c:aug7 }
```



The table belows shows the actions of the quality modifiers on triads and seventh chords. The default seventh step added to chords is a minor or flatted seventh, which makes the dominant seventh the basic seventh chord. All alterations are relative to the dominant seventh. A more complete table of modifier usage is found at [Section A.2 \[Common chord modifiers\]](#), page 444.

Modifier	Action	Example
None	The default action; produces a major triad.	
m, m7	The minor chord. This modifier lowers the 3rd.	

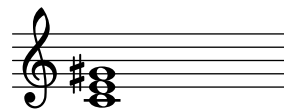
dim, dim7

The diminished chord. This modifier lowers the 3rd, 5th and (if present) the 7th step.



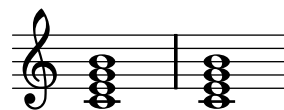
aug

The augmented chord. This modifier raises the 5th step.



maj, maj7

The major 7th chord. This modifier adds a raised 7th step. The 7 following maj is optional. Do NOT use this modifier to create a major triad.



See also

Notation Reference: [Section A.2 \[Common chord modifiers\]](#), page 444, [\[Extended and altered chords\]](#), page 281.

Snippets: [Section “Chords” in *Snippets*](#).

Known issues and warnings

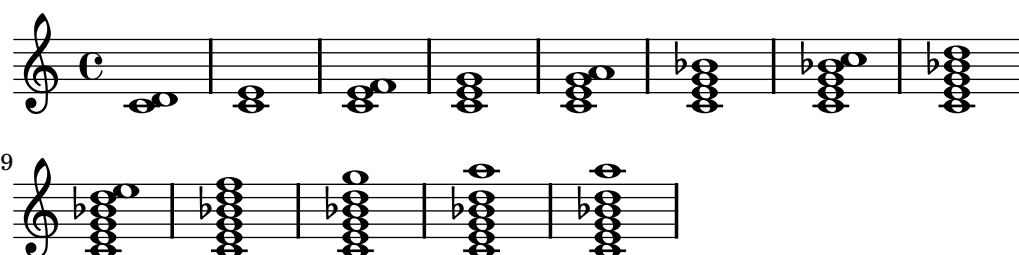
Only one quality modifier should be used per chord, typically on the highest step present in the chord. Chords with more than quality modifier will be parsed without an error or warning, but the results are unpredictable. Chords that cannot be achieved with a single quality modifier should be altered by individual pitches, as described in [\[Extended and altered chords\]](#), page 281.

Extended and altered chords

Chord structures of arbitrary complexity can be created in chord mode. The modifier string can be used to extend a chord, add or remove chord steps, raise or lower chord steps, and add a bass note or create an inversion.

The first number following the `:` is taken to be the extent of the chord. The chord is constructed by sequentially adding thirds to the root until the specified number has been reached. Note that the seventh step added as part of an extended chord will be the minor or flatted seventh, not the major seventh. If the extent is not a third (e.g., 6), thirds are added up to the highest third below the extent, and then the step of the extent is added. The largest possible value for the extent is 13. Any larger value is interpreted as 13.

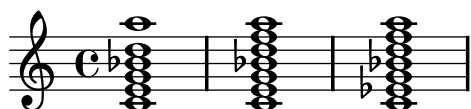
```
\chordmode {
  c1:2 c:3 c:4 c:5
  c1:6 c:7 c:8 c:9
  c1:10 c:11 c:12 c:13
  c1:14
}
```



Note that both `c:5` and `c` produce a C major triad.

Since an unaltered 11 does not sound good when combined with an unaltered 13, the 11 is removed from a `:13` chord (unless it is added explicitly).

```
\chordmode {
  c1:13 c:13.11 c:m13
}
```



Individual steps can be added to a chord. Additions follow the extent and are prefixed by a dot (.). The basic seventh step added to a chord is the minor or flatted seventh, rather than the major seventh.

```
\chordmode {
  c1:5.6 c:3.7.8 c:3.6.13
}
```



Added steps can be as high as desired.

```
\chordmode {
  c4:5.15 c:5.20 c:5.25 c:5.30
}
```



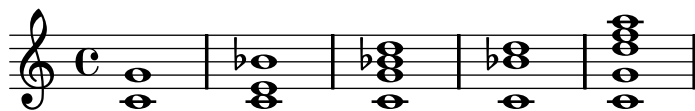
Added chord steps can be altered by suffixing a - or + sign to the number. To alter a step that is automatically included as part of the basic chord structure, add it as an altered step.

```
\chordmode {
  c1:7+ c:5+.3- c:3-.5-.7-
}
```



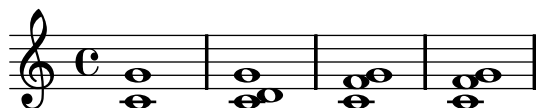
Following any steps to be added, a series of steps to be removed is introduced in a modifier string with a prefix of ^. If more than one step is to be removed, the steps to be removed are separated by . following the initial ^.

```
\chordmode {
  c1^3 c:7^5 c:9^3 c:9^3.5 c:13.11^3.7
}
```



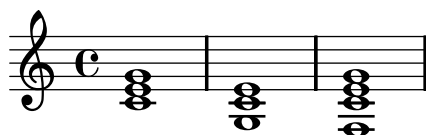
The modifier `sus` can be added to the modifier string to create suspended chords. This removes the 3rd step from the chord. Append either 2 or 4 to add the 2nd or 4th step to the chord. `sus` is equivalent to `^3`; `sus4` is equivalent to `.4^3`.

```
\chordmode {
  c1:sus c:sus2 c:sus4 c:5.4^3
}
```



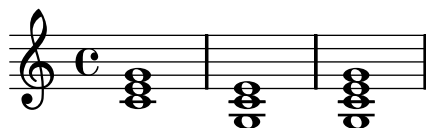
Inversions (putting a pitch other than the root on the bottom of the chord) and added bass notes can be specified by appending */pitch* to the chord.

```
\chordmode {
  c1 c/g c/f
}
```



A bass note that is part of the chord can be added, instead of moved as part of an inversion, by using */+pitch*.

```
\chordmode {
  c1 c/g c/+g
}
```



Chord modifiers that can be used to produce a variety of standard chords are shown in [Section A.2 \[Common chord modifiers\]](#), page 444.

See also

Notation Reference: [Section A.2 \[Common chord modifiers\]](#), page 444.

Snippets: [Section “Chords” in Snippets](#).

Known issues and warnings

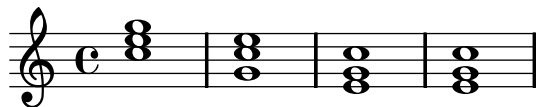
Each step can only be present in a chord once. The following simply produces the augmented chord, since `5+` is interpreted last.

```
\chordmode { c1:5.5-.5+ }
```



Only the second inversion can be created by adding a bass note. The first inversion requires changing the root of the chord.

```
\chordmode {
  c'1: c':/g e:6-3-~5 e:m6-~5
}
```



2.7.2 Displaying chords

Chords can be displayed by name, in addition to the standard display as notes on a staff.

Printing chord names

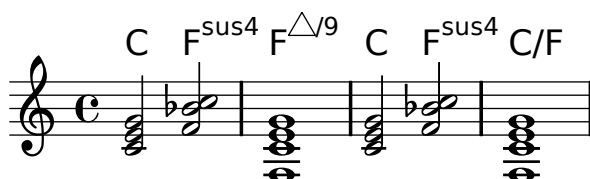
Chord names are printed in the `ChordNames` context:

```
\new ChordNames {
  \chordmode {
    c2 f4. g8
  }
}
```

C F G

Chords can be entered as simultaneous notes or through the use of chord mode. The displayed chord name will be the same, regardless of the mode of entry, unless there are inversions or added bass notes:

```
<<
  \new ChordNames {
    <c e g>2 <f bes c>
    <f c' e g>1
    \chordmode {
      c2 f:sus4 c1:/f
    }
  }
  {
    <c e g>2 <f bes c>
    <f, c' e g>1
    \chordmode {
      c2 f:sus4 c1:/f
    }
  }
>>
```



Rests passed to a `ChordNames` context will cause the `noChordSymbol` markup to be displayed.

```
<<
  \new ChordNames \chordmode {
    c1
```

```

r1
g1
c1
}
\new Score \chordmode {
  c1
  r1
  g1
  c1
}
>>

```



`\chords { ... }` is a shortcut notation for `\new ChordNames { \chordmode { ... } }`.

```

\chords {
  c2 f4.:m g8:maj7
}

```

C Fm G[△]

```

\new ChordNames {
  \chordmode {
    c2 f4.:m g8:maj7
  }
}

```

C Fm G[△]

Selected Snippets

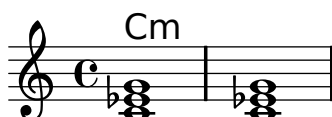
Showing chords at changes

Chord names can be displayed only at the start of lines and when the chord changes.

```

harmonies = \chordmode {
  c1:m c:m \break c:m c:m d
}
<<
\new ChordNames {
  \set chordChanges = ##t
  \harmonies
}
\new Staff {
  \relative c' { \harmonies }
}
>>

```





Simple lead sheet

When put together, chord names, a melody, and lyrics 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

Music Glossary: [Section “chord” in *Music Glossary*](#).

Notation Reference: [\[Writing music in parallel\]](#), page 133.

Snippets: [Section “Chords” in *Snippets*](#).

Internals Reference: [Section “ChordNames” in *Internals Reference*](#), [Section “ChordName” in *Internals Reference*](#), [Section “Chord_name_engraver” in *Internals Reference*](#), [Section “Volta_engraver” in *Internals Reference*](#), [Section “Bar_engraver” in *Internals Reference*](#).

Known issues and warnings

Chords containing inversions or altered bass notes are not named properly if entered using simultaneous music.


Customizing chord names

There is no unique system for naming chords. Different musical traditions use different names for the same set of chords. There are also different symbols displayed for a given chord name. The names and symbols displayed for chord names are customizable.

The basic chord name layout is a system for Jazz music, proposed by Klaus Ignatzek (see [Section “Literature list” in *Essay*](#)). The chord naming system can be modified as described below. An alternate jazz chord system has been developed using these modifications. The Ignatzek and alternate Jazz notation are shown on the chart in [Section A.1 \[Chord name chart\]](#), page 443.

In addition to the different naming systems, different note names are used for the root in different languages. The predefined variables `\germanChords`, `\semiGermanChords`, `\italianChords` and `\frenchChords` set these variables. The effect is demonstrated here:

default	E/D	Cm	B/B	B [#] /B [#]	B ^b /B ^b
german	E/d	Cm	H/h	H [#] /his	B/b
semi-german	E/d	Cm	H/h	H [#] /his	B ^b /b
italian	Mi/Re	Do m	Si/Si	Si [#] /Si [#]	Si ^b /Si ^b
french	Mi/Ré	Do m	Si/Si	Si [#] /Si [#]	Si ^b /Si ^b



German songbooks may indicate minor chords as lowercase letters, without any *m* suffix. This can be obtained by setting the `chordNameLowercaseMinor` property:

```
\chords {
  \set chordNameLowercaseMinor = ##t
  c2 d:m e:m f
}
```

C d e F

If none of the existing settings give the desired output, the chord name display can be tuned through the following properties.

`chordRootNamer`

The chord name is usually printed as a letter for the root 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.

`majorSevenSymbol`

This property contains the markup object used to follow the output of `chordRootNamer` to identify a major 7 chord. Predefined options are `whiteTriangleMarkup` and `blackTriangleMarkup`.

`chordNoteNamer`

When the chord name contains additional pitches other than the root (e.g., an added bass note), this function is used to print the additional pitch. By default the pitch is printed using `chordRootNamer`. The `chordNoteNamer` property can be set to a specialized function to change this behavior. For example, the bass note can be printed in lower case.

`chordNameSeparator`

Different parts of a chord name are normally separated by a slash. By setting `chordNameSeparator`, you can use any desired markup for a separator.

`chordNameExceptions`

This property is a list of pairs. The first item in each pair is a set of pitches used to identify the steps present in the chord. The second item is a markup that will follow the `chordRootNamer` output to create the chord name.

`chordPrefixSpacer`

The ‘m’ for minor chords is usually printed immediately to the right of the root of the chord. A spacer can be placed between the root and ‘m’ by setting `chordPrefixSpacer`. The spacer is not used when the root is altered.

Predefined commands

`\whiteTriangleMarkup`, `\blackTriangleMarkup`, `\germanChords`, `\semiGermanChords`, `\italianChords`, `\frenchChords`.

Selected Snippets

Chord name exceptions

The property `chordNameExceptions` can be used to store a list of special notations for specific chords.

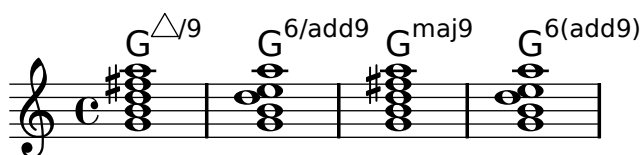
```
% modify maj9 and 6(add9)
% Exception music is chords with markups
chExceptionMusic = {
  <c e g b d'>1-\markup { \super "maj9" }
  <c e g a d'>1-\markup { \super "6(add9)" }
}

% Convert music to list and prepend to existing exceptions.
chExceptions = #( append
  ( sequential-music-to-chord-exceptions chExceptionMusic #t)
  ignatzekExceptions)

theMusic = \chordmode {
  g1:maj9 g1:6.9
  \set chordNameExceptions = #chExceptions
  g1:maj9 g1:6.9
}

\layout {
  ragged-right = ##t
}

<< \context ChordNames \theMusic
    \context Voice \theMusic
>>
```



chord name major7

The layout of the major 7 can be tuned with `majorSevenSymbol`.

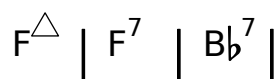
```
\chords {
  c:7+
  \set majorSevenSymbol = \markup { j7 }
  c:7+
}
```

$C^{\triangle}C^{j7}$

Adding bar lines to ChordNames context

To add bar line indications in the `ChordNames` context, add the `Bar_engraver`.

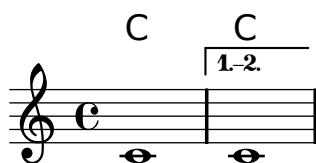
```
\new ChordNames \with {
  \override BarLine #'bar-size = #4
  \consists "Bar_engraver"
}
\chordmode {
  f1:maj7 f:7 bes:7
}
```



Volta below chords

By adding the `Volta_engraver` to the relevant staff, volte can be put under chords.

```
\score {
  <<
    \chords {
      c1
      c1
    }
    \new Staff \with {
      \consists "Volta_engraver"
    }
    {
      \repeat volta 2 { c'1 }
      \alternative { c' }
    }
  >>
  \layout {
    \context {
      \Score
      \remove "Volta_engraver"
    }
  }
}
```



Changing chord separator

The separator between different parts of a chord name can be set to any markup.

```
\chords {
  c:7sus4
  \set chordNameSeparator
    = \markup { \typewriter | }
  c:7sus4
}
```

$C^{7/sus4} C^7|sus4$

See also

Notation Reference: [Section A.1 \[Chord name chart\]](#), page 443, [Section A.2 \[Common chord modifiers\]](#), page 444.

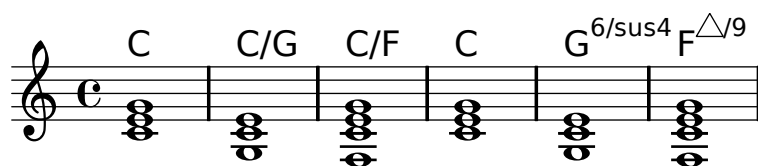
Installed Files: ‘scm/chords-ignatzek.scm’, ‘scm/chord-entry.scm’, ‘ly/chord-modifier-init.ly’.

Snippets: [Section “Chords” in Snippets](#).

Known issues and warnings

Chord names are determined from both the pitches that are present in the chord and the information on the chord structure that may have been entered in `\chordmode`. If the simultaneous pitches method of entering chords is used, undesired names result from inversions or bass notes.

```
myChords = \relative c' {
  \chordmode { c1 c/g c/f }
  <c e g>1 <g c e> <f c' e g>
}
<<
  \new ChordNames { \myChords }
  \new Staff { \myChords }
>>
```



2.7.3 Figured bass

Adagio.

Violino I.

Violino II.

Violone,
e Cembalo.

Figured bass notation can be displayed.

Introduction to figured bass

LilyPond has support for figured bass, also called thorough bass or basso continuo:

```
<<
\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`, that accepts entry of bass figures, and there is a context named `FiguredBass` that takes care of displaying `BassFigure` objects. Figured bass can also be displayed in `Staff` contexts.

`\figures{ ... }` is a shortcut notation for `\new FiguredBass { \figuremode { ... } }`.

Although the support for figured bass may superficially resemble chord support, it is much simpler. `\figuremode` mode simply stores the figures and the `FiguredBass` context prints them as entered. There is no conversion to pitches.

See also

Music Glossary: [Section “figured bass” in *Music Glossary*](#).

Snippets: [Section “Chords” in *Snippets*](#).

Entering figured bass

`\figuremode` is used to switch the input mode to figure mode. More information on different input modes can be found at [Section 5.4.1 \[Input modes\]](#), page 418.

In figure mode, a group of bass figures is delimited by `<` and `>`. The duration is entered after the `>`.

```
\new FiguredBass {
  \figuremode {
    <6 4>2
  }
}
```

6
4

Accidentals (including naturals) can be added to figures:

```
\figures {
  <7! 6+ 4-> <5++> <3-->
}
```

$\flat 7$ $\sharp 5$ $\flat 3$
 $\sharp 6$
 $\flat 4$

Augmented and diminished steps can be indicated:

```
\figures {
  <6\+ 5/> <7/>
}
```

$+6$ \sharp
 $\flat 5$

A backward slash through a figure (typically used for raised sixth steps) can be created:

```
\figures {
  <6> <6\\>
}
```

6 \flat

Vertical spaces and brackets can be included in figures:

```
\figures {
  <[12 _!] 8 [6 4]>
}
```

$\left[\begin{array}{c} 12 \\ \flat \\ 8 \\ \left[\begin{array}{c} 6 \\ 4 \end{array} \right] \end{array} \right]$

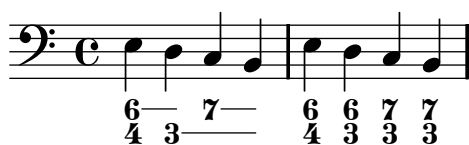
Any text markup can be inserted as a figure:

```
\figures {
  <\markup { \tiny \number 6 \super (1) } 5>
}
```

$6^{(1)}$
 $\flat 5$

Continuation lines can be used to indicate repeated figures:

```
<<
{
  \clef bass
  e4 d c b,
  e4 d c b,
}
\figures {
  \bassFigureExtendersOn
  <6 4>4 <6 3> <7 3> <7 3>
  \bassFigureExtendersOff
  <6 4>4 <6 3> <7 3> <7 3>
}
>>
```



In this case, the extender lines replace existing figures, unless the continuation lines have been explicitly terminated.

```
<<
\figures {
  \bassFigureExtendersOn
  <6 4>4 <6 4> <6\! 4\!> <6 4>
}
{
  \clef bass
  d4 d c c
}
>>
```



The table below summarizes the figure modifiers available.

Modifier	Purpose	Example
+, -, !	Accidentals	$\flat 7 \sharp 5 \flat 3$ $\sharp 6$ $\flat 4$
\+, /	Augmented and diminished steps	$+6$ \sharp
\\	Raised sixth step	$\mathbf{6}$
\!	End of continuation line	



Predefined commands

`\bassFigureExtendersOn`, `\bassFigureExtendersOff`.

Selected Snippets

Changing the positions of figured bass alterations

Accidentals and plus signs can appear before or after the numbers, depending on the `figuredBassAlterationDirection` and `figuredBassPlusDirection` properties.

```
\figures {
  <6\+> <5+> <6 4-> r
  \set figuredBassAlterationDirection = #RIGHT
  <6\+> <5+> <6 4-> r
}
```

```

\set figuredBassPlusDirection = #RIGHT
<6\+> <5+> <6 4-> r
\set figuredBassAlterationDirection = #LEFT
<6\+> <5+> <6 4-> r
}

```

+6 #5 6 **+6 5# 6** **6+ 5# 6** **6+ #5 6**
 $\flat 4$ $\flat 4$ $\flat 4$ $\flat 4$

See also

Snippets: [Section “Chords” in *Snippets*](#).

Internals Reference: [Section “BassFigure” in *Internals Reference*](#), [Section “BassFigureAlignment” in *Internals Reference*](#), [Section “BassFigureLine” in *Internals Reference*](#), [Section “BassFigureBracket” in *Internals Reference*](#), [Section “BassFigureContinuation” in *Internals Reference*](#), [Section “FiguredBass” in *Internals Reference*](#).

Displaying figured bass

Figured bass can be displayed using the `FiguredBass` context, or in most staff contexts.

When displayed in a `FiguredBass` context, the vertical location of the figures is independent of the notes on the staff.

```

<<
\relative c'' {
  c4 c'8 r8 c,4 c'
}
\new FiguredBass {
  \figuremode {
    <4>4 <10 6>8 s8
    <6 4>4 <6 4>
  }
}
>>

```



In the example above, the `FiguredBass` context must be explicitly instantiated to avoid creating a second (empty) staff.

Figured bass can also be added to `Staff` contexts directly. In this case, the vertical position of the figures is adjusted automatically.

```

<<
\new Staff = myStaff
\figuremode {
  <4>4 <10 6>8 s8
  <6 4>4 <6 4>
}
%% Put notes on same Staff as figures
\context Staff = myStaff

```

```

{
  \clef bass
  c4 c'8 r8 c4 c'
}
>>

```

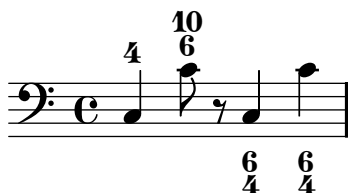


When added in a `Staff` context, figured bass can be displayed above or below the staff.

```

<<
  \new Staff = myStaff
  \figuremode {
    <4>4 <10 6>8 s8
    \bassFigureStaffAlignmentDown
    <6 4>4 <6 4>
  }
  %% Put notes on same Staff as figures
  \context Staff = myStaff
  {
    \clef bass
    c4 c'8 r8 c4 c'
  }
>>

```



Predefined commands

`\bassFigureStaffAlignmentDown`,
`\bassFigureStaffAlignmentNeutral`.

`\bassFigureStaffAlignmentUp`,

See also

Snippets: [Section “Chords” in *Snippets*](#).

Internals Reference: [Section “BassFigure” in *Internals Reference*](#), [Section “BassFigureAlignment” in *Internals Reference*](#), [Section “BassFigureLine” in *Internals Reference*](#), [Section “BassFigureBracket” in *Internals Reference*](#), [Section “BassFigureContinuation” in *Internals Reference*](#), [Section “FiguredBass” in *Internals Reference*](#).

Known issues and warnings

To ensure that continuation lines work properly, it is safest to use the same rhythm in the figure line as in the bass line.

```

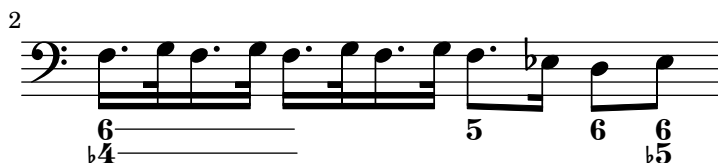
<<
{
  \clef bass

```

```

\repeat unfold 4 { f16. g32 } f8. es16 d8 es
}
\figures {
  \bassFigureExtendersOn
  % The extenders are correct here, with the same rhythm as the bass
  \repeat unfold 4 { <6 4->16. <6 4->32 }
  <5>8. r16 <6>8 <6\! 5->
}
>>
<<
{
  \clef bass
  \repeat unfold 4 { f16. g32 } f8. es16 d8 es
}
\figures {
  \bassFigureExtendersOn
  % The extenders are incorrect here, even though the timing is the same
  <6 4->4 <6 4->4
  <5>8. r16 <6>8 <6\! 5->
}
>>

```

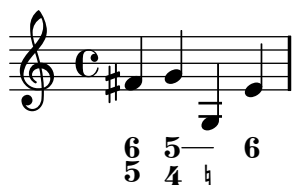


When using extender lines, adjacent figures with the same number in a different figure location can cause the figure positions to invert.

```

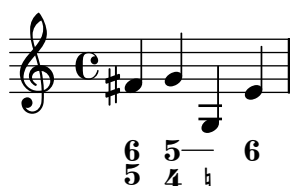
<<
{ fis4 g g, e' }
\figures {
  \bassFigureExtendersOn
  <6 5>4 <5\! 4> < 5 _!> <6>
}
>>

```



To avoid this problem, simply turn on extenders after the figure that begins the extender line and turn them off at the end of the extender line.

```
<<
{ fis4 g g, e' }
\figures {
  <6 5>4 <5 4>
  \bassFigureExtendersOn
  < 5 _!>4 <6>
  \bassFigureExtendersOff
}
>>
```



2.8 Contemporary music

From the beginning of the 20th Century there has been a massive expansion of compositional style and technique. New harmonic and rhythmic developments, an expansion of the pitch spectrum and the development of a wide range of new instrumental techniques have been accompanied by a parallel evolution and expansion of musical notation. The purpose of this section is to provide references and information relevant to working with these new notational techniques.

2.8.1 Pitch and harmony in contemporary music

This section highlights issues that are relevant to notating pitch and harmony in contemporary music.

References for pitch and harmony in contemporary music

- Standard quarter-tone notation is addressed in [\[Note names in other languages\]](#), page 7.
- Non-standard key signatures are addressed in [\[Key signature\]](#), page 16.
- Contemporary practises in displaying accidentals are addressed in [\[Automatic accidentals\]](#), page 20.

Microtonal notation

Contemporary key signatures and harmony

2.8.2 Contemporary approaches to rhythm

This section highlights issues that are relevant to the notation of rhythm in contemporary music.

References for contemporary approaches to rhythm

- Compound time signatures are addressed in [\[Time signature\]](#), page 52.
- Basic polymetric notation is addressed in [\[Polymetric notation\]](#), page 57.
- Feathered beams are addressed in [\[Feathered beams\]](#), page 72.
- Mensurstriche barlines (barlines between staves only) are addressed in [\[Grouping staves\]](#), page 137.

Tuplets in contemporary music

Contemporary time signatures

Extended polymetric notation

Beams in contemporary music

Bar lines in contemporary music

2.8.3 Graphical notation

2.8.4 Contemporary scoring techniques

2.8.5 New instrumental techniques

2.8.6 Further reading and scores of interest

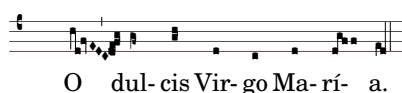
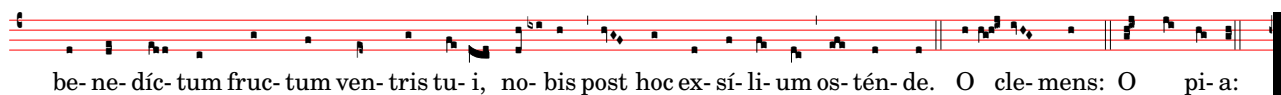
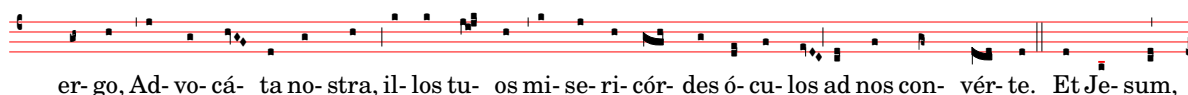
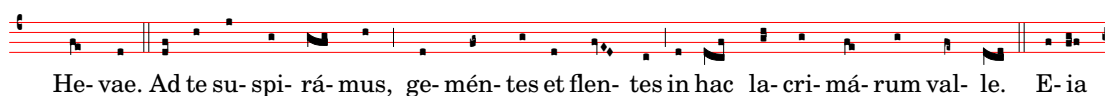
This section suggests books, musical examples and other resources useful in studying contemporary musical notation.

Books and articles on contemporary musical notation

- *Music Notation in the Twentieth Century: A Practical Guidebook* by Kurt Stone [W. W. Norton, 1980]
- *Music Notation: A Manual of Modern Practice* by Gardner Read [Taplinger, 1979]
- *Instrumentation and Orchestration* by Alfred Blatter [Schirmer, 2nd ed. 1997]

Scores and musical examples

2.9 Ancient notation



Support for ancient notation includes features for mensural notation and Gregorian chant notation. These features can be accessed either by modifying style properties of graphical objects such as note heads and rests, or by using one of the pre-defined contexts for mensural or Gregorian notation.

Many graphical objects, such as note heads and flags, accidentals, time signatures, and rests, provide a `style` property, which can be changed to emulate several different styles of ancient notation. See

- [Mensural note heads], page 304,
- [Mensural accidentals and key signatures], page 306,
- [Mensural rests], page 305,
- [Mensural clefs], page 302,
- [Gregorian clefs], page 308,
- [Mensural flags], page 304,
- [Mensural time signatures], page 303.

Some notational concepts are introduced specifically for ancient notation,

- [Custodes], page 300,
- [Divisiones], page 310,
- [Ligatures], page 300.

2.9.1 Overview of the supported styles

Three styles are available for typesetting Gregorian chant:

- *Editio Vaticana* is a complete style for Gregorian chant, following the appearance of the Solesmes editions, the official chant books of the Vatican since 1904. LilyPond has support for all the notational signs used in this style, including ligatures, *custodes*, and special signs such as the quilisma and the oriscus.
- The *Editio Medicaea* style offers certain features used in the Medicaea (or Ratisbona) editions which were used prior to the Solesmes editions. The most significant differences from the *Vaticana* style are the clefs, which have downward-slanted strokes, and the noteheads, which are square and regular.
- The *Hufnagel* (“horseshoe nail”) or *Gothic* style mimics the writing style in chant manuscripts from Germany and Central Europe during the middle ages. It is named after the basic note shape (the *virga*), which looks like a small nail.

Three styles emulate the appearance of late-medieval and renaissance manuscripts and prints of mensural music:

- The *Mensural* style most closely resembles the writing style used in late-medieval and early renaissance manuscripts, with its small and narrow, diamond-shaped noteheads and its rests which approach a hand-drawn style.
- The *Neomensural* style is a modernized and stylized version of the former: the noteheads are broader and the rests are made up of straight lines. This style is particularly suited, e.g., for incipits of transcribed pieces of mensural music.
- The *Petrucchi* style is named after Ottaviano Petrucci (1466-1539), the first printer to use movable type for music (in his *Harmonice musices odhecaton*, 1501). The style uses larger note heads than the other mensural styles.

Baroque and *Classical* are not complete styles but differ from the default style only in some details: certain noteheads (Baroque) and the quarter rest (Classical).

Only the mensural style has alternatives for all aspects of the notation. Thus, there are no rests or flags in the Gregorian styles, since these signs are not used in plainchant notation, and the Petrucci style has no flags or accidentals of its own.

Each element of the notation can be changed independently of the others, so that one can use mensural flags, petrucci noteheads, classical rests and vaticana clefs in the same piece, if one wishes.

2.9.2 Ancient notation—common features

Pre-defined contexts

For Gregorian chant and mensural notation, there are pre-defined voice and staff contexts available, which set all the various notation signs to values suitable for these styles. If one is satisfied with these defaults, one can proceed directly with note entry without worrying about the details on how to customize a context. See one of the pre-defined contexts `VaticanaVoice`, `VaticanaStaff`, `MensuralVoice`, and `MensuralStaff`. See further

- [\[Gregorian chant contexts\]](#), page 308,
- [\[Mensural contexts\]](#), page 301.

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 on the same syllable. They are also used in mensural notation.

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
  \[ e f a g \]
}
```



Two other ligature styles are available: the `Vaticana` for Gregorian chant, and the `Mensural` for mensural music (only white mensural ligatures are supported for mensural music, and with certain limitations). To use any of these styles, the default `Ligature_bracket_engraver` has to be replaced with one of the specialized ligature engravers in the `Voice` context, as explained in [\[White mensural ligatures\]](#), page 307 and [\[Gregorian square neume ligatures\]](#), page 311.

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\[... note\]`.

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 of the following line, thus helping the performer to manage line breaks during performance.

Custodes were frequently used in music notation until the seventeenth 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, and change the style of the custos with an `\override` if desired, as shown in the following example:



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.

<code>vaticana</code>	<code>medicaea</code>	<code>hufnagel</code>	<code>mensural</code>
		✓	✓

See also

Internals Reference: [Section “Custos” in *Internals Reference*](#).

Snippets: [Section “Ancient notation” in *Snippets*](#).

Figured bass support

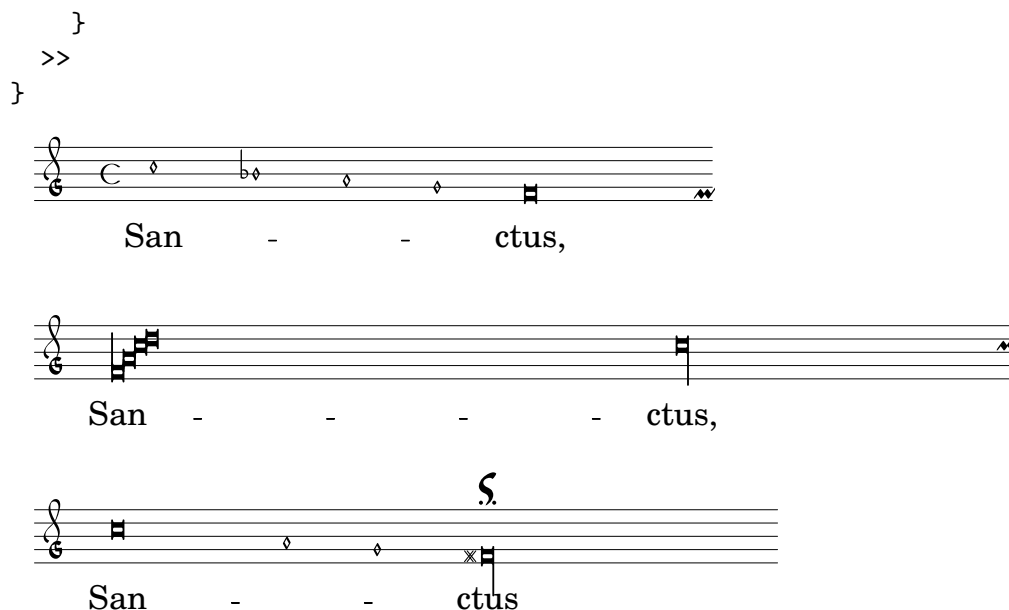
There is limited support for figured bass notation from the Baroque period; see [Section 2.7.3 \[Figured bass\]](#), page 290.

2.9.3 Typesetting mensural music

Mensural contexts

The predefined `MensuralVoice` and `MensuralStaff` contexts 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
    }
  }
}
```



Mensural clefs

The following table shows all mensural 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, numbered from the lowest to the highest line. Still, you can manually force a clef glyph to be typeset on an arbitrary line, as described in [\[Clef\]](#), page 13. The note printed to the right side of each clef in the example column denotes the `c'` with respect to that clef.

Petrucchi used C clefs with differently balanced left-side vertical beams, depending on which staff line it is printed.

Description	Supported Clefs	Example
mensural C clef	<code>mensural-c1</code> , <code>mensural-c2</code> , <code>mensural-c3</code> , <code>mensural-c4</code>	
mensural F clef	<code>mensural-f</code>	
mensural G clef	<code>mensural-g</code>	
neomensural C clef	<code>neomensural-c1</code> , <code>neomensural-c2</code> , <code>neomensural-c3</code> , <code>neomensural-c4</code>	
petrucci style C clefs, for use on different staff lines (the example shows the 2nd staff line C clef)	<code>petrucci-c1</code> , <code>petrucci-c2</code> , <code>petrucci-c3</code> , <code>petrucci-c4</code> , <code>petrucci-c5</code>	

petrucci style F clef

petrucci-f



petrucci style G clef

petrucci-g



See also

Notation Reference: [\[Clef\]](#), page 13.

Known issues and warnings

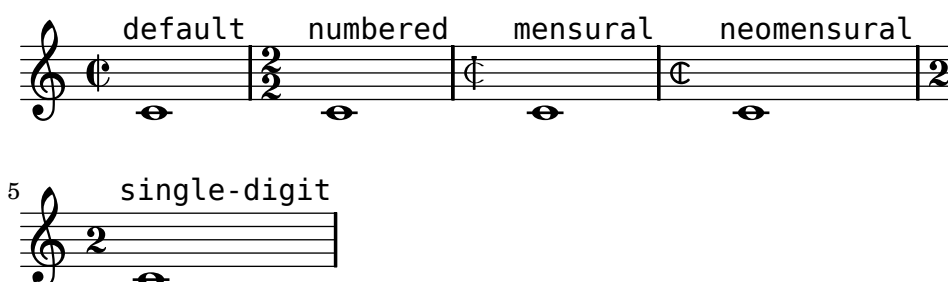
The mensural g clef is mapped to the Petrucci g clef.

Mensural time signatures

There is limited support for mensuration signs (which are similar to, but not exactly the same as time signatures). The glyphs are hard-wired to particular time fractions. In other words, to get a particular mensuration sign with the `\time n/m` command, `n` and `m` have to be chosen according to the following table

<code>\time 4/4</code>	<code>\time 6/4</code>	<code>\time 2/2</code>	<code>\time 6/8</code>
<code>\time 3/2</code>	<code>\time 3/4</code>	<code>\time 9/4</code>	<code>\time 9/8</code>
<code>\time 4/8</code>	<code>\time 2/4</code>		

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. The following examples show the differences in style:



See also

Notation Reference: [\[Time signature\]](#), page 52, 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 breve = 3 semibreves (*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 `mensural68alt` and `neomensural68alt` symbols (alternate symbols for 6/8) are not addressable with `\time`. Use `\markup {\musicglyph #"timesig.mensural68alt" }` instead.

Mensural 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 by:

- Providing a `maxima` notehead, and
- Using a square shape for `\breve` note heads.

The `neomensural`, `mensural`, and `petrucci` styles differ from the `baroque` style by:

- Using rhomboidal heads for semibreves and all smaller durations, and
- Centering the stems on the note heads.

The following example demonstrates the `petrucci` style:

```
\set Score.skipBars = ##t
\autoBeamOff
\override NoteHead #'style = #'petrucci
a'\maxima a'\longa a'\breve a'1 a'2 a'4 a'8 a'16 a'
```



See also

Notation Reference: [Section A.7 \[Note head styles\]](#), page 473, gives an overview of all available note head styles.

Mensural 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 or Petrucci notation. There are no flags in Gregorian chant notation.

Known issues and warnings

The attachment of ancient flags to stems is slightly off.

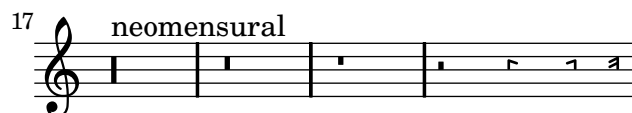
Vertically aligning each flag with a staff line assumes that stems always end either exactly on or exactly in the middle between two staff lines. This may not always be true when using advanced layout features of classical notation (which however are typically out of scope for mensural notation).

Mensural 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 `mensural` and the `neomensural` styles mimic the appearance of rests in manuscripts and prints up to the 16th century.

The following example demonstrates the `mensural` and `neomensural` styles:

```
\set Score.skipBars = ##t
\override Rest #'style = #'classical
r\longa^"classical" r\breve r1 r2 r4 r8 r16 s \break
\override Rest #'style = #'mensural
r\longa^"mensural" r\breve r1 r2 r4 r8 r16 s \break
\override Rest #'style = #'neomensural
r\longa^"neomensural" 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 [Section “Ancient notation” in *Snippets*](#) for a chart of all rests.

See also

Notation Reference: [\[Rests\]](#), [page 45](#), gives a general introduction into the use of rests.

Mensural accidentals and key signatures

The `mensural` style provides a sharp and a flat sign different from the default style. If called for, the natural sign will be taken from the `vaticana` style.

mensural

♭ ✖

The style for accidentals and key signatures is controlled by the `glyph-name-alist` property of the grobs `Accidental` and `KeySignature`, respectively; e.g.:

```
\override Staff.Accidental #'glyph-name-alist = #alteration-mensural-glyph-name-alist
```

See also

Notation Reference: [Section 1.1 \[Pitches\]](#), [page 1](#), [\[Accidentals\]](#), [page 4](#), and [\[Automatic accidentals\]](#), [page 20](#), give a general introduction of the use of accidentals. [\[Key signature\]](#), [page 16](#), gives a general introduction of the use of key signatures.

Internals Reference: [Section “KeySignature”](#) in *Internals Reference*.

Annotational accidentals (*musica ficta*)

In European music from before about 1600, singers were expected to chromatically alter notes at their own initiative according to certain rules. 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 accidental as *musica ficta* until it is unset with `\set suggestAccidentals = ##f`. A more practical way is to use `\once \set suggestAccidentals = ##t`, which can even be defined as a convenient shorthand:

```
ficta = { \once \set suggestAccidentals = ##t }
\score { \relative c''
  \new MensuralVoice {
    \once \set suggestAccidentals = ##t
    bes4 a2 g2 \ficta fis8 \ficta e! fis2 g1
  }
}
```



See also

Internals Reference: [Section “Accidental_engraver”](#) in *Internals Reference*, [Section “AccidentalSuggestion”](#) in *Internals Reference*.

White mensural ligatures

There is limited support for white mensural ligatures.

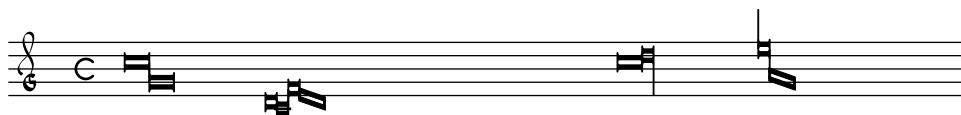
To engrave white mensural ligatures, in the layout block, replace the `Ligature_bracket_engraver` with the `Mensural_ligature_engraver` in the `Voice` context:

```
\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,

```
\score {
  \transpose c c' {
    \set Score.timing = ##f
    \set Score.defaultBarType = "empty"
    \override NoteHead #'style = #'neomensural
    \override Staff.TimeSignature #'style = #'neomensural
    \clef "petrucci-g"
    \[ c'\maxima g \]
    \[ d\longa c\breve f e d \]
    \[ c'\maxima d'\longa \]
    \[ e'1 a g\breve \]
  }
  \layout {
    \context {
      \Voice
      \remove Ligature_bracket_engraver
      \consists Mensural_ligature_engraver
    }
  }
}
```



Without replacing `Ligature_bracket_engraver` with `Mensural_ligature_engraver`, the same music transcribes to the following



Known issues and warnings

Horizontal spacing of ligatures is poor.

2.9.4 Typesetting Gregorian chant

When typesetting a piece in Gregorian chant notation, the `Vaticana_ligature_engraver` automatically selects 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, the `Mensural_ligature_engraver` automatically assembles mensural ligatures. See [\[Ligatures\]](#), page 300, for how ligature engravers work.

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.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,


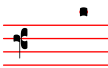







San- ctus

Gregorian clefs

The following table shows all Gregorian 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, numbered from the lowest to the highest line. Still, you can manually force a clef glyph to be typeset on an arbitrary line, as described in [\[Clef\]](#), page 13. 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
-------------	-----------------	---------

Editio Vaticana style do clef	<code>vaticana-do1</code> , <code>vaticana-do2</code> , <code>vaticana-do3</code>	
Editio Vaticana style fa clef	<code>vaticana-fa1</code> , <code>vaticana-fa2</code>	
Editio Medicaea style do clef	<code>medicaea-do1</code> , <code>medicaea-do2</code> , <code>medicaea-do3</code>	
Editio Medicaea style fa clef	<code>medicaea-fa1</code> , <code>medicaea-fa2</code>	
hufnagel style do clef	<code>hufnagel-do1</code> , <code>hufnagel-do2</code> , <code>hufnagel-do3</code>	
hufnagel style fa clef	<code>hufnagel-fa1</code> , <code>hufnagel-fa2</code>	
hufnagel style combined do/fa clef	<code>hufnagel-do-fa</code>	

See also

Notation Reference: [\[Clef\]](#), [page 13](#).

Gregorian accidentals and key signatures

Accidentals for the three different Gregorian styles are available:

vaticana medicaea hufnagel

♭ ♯ ♮ ♭

As shown, not all accidentals are supported by each style. When trying to access an unsupported accidental, LilyPond will switch to a different style.

The style for accidentals and key signatures is controlled by the `glyph-name-alist` property of the grobs `Accidental` and `KeySignature`, respectively; e.g.:

```
\override Staff.Accidental #'glyph-name-alist = #alteration-mensural-glyph-name-alist
```

See also

Notation Reference: [Section 1.1 \[Pitches\]](#), [page 1](#), [\[Accidentals\]](#), [page 4](#), and [\[Automatic accidentals\]](#), [page 20](#), give a general introduction of the use of accidentals. [\[Key signature\]](#), [page 16](#), gives a general introduction of the use of key signatures.

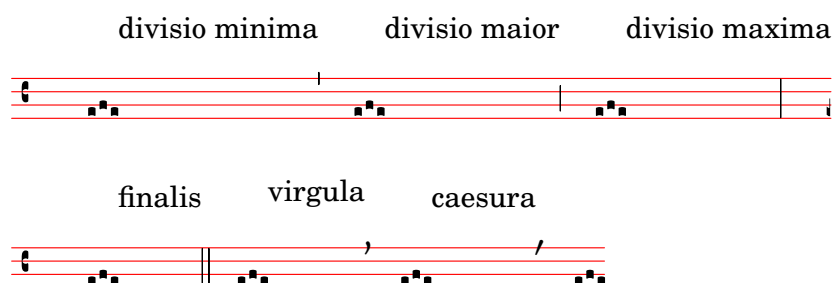
Internals Reference: [Section “KeySignature” in *Internals Reference*](#).

Divisiones

There are no rests in Gregorian chant notation; instead, it uses [\[Divisiones\]](#), page 310.

A *divisio* (plural: *divisiones*; Latin word for ‘division’) is a staff context symbol that is used to indicate the phrase and section structure of Gregorian music. 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 [\[Breath marks\]](#), page 102. 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.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.ly’ also defines `\virgula` and `\caesura`



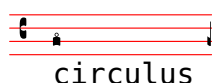
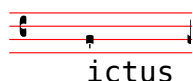
Predefined commands

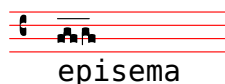
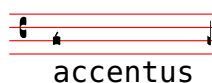
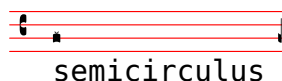
`\virgula`, `\caesura`, `\divisioMinima`, `\divisioMaior`, `\divisioMaxima`, `\finalis`.

Gregorian articulation signs

In addition to the standard articulation signs described in section [\[Articulations and ornaments\]](#), page 90, articulation signs specifically designed for use with notation in *Editio Vaticana* style are provided.

```
\include "gregorian.ly"
\score {
  \new VaticanaVoice {
    \override TextScript #'font-family = #'typewriter
    \override TextScript #'font-shape = #'upright
    \override Script #'padding = #-0.1
    a\ictus_"ictus " \bar "" \break
    a\circulus_"circulus " \bar "" \break
    a\semicirculus_"semicirculus " \bar "" \break
    a\accentus_"accentus " \bar "" \break
    \[ a_"episema" \epistemInitium \pes b \flexa a b \epistemFinis \flexa a \]
  }
}
```





Snippets: Section “Ancient notation” in *Snippets*.

Internals Reference: Section “Episema” in *Internals Reference*, Section “EpisemaEvent” in *Internals Reference*, Section “Episema_engraver” in *Internals Reference*, Section “Script” in *Internals Reference*, Section “ScriptEvent” in *Internals Reference*, Section “Script_engraver” in *Internals Reference*.

Known issues and warnings

Some articulations are vertically placed too closely to the corresponding note heads.

Augmentum dots (*morae*)

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.ly"
\score {
  \new VaticanaVoice {
    \[ \augmentum a \flexa \augmentum g \]
    \augmentum g
  }
}
```



See also

Notation Reference: [Breath marks], page 102.

Internals Reference: Section “BreathingSign” in *Internals Reference*.

Snippets: Section “Ancient notation” in *Snippets*.

Gregorian square neume 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 support for Gregorian neumes is enabled by `\includeing "gregorian.ly"` at the beginning of the file. This makes available a number of extra commands to produce the neume symbols used in plainchant notation.

Note heads can be *modified* and/or *joined*.

- The shape of the note head can be modified by *prefixing* the note name with any of the following commands: `\virga`, `\strophica`, `\inclinatum`, `\auctum`, `\descendens`, `\ascendens`, `\oriscus`, `\quilisma`, `\deminutum`, `\cavum`, `\linea`.
- Ligatures, properly speaking (i.e. notes joined together), are produced by placing one of the joining commands `\pes` or `\flexa`, for upwards and downwards movement, respectively, *between* the notes to be joined.

A note name without any qualifiers will produce a *punctum*. All other neumes, including the single-note neumes with a different shape such as the *virga*, are in principle considered as ligatures and should therefore be placed between `\[...]`.

Single-note neumes:

- The *punctum* is the basic note shape (in the *Vaticana* style: a square with some curvature for typographical finesse). In addition to the regular *punctum*, there is also the oblique *punctum inclinatum*, produced with the prefix `\inclinatum`. The regular *punctum* can be modified with `\cavum`, which produces a hollow note, and `\linea`, which draws vertical lines on either side of the note.
- The *virga* has a descending stem on the right side. It is produced by the modifier `\virga`.

Ligatures

Unlike most other neumes notation systems, the typographical appearance of ligatures is not directly dictated by the input commands, but follows certain conventions dependent on musical meaning. For example, a three-note ligature with the musical shape low-high-low, such as `\[a \pes b \flexa g]`, produces a Torculus consisting of three Punctum heads, while the shape high-low-high, such as `\[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.

Liquescent neumes

Another main category of notes in Gregorian chant is the so-called liquescent neumes. They are used under certain circumstances at the end of a syllable which ends in a ‘liquescent’ letter, i.e. the sounding consonants that can hold a tone (the nasals, l, r, v, j, and their diphthong equivalents). Thus, the liquescent neumes are never used alone (although some of them can be produced), and they always fall at the end of a ligature.

Liquescent neumes are represented graphically in two different, more or less interchangeable ways: with a smaller note or by ‘twisting’ the main note upwards or downwards. The first is produced by making a regular `\pes` or `\flexa` and modifying the shape of the second note: `\[a \pes \deminutum b]`, the second by modifying the shape of a single-note neume with `\auctum` and one of the direction markers `\descendens` or `\ascendens`, e.g., `\[\auctum \descendens a]`.

Special signs

A third category of signs is made up of a small number of signs with a special meaning (which, incidentally, in most cases is only vaguely known): the *quilisma*, the *oriscus*, and the *strophicus*. These are all produced by prefixing a note name with the corresponding modifier, `\quilisma`, `\oriscus`, or `\strophica`.







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.

Note that the use of these signs in the music itself follows certain rules, which are not checked by LilyPond. E.g., the *quilisma* is always the middle note of an ascending ligature, and usually falls on a half-tone step, but it is perfectly possible, although incorrect, to make a single-note quilisma.

In addition to the note signs, gregorian.ly also defines the commands `\versus`, `\responsum`, `\ij`, `\iij`, `\IJ`, and `\IIJ`, that will produce the corresponding characters, e.g., for use in lyrics, as section markers, etc. These commands use special unicode characters and will only work if a font is used which supports them.

The following table shows a limited, but still representative pool of Gregorian ligatures, together with the code fragments that produce the ligatures. The table is based on the extended neumes table of the 2nd volume of the Antiphonale Romanum (*Liber Hymnarius*), published 1983 by the monks of Solesmes. The first column gives the name of the ligature, with the main form in boldface and the liquescent forms in italics. The third column shows the code fragment that produces this ligature, using `g`, `a`, and `b` as example pitches.

Single-note neums

Basic and <i>Liquescent</i> forms	Output	LilyPond code
Punctum		<code>\[b \]</code>
		<code>\[\cavum b \]</code>
		<code>\[\linea b \]</code>
<i>Punctum Auctum Ascendens</i>		<code>\[\auctum \ascendens b \]</code>
<i>Punctum Auctum Descendens</i>		<code>\[\auctum \descendens b \]</code>
Punctum inclinatum		<code>\[\inclinatum b \]</code>

Punctum Inclinatum Auctum

\[\inclinatum \auctum b \]

Punctum Inclinatum Parvum

\[\inclinatum \deminutum b \]

Virga**Two-note ligatures****Clivis vel Flexa**

\[b \flexa g \]

Clivis Aucta Descendens

\[b \flexa \auctum \descendens g \]

Clivis Aucta Ascendens

\[b \flexa \auctum \ascendens g \]

Cephalicus

\[b \flexa \deminutum g \]

Podatus/Pes

\[g \pes b \]

Pes Auctus Descendens

\[g \pes \auctum \descendens b \]

Pes Auctus Ascendens

\[g \pes \auctum \ascendens b \]

Epiphonus

\[g \pes \deminutum b \]

*Pes Initio Debilis*

\[\deminutum g \pes b \]

*Pes Auctus Descendens Initio Debilis*\[\deminutum g \pes \auctum
\descendens b \]**Multi-note ligatures****Torculus**

\[a \pes b \flexa g \]

*Torculus Auctus Descendens*\[a \pes b \flexa \auctum
\descendens g \]*Torculus Deminutus*\[a \pes b \flexa \deminutum g
\]*Torculus Initio Debilis*\[\deminutum a \pes b \flexa g
\]*Torculus Auctus Descendens Initio
Debilis*\[\deminutum a \pes b \flexa
\auctum \descendens g \]*Torculus Deminutus Initio Debilis*\[\deminutum a \pes b \flexa
\deminutum g \]

Porrectus $\backslash[a \backslash flexa g \backslash pes b \backslash]$ *Porrectus Auctus Descendens* $\backslash[a \backslash flexa g \backslash pes \backslash auctum \backslash descendens b \backslash]$ *Porrectus Deminutus* $\backslash[a \backslash flexa g \backslash pes \backslash deminutum b \backslash]$ **Climacus** $\backslash[\backslash virga b \backslash inclinatum a \backslash inclinatum g \backslash]$ *Climacus Auctus* $\backslash[\backslash virga b \backslash inclinatum a \backslash inclinatum \backslash auctum g \backslash]$ *Climacus Deminutus* $\backslash[\backslash virga b \backslash inclinatum a \backslash inclinatum \backslash deminutum g \backslash]$ **Scandicus** $\backslash[g \backslash pes a \backslash virga b \backslash]$ *Scandicus Auctus Descendens* $\backslash[g \backslash pes a \backslash pes \backslash auctum \backslash descendens b \backslash]$ *Scandicus Deminutus* $\backslash[g \backslash pes a \backslash pes \backslash deminutum b \backslash]$ **Special Signs**

Quilisma $\backslash[g \backslash pes \backslash quilisma a \backslash pes b \backslash]$ *Quilisma Pes Auctus Descendens* $\backslash[\backslash quilisma g \backslash pes \backslash auctum \backslash descendens b \backslash]$ **Oriscus** $\backslash[\backslash oriscus b \backslash]$ *Pes Quassus* $\backslash[\backslash oriscus g \backslash pes \backslash virga b \backslash]$ *Pes Quassus Auctus Descendens* $\backslash[\backslash oriscus g \backslash pes \backslash auctum \backslash descendens b \backslash]$ **Salicus** $\backslash[g \backslash oriscus a \backslash pes \backslash virga b \backslash]$ *Salicus Auctus Descendens* $\backslash[g \backslash oriscus a \backslash pes \backslash auctum \backslash descendens b \backslash]$ **(Apo)stroph** $\backslash[\backslash stroph a b \backslash]$ *Stroph Aucta* $\backslash[\backslash stroph a \backslash auctum b \backslash]$ **Bistroph** $\backslash[\backslash stroph a b \backslash stroph a b \backslash]$ 

Tristropha

◆◆◆

```
\[ \stropha b \stropha b
\stropha b \]
```

Trigonus

◆◆

```
\[ \stropha b \stropha b
\stropha a \]
```

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.

2.9.5 Working with ancient music—scenarios and solutions

Working with ancient music frequently involves particular tasks which differ considerably from the modern notation for which LilyPond is designed. In the rest of this section, a number of typical scenarios are outlined, with suggestions of solutions. These involve:

- how to make incipits (i.e. prefatory material to indicate what the original has looked like) to modern transcriptions of mensural music;
- how to achieve the *Mensurstriche* layout frequently used for modern transcriptions of polyphonic music;
- how to transcribe Gregorian chant in modern notation;
- how to generate both ancient and modern notation from the same source.

Incipits

TBC

Mensurstriche layout

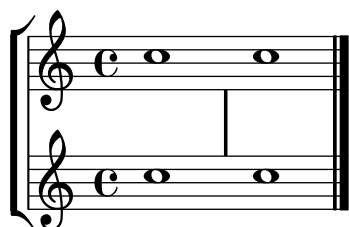
Mensurstriche ('mensuration lines') is the accepted term for bar lines that are drawn between the staves of a system but not through the staves themselves. It is a common way to preserve the rhythmic appearance of the original, i.e. not having to break syncopated notes at bar lines, while still providing the orientation aids that bar lines give.

The mensurstriche-layout where the bar lines do not show on the staves but between staves can be achieved with a `StaffGroup` instead of a `ChoirStaff`. The bar line on staves is blanked out by setting the `transparent` property.

```

global = {
  \override Staff.BarLine #'transparent = ##t
  s1 s
  % the final bar line is not interrupted
  \revert Staff.BarLine #'transparent
  \bar "|."
}
\new StaffGroup \relative c'' {
  <<
    \new Staff { << \global { c1 c } >> }
    \new Staff { << \global { c c } >> }
  >>
}

```



Transcribing Gregorian chant

Gregorian chant can be transcribed into modern notation with a number of simple tweaks.

Stems. Stems can be left out altogether by `\remove`-ing the `Stem_engraver` from the `Voice` context:

```

\layout {
  ...
  \context {
    \Voice
    \remove "Stem_engraver"
  }
}

```

However, in some transcription styles, stems are used occasionally, for example to indicate the transition from a single-tone recitative to a fixed melodic gesture. In these cases, one can use either `\override Stem #'transparent = ##t` or `\override Stem #'length = #0` instead, and restore the stem when needed with the corresponding `\once \override Stem #'transparent = ##f` (see example below).

Timing. For unmetered chant, there are several alternatives.

The `Time_signature_engraver` can be removed from the `Staff` context without any negative side effects. The alternative, to make it transparent, will leave an empty space in the score, since the invisible signature will still take up space.

In many cases, `\set Score.timing = ##f` will give good results. Another alternative is to use `\CadenzaOn` and `\CadenzaOff`.

To remove the barlines, the radical approach is to `\remove` the `Bar_engraver` from the `Staff` context. Again, one may want to use `\override BarLine #'transparent = ##t` instead, if an occasional barline is wanted.

A common type of transcription is recitativic chant where the repeated notes are indicated with a single breve. The text to the recitation tone can be dealt with in two different ways: either set as a single, left-aligned syllable:


Noctem quietam et finem perfectum concedat nobis Dominus omnipotens.

```

\include "gregorian.ly"
chant = \relative c' {
  \clef "G_8"
  \set Score.timing = ##f
  c\breve \override NoteHead #'transparent = ##t  c c c c c
  \revert NoteHead #'transparent
  \override Stem #'transparent = ##f \stemUp c4 b4 a
  \override Stem #'transparent = ##t c2 c4 \divisioMaior
  c\breve \override NoteHead #'transparent = ##t c c c c c c c
  \revert NoteHead #'transparent c4 c f, f \finalis
}

verba = \lyricmode {
  No -- ctem qui -- e -- tam et fi -- nem per -- fec -- tum
  con -- ce -- dat no -- bis Do -- mi -- nus om -- ni -- po -- tens.
}

```

```

\score {
  \new Staff <<
    \new Voice = "melody" \chant
    \new Lyrics \lyricsto "melody" \verba
  >>
  \layout {
    \context {
      \Staff
      \remove "Time_signature_engraver"
      \override BarLine #'transparent = ##t
      \override Stem #'transparent = ##t
    }
  }
}

```



Another common situation is transcription of neumatic or melismatic chants, i.e. chants with a varying number of notes to each syllable. In this case, one would want to set the syllable groups clearly apart, usually also the subdivisions of a longer melisma. One way to achieve this is to use a fixed `\time`, e.g., `1/4`, and let each syllable or note group fill one of these measures, with the help of tuplets or shorter durations. If the barlines and all other rhythmical indications are made transparent, and the space around the barlines is increased, this will give a fairly good representation in modern notation of the original.

To avoid that syllables of different width (such as “-ri” and “-rum”) spread the syllable note groups unevenly apart, the `#'X-extent` property of the `LyricText` object may be set to a fixed value. Another, more cumbersome way would be to add the syllables as `\markup` elements. If further adjustments are necessary, this can be easily done with `s` ‘notes’.

```

spiritus = \relative c' {
  \time 1/4
  \override Lyrics.LyricText #'X-extent = #'(0 . 3)
  d4 \times 2/3 { f8 a g } g a a4 g f8 e
  d4 f8 g g8 d f g a g f4 g8 a a4 s
  \times 2/3 { g8 f d } e f g a g4
}

spirLyr = \lyricmode {
  Spi -- ri -- _ _ tus _ Do -- mi -- ni _ re -- ple -- _ vit _
  or -- _ bem _ ter -- ra -- _ rum, al -- _ _ le -- _ lu
  -- _ ia.
}

\score {
  \new Staff <<
    \new Voice = "chant" \spiritus
    \new Lyrics = "one" \lyricsto "chant" \spirLyr
  >>
  \layout {
    \context {
      \Staff

```

```

\remove "Time_signature_engraver"
\override BarLine #'X-extent = #'(-1 . 1)
\override Stem #'transparent = ##t
\override Beam #'transparent = ##t
\override BarLine #'transparent = ##t
\override TupletNumber #'transparent = ##t
}
}
}

```



Ancient and modern from one source

TBC

Editorial markings

TBC

2.10 World music

The purpose of this section is to highlight musical notation issues that are relevant to traditions outside the Western tradition.

2.10.1 Arabic music

This section highlights issues that are relevant to notating Arabic music.

References for Arabic music

Arabic music so far has been mainly an oral tradition. When music is transcribed, it is usually in a sketch format, on which performers are expected to improvise significantly. Increasingly, Western notation, with a few variations, is adopted in order to communicate and preserve Arabic music.

Some elements of Western musical notation such as the transcription of chords or independent parts, are not required to typeset the more traditional Arabic pieces. There are however some different issues, such as the need to indicate medium intervals that are somewhere between a semi-tone and a tone, in addition to the minor and major intervals that are used in Western music. There is also the need to group and indicate a large number of different maqams (modes) that are part of Arabic music.

In general, Arabic music notation does not attempt to precisely indicate microtonal elements that are present in musical practice.

Several issues that are relevant to Arabic music are covered elsewhere:

- Note names and accidentals (including quarter tones) can be tailored as discussed in [\[Note names in other languages\]](#), page 7.

- Additional key signatures can also be tailored as described in [Key signature], page 16.
- Complex time signatures may require that notes be grouped manually as described in [Manual beams], page 69.
- *Takasim* which are rhythmically free improvisations may be written down omitting bar lines as described in [Unmetered music], page 55.

See also

Notation Reference: [Note names in other languages], page 7, [Key signature], page 16, [Manual beams], page 69.

Snippets: Section “World music” in *Snippets*.

Arabic note names

The more traditional Arabic note names can be quite long and are not suitable for the purpose of music writing, so they are not used. English note names are not very familiar in Arabic music education, so Italian or Solfege note names (do, re, mi, fa, sol, la, si) are used instead. Modifiers (accidentals) can also be used, as discussed in [Note names in other languages], page 7.

For example, this is how the Arabic *rast* scale can be notated:

```
\include "arabic.ly"
\relative do' {
  do re misb fa sol la sisb do sisb la sol fa misb re do
}
```



The symbol for semi-flat does not match the symbol which is used in Arabic notation. The `\dwn` symbol defined in `arabic.ly` may be used preceding a flat symbol as a work around if it is important to use the specific Arabic semi-flat symbol. The appearance of the semi-flat symbol in the key signature cannot be altered by using this method.

```
\include "arabic.ly"
\relative do' {
  \set Staff.extraNatural = ##f
  dod dob dosd \dwn dob dobsb dodsd do do
}
```



See also

Notation Reference: [Note names in other languages], page 7.

Snippets: Section “World music” in *Snippets*.

Arabic key signatures

In addition to the minor and major key signatures, the following key signatures are defined in `arabic.ly`: *bayati*, *rast*, *sikah*, *iraq*, and *kurd*. These key signatures define a small number of maqam groups rather than the large number of maqams that are in common use.

In general, a maqam uses the key signature of its group, or a neighbouring group, and varying accidentals are marked throughout the music.

For example to indicate the key signature of a maqam muhayer piece:

```
\key re \bayati
```

Here *re* is the default pitch of the muhayer maqam, and *bayati* is the name of the base maqam in the group.

While the key signature indicates the group, it is common for the title to indicate the more specific maqam, so in this example, the name of maqam muhayer should appear in the title.

Other maqams in the same bayati group, as shown in the table below: (bayati, hussaini, saba, and ushaq) can be indicated in the same way. These are all variations of the base and most common maqam in the group, which is bayati. They usually differ from the base maqam in their upper tetrachords, or certain flow details that don't change their fundamental nature, as siblings.

The other maqam in the same group (Nawa) is related to bayati by modulation which is indicated in the table in parenthesis for those maqams that are modulations of their base maqam. Arabic maqams admit of only limited modulations, due to the nature of Arabic musical instruments. Nawa can be indicated as follows:

```
\key sol \bayati
```

In Arabic music, the same term such as bayati that is used to indicate a maqam group, is also a maqam which is usually the most important in the group, and can also be thought of as a base maqam.

Here is one suggested grouping that maps the more common maqams to key signatures:

maqam group	key	finalis	Other maqmas in group (finalis)
ajam	major	sib	jaharka (fa)
bayati	bayati	re	hussaini, muhayer, saba, ushaq, nawa (sol)
hijaz	kurd	re	shahnaz, shad arban (sol), hijazkar (do)
iraq	iraq	sisb	-
kurd	kurd	re	hijazkar kurd (do)
nahawand	minor	do	busalik (re), farah faza (sol)
nakriz	minor	do	nawa athar, hisar (re)
rast	rast	do	mahur, yakah (sol)
sikah	sikah	misb	huzam

Selected Snippets

Non-traditional key signatures

The commonly used `\key` command sets the `keySignature` property, in the `Staff` context.

To create non-standard key signatures, set 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.)
```

Alternatively, for each item in the list, using the more concise format `(step . alter)` 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 . 6) . ,FLAT)
                                ((0 . 5) . ,FLAT)
                                ((0 . 3) . ,SHARP))

  c4 d e fis
  aes4 bes c2
}
```



See also

Notation Reference: [\[Key signature\]](#), page 16.

Learning Manual: [Section “Accidentals and key signatures”](#) in *Learning Manual*.

Internals Reference: [Section “KeySignature”](#) in *Internals Reference*.

Snippets: [Section “World music”](#) in *Snippets*, [Section “Pitches”](#) in *Snippets*.

Arabic time signatures

Some Arabic and Turkish music classical forms such as *Semai* use unusual time signatures such as 10/8. This may lead to an automatic grouping of notes that is quite different from existing typeset music, where notes may not be grouped on the beat, but in a manner that is difficult to match by adjusting automatic beaming. You can override this by switching off automatic beaming and beaming the notes manually. Where matching existing typeset music is not an issue, you may still want to adjust the beaming behaviour and/or use compound time signatures.

Selected Snippets

Compound time signatures

Odd 20th century time signatures (such as "5/8") can often be played as compound time signatures (e.g. "3/8 + 2/8"), which combine two or more unequal metrics. LilyPond can make such music quite easy to read and play, by explicitly printing the compound time signatures and adapting the automatic beaming behavior. (Graphic measure grouping indications can also be added; see the appropriate snippet in this database.)

```
#(define ((compound-time one two num) grob)
  (grob-interpret-markup grob
    (markup #:override '(baseline-skip . 0) #:number
      (:line ((#:column (one num))
        #:vcenter "+"
        (:column (two num)))))))

\relative c' {
  \override Staff.TimeSignature #'stencil = #(compound-time "2" "3" "8")
  \time 5/8
  \override BeamSettings #'Staff #'(5 . 8) #'end
  #'((* . (2 3)))
  c8 d e fis gis
  c8 fis, gis e d
  c8 d e4 gis8
}
```

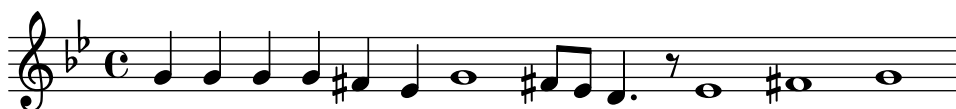


Arabic improvisation

For improvisations or taqasim which are temporarily free, the time signature can be omitted and `\cadenzaOn` can be used. Adjusting the accidental style might be required, since the absence of bar lines will cause the accidental to be marked only once. Here is an example of what could be the start of a hijaz improvisation:

```
\include "arabic.ly"
```

```
\relative sol' {
  \key re \kurd
  #(set-accidental-style 'forget)
  \cadenzaOn
  sol4 sol sol sol fad mib sol1 fad8 mib re4. r8 mib1 fad sol
}
```



See also

Notation Reference: [\[Manual beams\]](#), page 69, [\[Automatic beams\]](#), page 63, [\[Unmetered music\]](#), page 55, [\[Automatic accidentals\]](#), page 20, [\[Setting automatic beam behavior\]](#), page 64, [\[Time signature\]](#), page 52.

Snippets: [Section “World music” in *Snippets*](#).

Arabic music example

Here is a template that also uses the start of a Turkish Semai that is familiar in Arabic music education in order to illustrate some of the peculiarities of Arabic music notation, such as medium intervals and unusual modes that are discussed in this section.

```
\include "arabic.ly"
```

```
\score {
  \relative re' {
    \set Staff.extraNatural = ##f
    \set Staff.autoBeaming = ##f
    \key re \bayati
    \time 10/8

    re4 re'8 re16 [misb re do] sisb [la sisb do] re4 r8
    re16 [misb do re] sisb [do] la [sisb sol8] la [sisb] do [re] misb
    fa4 fa16 [misb] misb8. [re16] re8 [misb] re [do] sisb
    do4 sisb8 misb16 [re do sisb] la [do sisb la] la4 r8
  }
  \header {
    title = "Semai Muhayer"
    composer = "Jamil Bek"
  }
}
```





See also

Snippets: [Section “World music” in *Snippets*](#).

Further reading for Arabic music

1. *The music of the Arabs* by Habib Hassan Touma [Amadeus Press, 1996], contains a discussion of maqams and their method of groupings.

There are also various web sites that explain maqams and some provide audio examples such as :

- <http://www.maqamworld.com/>
- <http://www.turath.org/>

There are some variations in the details of how maqams are grouped, despite agreement on the criteria of grouping maqams that are related through common lower tetra chords, or through modulation.

2. There is not a complete consistency, sometimes even in the same text on how key signatures for particular maqams should be specified. It is common, however, to use a key signature per group, rather than a different key signature for each different makam.

Method books by the following authors for the *Oud*, the Arabic lute, contain examples of mainly Turkish and Arabic compositions.

- Charbel Rouhana
- George Farah
- Ibrahim Ali Darwish Al-masri

2.10.2 Turkish classical music

This section highlights issues that are relevant to notating Turkish classical music.

References for Turkish classical music

Turkish classical music developed in the Ottoman Empire in a period roughly contemporaneous with classical music in Europe, and has continued on into the 20th and 21st centuries as a vibrant and distinct tradition with its own compositional forms, theory and performance styles. Among its striking features is the use of microtonal intervals based on ‘commas’ of $1/9$ of a tone, from which are constructed the melodic forms known as *makam* (plural *makamlar*).

Some issues relevant to Turkish classical music are covered elsewhere:

- Note names and accidentals are provided in [\[Non-Western note names and accidentals\]](#), page 8.

Turkish note names

Pitches in Turkish classical music traditionally have unique names, and the basis of pitch on $1/9$ -tone divisions means makamlar employ a completely different set of intervals from Western scales and modes: *koma* ($1/9$ of a tone), *eksik bakiye* ($3/9$), *bakiye* ($4/9$), *küçük mücenneb* ($5/9$), *büyük mücenneb* ($8/9$), *tanîni* (a whole tone) and *artık ikili* ($12/9$ or $13/9$ of a tone).

From a modern notational point of view it is convenient to use the standard Western staff notes (c, d, e, ...) with special accidentals that raise or lower notes by intervals of $1/9$, $4/9$, $5/9$ and $8/9$ of a tone. These accidentals are defined in the file ‘*makam.ly*’ (to locate this file on your system, see [Section “Other sources of information” in *Learning Manual*](#)). A more detailed description is provided in [\[Non-Western note names and accidentals\]](#), page 8.

3 General input and output

This section deals with general LilyPond input and output issues, rather than specific notation.

3.1 Input structure

The main format of input for LilyPond are text files. By convention, these files end with `.ly`.

3.1.1 Structure of a score

A `\score` block must contain a single music expression delimited by curly brackets:

```
\score {
...
}
```

Note: There must be **only one** outer music expression in a `\score` block, and it **must** be surrounded by curly brackets.

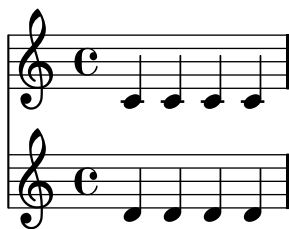
This single music expression may be of any size, and may contain other music expressions to any complexity. All of these examples are music expressions:

```
{ 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 }
>>
>>
}
```

```
}
```

Comments are one exception to this general rule. (For others see [Section 3.1.3 \[File structure\]](#), [page 330](#).) Both single-line comments and comments delimited by `%{ .. %}` may be placed anywhere within an input file. They may be placed inside or outside a `\score` block, and inside or outside the single music expression within a `\score` block.

See also

Learning Manual: [Section “Working on input files”](#) in *Learning Manual*, [Section “Music expressions explained”](#) in *Learning Manual*, [Section “Score is a \(single\) compound musical expression”](#) in *Learning Manual*.

3.1.2 Multiple scores in a book

A document may contain multiple pieces of music and text. 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 3.1.3 \[File structure\]](#), [page 330](#). 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" }
}

```

Pieces of music may be grouped into book parts using `\bookpart` blocks. Book parts are separated by a page break, and can start with a title, like the book itself, by specifying a `\header` block.

```

\bookpart {
  \header {
    title = "Book title"
    subtitle = "First part"
  }
  \score { ... }
  ...
}
\bookpart {
  \header {
    subtitle = "Second part"
  }
  \score { ... }
  ...
}

```

3.1.3 File structure

A `.ly` file may contain 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 book-wide settings. If more than one such definition of the same type is entered at the top level any definitions in the later expressions have precedence.
- 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'`.
- 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 to create 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 `\bookpart` block. A book may be divided into several parts, using `\bookpart` blocks, in order to ease the page breaking, or to use different `\paper` settings in different parts.
- 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 \scores` 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.

- A variable, such as

```
foo = { c4 d e d }
```

This can be used later on in the file by entering `\foo`. The name of a variable should have alphabetic characters only; no numbers, underscores or dashes.

The following example shows three things that may be entered at `toplevel`

```
\layout {
  % Don't justify the output
  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`

- A single-line comment, introduced by a leading % sign.
- A multi-line comment delimited by %{ .. %}.

Whitespace between items in the input stream is generally ignored, and may be freely omitted or extended to enhance readability. However, whitespace should always be used in the following circumstances to avoid errors:

- Around every opening and closing curly bracket.
- After every command or variable, i.e. every item that begins with a \ sign.
- After every item that is to be interpreted as a Scheme expression, i.e. every item that begins with a # sign.
- To separate all elements of a Scheme expression.
- In `lyricmode` to separate all the terms in both `\override` and `\set` commands. In particular, spaces must be used around the dot and the equals sign in commands like `\override Score . LyricText #'font-size = #5` and before and after the entire command.

See also

Learning Manual: [Section “How LilyPond input files work”](#) in *Learning Manual*.

3.2 Titles and headers

Almost all printed music includes a title and the composer’s name; some pieces include a lot more information.

3.2.1 Creating titles

Titles are created for each `\score` block, as well as for the full input file (or `\book` block) and book parts (created by `\bookpart` blocks).

The contents of the titles are taken from the `\header` blocks. The header block for a book supports the following

<code>dedication</code>	The dedicatee of the music, centered at the top of the first page.
<code>title</code>	The title of the music, centered just below the dedication.
<code>subtitle</code>	Subtitle, centered below the title.
<code>subsubtitle</code>	Subsubtitle, centered below the subtitle.
<code>poet</code>	Name of the poet, flush-left below the subsubtitle.
<code>instrument</code>	Name of the instrument, centered below the subsubtitle. Also centered at the top of pages (other than the first page).
<code>composer</code>	Name of the composer, flush-right below the subsubtitle.
<code>meter</code>	Meter string, flush-left below the poet.
<code>arranger</code>	Name of the arranger, flush-right below the composer.
<code>piece</code>	Name of the piece, flush-left below the meter.
<code>opus</code>	Name of the opus, flush-right below the arranger.
<code>breakbefore</code>	This forces the title to start on a new page (set to <code>##t</code> or <code>##f</code>).

copyright

Copyright notice, centered at the bottom of the first page. To insert the copyright symbol, see [Section 3.3.3 \[Text encoding\]](#), page 344.

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 \[Formatting text\]](#), page 182, commands in the header.

```
\paper {
  line-width = 9.0\cm
  paper-height = 10.0\cm
}

\book {
  \header {
    dedication = "dedicated to me"
    title = \markup \center-column { "Title first line" "Title second line,
longer" }
    subtitle = "the subtitle,"
    subsubtitle = #(string-append "subsubtitle LilyPond version "
(lilypond-version))
    poet = "Poet"
    composer = \markup \center-column { "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"
    }
  }
}
```

dedicated to me

Title first line

Title second line, longer

the subtitle,

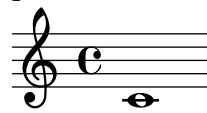
subsubtitle LilyPond version 2.13.27

Poet *instrument* composer

(1847-1973)

meter **a r r a n g e r**


piece1 opus1



and now...

2 *instrument* opus2

piece2



Music engraving by LilyPond 2.13.27—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{
  print-all-headers = ##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
}
```

3.2.2 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.

¹ Nicely printed parts are good PR for us, so please leave the tagline if you can.

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.

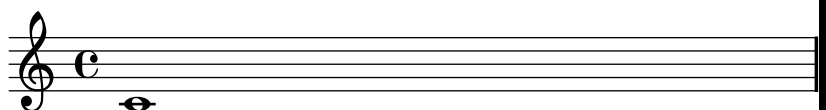
```
\paper {
  bookTitleMarkup = \markup {
    \fill-line {
      \fromproperty #'header:title
      \fromproperty #'header:composer
    }
  }
}
```

3.2.3 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 referred 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'1
    }
  }

  \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 1

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 occurred, 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`.

3.2.4 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'4 % ...
```

```
\tocItem \markup "Some particular point in the first score"
```

```
d'4 % ...
```

```
}
```

```
}
```

```
\tocItem \markup "Second score"
```

```
\score {
```

```
{
```

```
e'4 % ...
```

```
}
```

```
}
```

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 larger font size
tocItemMarkup = \markup \large \fill-line {
  \fromproperty #'toc:text \fromproperty #'toc:page
}
}

```

Note how the toc element text and page number are referred 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`.

3.3 Working with input files

3.3.1 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 "otherfile.ly"` is equivalent to pasting the contents of `'otherfile.ly'` into the current file at the place where the `\include` appears. For example, in 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. Normally the included file will define a number of variables which then become available for use in the full score file. Tagged sections can be marked in included files to assist in making them usable in different places in a score, see [Section 3.3.2 \[Different editions from one source\]](#), page 340.

Files in the current working directory may be referenced by specifying just the file name after the `\include` command. Files in other locations may be included by giving either a full path reference or a relative path reference (but use the UNIX forward slash, `/`, rather than the DOS/Windows back slash, `\`, as the directory separator.) For example, if `'stuff.ly'` is located one directory higher than the current working directory, use

```
\include "../stuff.ly"
```

or if the included orchestral parts files are all located in a subdirectory called `'parts'` within the current directory, use

```
\include "parts/VI.ly"
\include "parts/VII.ly"
... etc
```

Files which are to be included can also contain `\include` statements of their own. By default, these second-level `\include` statements are not interpreted until they have been brought into the main file, so the file names they specify must all be relative to the directory containing the main file, not the directory containing the included file. However, this behavior can be changed by passing the option `-drelative-includes` option at the command line (or by adding `#{ly:set-option 'relative-includes #t}` at the top of the main input file). With `relative-includes` set, the path for each `\include` command will be taken relative to the file containing that command. This behavior is recommended and it will become the default behavior in a future version of lilypond.

Files can also be included from a directory in a search path specified as an option when invoking LilyPond from the command line. The included files are then specified using just their file name. For example, to compile `'main.ly'` which includes files located in a subdirectory called `'parts'` by this method, `cd` to the directory containing `'main.ly'` and enter

```
lilypond --include=parts main.ly
```

and in `main.ly` write

```
\include "VI.ly"
\include "VII.ly"
... etc
```

Files which are to be included in many scores may be placed in the LilyPond directory `'../ly'`. (The location of this directory is installation-dependent - see [Section “Other sources of information” in *Learning Manual*](#)). These files can then be included simply by naming them on an `\include` statement. This is how the language-dependent files like `'english.ly'` are included.

LilyPond includes a number of files by default when you start the program. These includes are not apparent to the user, but the files may be identified by running `lilypond --verbose` from the command line. This will display a list of paths and files that LilyPond uses, along with

much other information. Alternatively, the more important of these files are discussed in [Section “Other sources of information” in *Learning Manual*](#). These files may be edited, but changes to them will be lost on installing a new version of LilyPond.

Some simple examples of using `\include` are shown in [Section “Scores and parts” in *Learning Manual*](#).

See also

Learning Manual: [Section “Other sources of information” in *Learning Manual*](#), [Section “Scores and parts” in *Learning Manual*](#).

Known issues and warnings

If an included file is given a name which is the same as one in LilyPond’s installation files, LilyPond’s file from the installation files takes precedence.

3.3.2 Different editions from one source

Several mechanisms are available to facilitate the generation of different versions of a score from the same music source. Variables are perhaps most useful for combining lengthy sections of music and/or annotation in various ways, while tags are more useful for selecting one from several alternative shorter sections of music. Whichever method is used, separating the notation from the structure of the score will make it easier to change the structure while leaving the notation untouched.

Using variables

If sections of the music are defined in variables they can be reused in different parts of the score, see [Section “Organizing pieces with variables” in *Learning Manual*](#). For example, an *a cappella* vocal score frequently includes a piano reduction of the parts for rehearsal purposes which is identical to the vocal music, so the music need be entered only once. Music from two variables may be combined on one staff, see [\[Automatic part combining\]](#), page 130. Here is an example:

```
sopranoMusic = \relative c' { a4 b c b8( a) }
altoMusic = \relative g' { e4 e e f }
tenorMusic = \relative c' { c4 b e d8( c) }
bassMusic = \relative c' { a4 gis a d, }
allLyrics = \lyricmode {King of glo -- ry }
<<
  \new Staff = "Soprano" \sopranoMusic
  \new Lyrics \allLyrics
  \new Staff = "Alto" \altoMusic
  \new Lyrics \allLyrics
  \new Staff = "Tenor" {
    \clef "treble_8"
    \tenorMusic
  }
  \new Lyrics \allLyrics
  \new Staff = "Bass" {
    \clef "bass"
    \bassMusic
  }
  \new Lyrics \allLyrics
  \new PianoStaff <<
    \new Staff = "RH" {
      \set Staff.printPartCombineTexts = ##f
```

```

\partcombine
\sopranoMusic
\altoMusic
}
\new Staff = "LH" {
  \set Staff.printPartCombineTexts = ##f
  \clef "bass"
  \partcombine
  \tenorMusic
  \bassMusic
}
>>
>>

```



Separate scores showing just the vocal parts or just the piano part can be produced by changing just the structural statements, leaving the musical notation unchanged.

For lengthy scores, the variable definitions may be placed in separate files which are then included, see [Section 3.3.1 \[Including LilyPond files\]](#), page 339.

Using tags

The `\tag #'partA` command marks a music expression with the name *partA*. Expressions tagged in this way can be selected or filtered out by name later, using either `\keepWithTag #'name` or `\removeWithTag #'name`. The result of applying these filters to tagged music is as follows:

Filter

Tagged music preceded by `\keepWithTag` `#'name`

Result

Untagged music and music tagged with *name* is included; music tagged with any other tag name is excluded.

Tagged music preceded by `\removeWithTag #'name` Untagged music and music tagged with any tag name other than *name* is included; music tagged with *name* is excluded.

Tagged music not preceded by either `\keepWithTag` or `\removeWithTag` All tagged and untagged music is included.

The arguments of the `\tag`, `\keepWithTag` and `\removeWithTag` commands should be a symbol (such as `#'score` or `#'part`), followed by a music expression.

In the following example, we see two versions of a piece of music, one showing trills with the usual notation, and one with trills explicitly expanded:

```
music = \relative g' {
  g8. c32 d
  \tag #'trills { d8.\trill }
  \tag #'expand { \repeat unfold 3 { e32 d } }
  c32 d
}
```

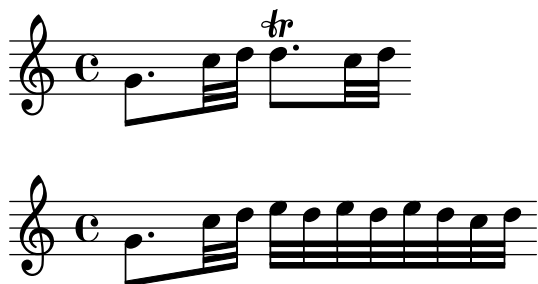
```
\score {
  \keepWithTag #'trills \music
}
\score {
  \keepWithTag #'expand \music
}
```



Alternatively, it is sometimes easier to exclude sections of music:

```
music = \relative g' {
  g8. c32 d
  \tag #'trills { d8.\trill }
  \tag #'expand { \repeat unfold 3 { e32 d } }
  c32 d
}
```

```
\score {
  \removeWithTag #'expand
  \music
}
\score {
  \removeWithTag #'trills
  \music
}
```



Tagged filtering can be applied to articulations, texts, etc. by prepending `-\tag #'your-tag`

to an articulation. For example, this would define a note with a conditional fingering indication and a note with a conditional annotation:

```
c1-\tag #'finger ^4
c1-\tag #'warn ^"Watch!"
```

Multiple tags may be placed on expressions with multiple `\tag` entries:

```
music = \relative c'' {
  \tag #'a \tag #'both { a4 a a a }
  \tag #'b \tag #'both { b4 b b b }
}
<<
\keepWithTag #'a \music
\keepWithTag #'b \music
\keepWithTag #'both \music
>>
```



Multiple `\removeWithTag` filters may be applied to a single music expression to remove several differently named tagged sections:

```
music = \relative c'' {
  \tag #'A { a4 a a a }
  \tag #'B { b4 b b b }
  \tag #'C { c4 c c c }
  \tag #'D { d4 d d d }
}
{
  \removeWithTag #'B
  \removeWithTag #'C
  \music
}
```



Two or more `\keepWithTag` filters applied to a single music expression will cause *all* tagged sections to be removed, as the first filter will remove all tagged sections except the one named, and the second filter will remove even that tagged section.

See also

Learning Manual: [Section “Organizing pieces with variables”](#) in *Learning Manual*.

Notation Reference: [\[Automatic part combining\]](#), page 130, [Section 3.3.1 \[Including LilyPond files\]](#), page 339.

3.3.3 Text encoding

LilyPond uses the character repertoire defined by the Unicode consortium and ISO/IEC 10646. This defines a unique name and code point for the character sets used in virtually all modern languages and many others too. Unicode can be implemented using several different encodings. LilyPond uses the UTF-8 encoding (UTF stands for Unicode Transformation Format) which represents all common Latin characters in one byte, and represents other characters using a variable length format of up to four bytes.

The actual appearance of the characters is determined by the glyphs defined in the particular fonts available - a font defines the mapping of a subset of the Unicode code points to glyphs. LilyPond uses the Pango library to layout and render multi-lingual texts.

LilyPond 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 encoded in 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. All MS Windows systems later than NT use Unicode as their native character encoding, so even Notepad can edit and save a file in UTF-8 format. A more functional alternative for Windows is BabelPad.

If a LilyPond input file containing a non-ASCII character is not saved in UTF-8 format the error message

```
FT_Get_Glyph_Name () error: invalid argument
```

will be generated.

Here is an example showing Cyrillic, Hebrew and Portuguese text:



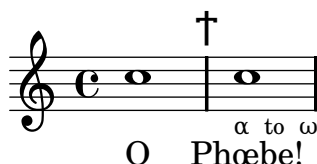
To enter a single character for which the Unicode code point is known but which is not available in the editor being used, use either `\char ##xhhhh` or `\char #dddd` within a `\markup` block, where `hhhh` is the hexadecimal code for the character required and `dddd` is the corresponding decimal value. Leading zeroes may be omitted, but it is usual to specify all four characters in the hexadecimal representation. (Note that the UTF-8 encoding of the code point should *not* be used after `\char`, as UTF-8 encodings contain extra bits indicating the number of octets.) Unicode code charts and a character name index giving the code point in hexadecimal for any character can be found on the Unicode Consortium website, <http://www.unicode.org/>.

For example, `\char ##x03BE` and `\char #958` would both enter the Unicode U+03BE character, which has the Unicode name “Greek Small Letter Xi”.

Any Unicode code point may be entered in this way and if all special characters are entered in this format it is not necessary to save the input file in UTF-8 format. Of course, a font containing all such encoded characters must be installed and available to LilyPond.

The following example shows Unicode hexadecimal values being entered in four places – in a rehearsal mark, as articulation text, in lyrics and as stand-alone text below the score:

```
\score {
  \relative c'' {
    c1 \mark \markup { \char ##x03EE }
    c1_\markup { \tiny { \char ##x03B1 " to " \char ##x03C9 } }
  }
  \addlyrics { 0 \markup { \concat { Ph \char ##x0153 be! } } }
}
\markup { "Copyright 2008--2010" \char ##x00A9 }
```



Copyright 2008--2010 ©

To enter the copyright sign in the copyright notice use:

```
\header {
  copyright = \markup { \char ##x00A9 "2008" }
}
```

3.3.4 Displaying LilyPond notation

Displaying a music expression in LilyPond notation can be done with the music function `\displayLilyMusic` but only when using the command line. For example,

```
{
  \displayLilyMusic \transpose c a, { c4 e g a bes }
}
```

will display

```
{ a,4 cis e fis g }
```

By default, LilyPond will print these messages to the console along with all the other LilyPond compilation messages. To split up these messages and save the results of `\display{STUFF}`, redirect the output to a file.

```
lilypond file.ly >display.txt
```

3.4 Controlling output

3.4.1 Extracting fragments of music

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 defining 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 [Section “Invoking lilypond” in *Application Usage*](#).

3.4.2 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

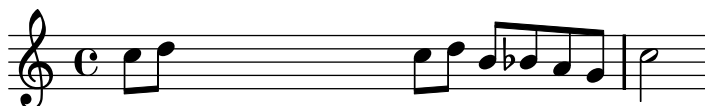
```
showLastLength = R1*5
\score { ... }
```

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. When working on the beginning of a score you have already typeset (e.g. to add a new part), the `showFirstLength` property may be useful as well.

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
  e8 e e e e e e
  \set Score.skipTypesetting = ##f
  c8 d b bes a g c2
}
```



In polyphonic music, `Score.skipTypesetting` will affect all voices and staves, saving even more time.

3.4.3 Alternative output formats

The default output formats for the printed score are Portable Document Format (PDF) and PostScript (PS). Scalable Vector Graphics (SVG), Encapsulated PostScript (EPS) and Portable Network Graphics (PNG) output formats are also available through command line options, see [Section “Command line options for lilypond” in *Application Usage*](#).

3.4.4 Replacing the notation font

Gonville is an alternative to the Feta font used in LilyPond and can be downloaded from:

<http://www.chiark.greenend.org.uk/~sgtatham/gonville/>

Here are a few sample bars of music set in Gonville:



Here are a few sample bars of music set in LilyPond's Feta font:



Installation Instructions for MacOS

Download and extract the zip file. Copy the `lilyfonts` directory to `'SHARE_DIR/lilypond/current'`; for more information, see [Section “Other sources of information” in *Learning Manual*](#). Move the existing `fonts` directory to `fonts_orig` and move the `lilyfonts` directory to `fonts`. Simply move `fonts_orig` back to `fonts` to revert back to Feta.

See also

Learning Manual: [Section “Other sources of information” in *Learning Manual*](#).

Known issues and warnings

Gonville cannot be used to typeset ‘Ancient Music’ notation. Please refer to the author’s website for more information on this and other specifics including licensing of Gonville.

3.5 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.

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.

3.5.1 Creating MIDI files

To create a MIDI output file from a LilyPond input file, add a `\midi` block to a score, for example,

```
\score {
  ...music...
  \midi { }
}
```

If there is a `\midi` block in a `\score` with no `\layout` block, only MIDI output will be produced. When notation is needed too, a `\layout` block must also be present.

```
\score {
  ...music...
  \midi { }
  \layout { }
}
```

Pitches, rhythms, ties, dynamics, and tempo changes are interpreted and translated correctly to the MIDI output. 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 effect of dynamic markings on the MIDI output can be removed completely, see [Section 3.5.2 \[MIDI block\]](#), [page 349](#).

The initial tempo and later tempo changes can be specified with the `\tempo` command within the music notation. These are reflected in tempo changes in the MIDI output. This command will normally result in the metronome mark being printed, but this can be suppressed, see [\[Metronome marks\]](#), [page 151](#). An alternative way of specifying the initial or overall MIDI tempo is described below, see [Section 3.5.2 \[MIDI block\]](#), [page 349](#).

Due to some limitations on Windows, the default extension for MIDI files on Windows is `.mid`. Other operating systems still use the extension `.midi`. If a different extension is preferred, insert the following line at the top-level of the input file, before the start of any `\book`, `\bookpart` or `\score` blocks:

```
#(ly:set-option 'midi-extension "midi")
```

The line above will set the default extension for MIDI files to `.midi`.

Alternatively, this option can also be supplied on the command line:

```
lilypond ... -dmidi-extension=midi lilyFile.ly
```

Instrument names

The MIDI instrument to be used is specified by setting the `Staff.midiInstrument` property to the instrument name. The name should be chosen from the list in [Section A.4 \[MIDI instruments\]](#), [page 455](#).

```
\new Staff {
  \set Staff.midiInstrument = #"glockenspiel"
  ...notes...
}

\new Staff \with {midiInstrument = #"cello"} {
  ...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.

Selected Snippets

Changing MIDI output to one channel per voice

When outputting MIDI, the default behavior is for each staff to represent one MIDI channel, with all the voices on a staff amalgamated. This minimizes the risk of running out of MIDI channels, since there are only 16 available per track.

However, by moving the `Staff_performer` to the `Voice` context, each voice on a staff can have its own MIDI channel, as is demonstrated by the following example: despite being on the same staff, two MIDI channels are created, each with a different `midiInstrument`.

```
\score {
  \new Staff <<
```

```

\new Voice \relative c''' {
  \set midiInstrument = #"flute"
  \voiceOne
  \key g \major
  \time 2/2
  r2 g-"Flute" ~
  g fis ~
  fis4 g8 fis e2 ~
  e4 d8 cis d2
}
\new Voice \relative c'' {
  \set midiInstrument = #"clarinet"
  \voiceTwo
  b1-"Clarinet"
  a2. b8 a
  g2. fis8 e
  fis2 r
}
>>
\layout { }
\midi {
  \context {
    \Staff
    \remove "Staff_performer"
  }
  \context {
    \Voice
    \consists "Staff_performer"
  }
  \context {
    \Score
    tempoWholesPerMinute = #(ly:make-moment 72 2)
  }
}
}

```



Known issues and warnings

Changes in the MIDI volume take place only on starting a note, so crescendi and decrescendi cannot affect the volume of a single note.

Not all midi players correctly handle tempo changes in the midi output. Players that are known to work include MS Windows Media Player and [timidity](#).

3.5.2 MIDI block

A `\midi` block must appear within a score block if MIDI output is required. It is analogous to the layout block, but somewhat simpler. Often, the `\midi` block is left empty, but it can

contain context rearrangements, new context definitions or code to set the values of properties. For example, the following will set the initial tempo exported to a MIDI file without causing a tempo indication to be printed:

```
\score {
  ...music...
  \midi {
    \context {
      \Score
      tempoWholesPerMinute = #(ly:make-moment 72 4)
    }
  }
}
```

In this example the tempo is set to 72 quarter note beats per minute. This kind of tempo specification cannot take a dotted note length as an argument. If one is required, break the dotted note 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)
```

Context definitions follow precisely the same syntax as those within a `\layout` block. Translation modules for sound are called performers. The contexts for MIDI output are defined in ‘`../ly/performer-init.ly`’, see [Section “Other sources of information” in *Learning Manual*](#). For example, to remove the effect of dynamics from the MIDI output, insert the following lines in the `\midi{ }` block.

```
\midi {
  ...
  \context {
    \Voice
    \remove "Dynamic_performer"
  }
}
```

MIDI output is created only when a `\midi` block is included within a score block defined with a `\score` command. If it is placed within an explicitly instantiated score context (i.e. within a `\new Score` block) the file will fail. To solve this, enclose the `\new Score` and the `\midi` commands in a `\score` block.

```
\score {
  \new Score { ...notes... }
  \midi { }
}
```

3.5.3 What goes into the MIDI output?

Supported in MIDI

The following items of notation are reflected in the MIDI output:

- Pitches
- Microtones (See [\[Accidentals\]](#), page 4. Rendering needs a player that supports pitch bend.)
- Chords entered as chord names
- Rhythms entered as note durations, including tuplets
- Tremolos entered without ‘`:[number]`’
- Ties
- Dynamic marks

- Crescendi, decrescendi over multiple notes
- Tempo changes entered with a tempo marking
- Lyrics

Unsupported in MIDI

The following items of notation have no effect on the MIDI output:

- Rhythms entered as annotations, e.g. swing
- Tempo changes entered as annotations with no tempo marking
- Staccato and other articulations and ornamentations
- Slurs and Phrasing slurs
- Crescendi, decrescendi over a single note
- Tremolos entered with ‘:[*number*]’
- Figured bass
- Microtonal chords

3.5.4 Repeats in MIDI

With a few minor additions, all types of repeats can be represented 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 { .. }
}
```

3.5.5 Controlling MIDI dynamics

MIDI dynamics are implemented by the `Dynamic_performer` which lives by default in the `Voice` context. It is possible to control the overall MIDI volume, the relative volume of dynamic markings and the relative volume of different instruments.

Dynamic marks

Dynamic marks are translated to a fixed fraction of the available MIDI volume range. The default fractions range from 0.25 for *ppppp* to 0.95 for *ffff*. The set of dynamic marks and the associated fractions can be seen in ‘`../scm/midi.scm`’, see [Section “Other sources of information” in Learning Manual](#). This set of fractions may be changed or extended by providing a function which takes a dynamic mark as its argument and returns the required fraction, and setting `Score.dynamicAbsoluteVolumeFunction` to this function.

For example, if a *rinforzando* dynamic marking, `\rfz`, is required, this will not by default have any effect on the MIDI volume, as this dynamic marking is not included in the default set. Similarly, if a new dynamic marking has been defined with `make-dynamic-script` that too will not be included in the default set. The following example shows how the MIDI volume for such dynamic markings might be added. The Scheme function sets the fraction to 0.9 if a dynamic mark of `rfz` is found, or calls the default function otherwise.

```
#(define (myDynamics dynamic)
  (if (equal? dynamic "rfz")
      0.9
      (default-dynamic-absolute-volume dynamic)))

\score {
  \new Staff {
    \set Staff.midiInstrument = #"cello"
    \set Score.dynamicAbsoluteVolumeFunction = #myDynamics
    \new Voice {
      \relative c'' {
        a4\pp b c-\rfz
      }
    }
  }
  \layout {}
  \midi {}
}
```



Alternatively, if the whole table of fractions needs to be redefined, it would be better to use the `default-dynamic-absolute-volume` procedure in ‘`../scm/midi.scm`’ and the associated table as a model. The final example in this section shows how this might be done.

Overall MIDI volume

The minimum and maximum overall volume of MIDI dynamic markings is controlled by setting the properties `midiMinimumVolume` and `midiMaximumVolume` at the `Score` level. These properties have an effect only on dynamic marks, so if they are to apply from the start of the score a dynamic mark must be placed there. The fraction corresponding to each dynamic mark is modified with this formula

```

midiMinimumVolume + (midiMaximumVolume - midiMinimumVolume) * fraction

```

In the following example the dynamic range of the overall MIDI volume is limited to the range 0.2 - 0.5.

```

\score {
  <<
    \new Staff {
      \key g \major
      \time 2/2
      \set Staff.midiInstrument = #"flute"
      \new Voice \relative c''' {
        r2 g\mp g fis~
        fis4 g8 fis e2~
        e4 d8 cis d2
      }
    }
    \new Staff {
      \key g \major
      \set Staff.midiInstrument = #"clarinet"
      \new Voice \relative c'' {
        b1\p a2. b8 a
        g2. fis8 e
        fis2 r
      }
    }
  >>
  \layout {}
  \midi {
    \context {
      \Score
      tempoWholesPerMinute = #(ly:make-moment 72 2)
      midiMinimumVolume = #0.2
      midiMaximumVolume = #0.5
    }
  }
}

```



Equalizing different instruments (i)

If the minimum and maximum MIDI volume properties are set in the **Staff** context the relative volumes of the MIDI instruments can be controlled. This gives a basic instrument equalizer, which can enhance the quality of the MIDI output remarkably.

In this example the volume of the clarinet is reduced relative to the volume of the flute. There must be a dynamic mark on the first note of each instrument for this to work correctly.

```

\score {

```

```

<<
  \new Staff {
    \key g \major
    \time 2/2
    \set Staff.midiInstrument = #"flute"
    \set Staff.midiMinimumVolume = #0.7
    \set Staff.midiMaximumVolume = #0.9
    \new Voice \relative c''' {
      r2 g\mp g fis~
      fis4 g8 fis e2~
      e4 d8 cis d2
    }
  }
  \new Staff {
    \key g \major
    \set Staff.midiInstrument = #"clarinet"
    \set Staff.midiMinimumVolume = #0.3
    \set Staff.midiMaximumVolume = #0.6
    \new Voice \relative c'' {
      b1\p a2. b8 a
      g2. fis8 e
      fis2 r
    }
  }
>>
\layout {}
\midi {
  \context {
    \Score
    tempoWholesPerMinute = #(ly:make-moment 72 2)
  }
}

```



Equalizing different instruments (ii)

If the MIDI minimum and maximum volume properties are not set LilyPond will, by default, apply a small degree of equalization to a few instruments. The instruments and the equalization applied are shown in the table *instrument-equalizer-alist* in ‘*../scm/midi.scm*’.

This basic default equalizer can be replaced by setting `instrumentEqualizer` in the `Score` context to a new Scheme procedure which accepts a MIDI instrument name as its only argument and returns a pair of fractions giving the minimum and maximum volumes to be applied to that instrument. This replacement is done in the same way as shown for resetting the `dynamicAbsoluteVolumeFunction` at the start of this section. The default equalizer, *default-instrument-equalizer*, in ‘*../scm/midi.scm*’ shows how such a procedure might be written.

The following example sets the relative flute and clarinet volumes to the same values as the previous example.

```

#(define my-instrument-equalizer-alist '())

#(set! my-instrument-equalizer-alist
  (append
    '(
      ("flute" . (0.7 . 0.9))
      ("clarinet" . (0.3 . 0.6)))
    my-instrument-equalizer-alist))

#(define (my-instrument-equalizer s)
  (let ((entry (assoc s my-instrument-equalizer-alist)))
    (if entry
      (cdr entry))))

\score {
  <<
    \new Staff {
      \key g \major
      \time 2/2
      \set Score.instrumentEqualizer = #my-instrument-equalizer
      \set Staff.midiInstrument = #"flute"
      \new Voice \relative c''' {
        r2 g\mp g fis~
        fis4 g8 fis e2~
        e4 d8 cis d2
      }
    }
    \new Staff {
      \key g \major
      \set Staff.midiInstrument = #"clarinet"
      \new Voice \relative c'' {
        b1\p a2. b8 a
        g2. fis8 e
        fis2 r
      }
    }
  >>
  \layout { }
  \midi {
    \context {
      \Score
      tempoWholesPerMinute = #(ly:make-moment 72 2)
    }
  }
}

```



3.5.6 Percussion in MIDI

Percussion instruments are generally notated in a **DrumStaff** context and when notated in this way they are outputted correctly to MIDI channel 10, but some pitched percussion instruments, like the xylophone, marimba, vibraphone, timpani, etc., are treated like “normal” instruments and music for these instruments should be entered in a normal **Staff** context, not a **DrumStaff** context, to obtain the correct MIDI output.

Some non-pitched percussion sounds included in the general MIDI standard, like melodic tom, taiko drum, synth drum, etc., cannot be reached via MIDI channel 10, so the notation for such instruments should also be entered in a normal **Staff** context, using suitable normal pitches.

Many percussion instruments are not included in the general MIDI standard, e.g. castanets. The easiest, although unsatisfactory, method of producing some MIDI output when writing for such instruments is to substitute the nearest sound from the standard set.

Known issues and warnings

Because the general MIDI standard does not contain rim shots, the sidestick is used for this purpose instead.

4 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.

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 chapter can be placed in either.

4.1 Paper and pages

This section deals with the boundaries that define the area within which music can be printed.

4.1.1 Paper size

Two functions are available for changing the paper size: `set-default-paper-size` and `set-paper-size`. `set-default-paper-size` must be placed in the toplevel scope, and `set-paper-size` must be placed in a `\paper` block:

```
#(set-default-paper-size "a4")

\paper {
  #(set-paper-size "a4")
}
```

`set-default-paper-size` sets the size of all pages, whereas `set-paper-size` only sets the size of the pages that the `\paper` block applies to. For example, if the `\paper` block is at the top of the file, then it will apply the paper size to all pages. If the `\paper` block is inside a `\book`, then the paper size will only apply to that book.

Common paper sizes are available, including `a4`, `letter`, `legal`, and `11x17` (also known as tabloid). Many more paper sizes are supported by default. For details, see ‘`scm/paper.scm`’, and search for the definition of `paper-alist`.

Note: The default paper size is `a4`.

Extra sizes may be added by editing the definition of `paper-alist` in the initialization file ‘`scm/paper.scm`’, however they will be overridden on a subsequent install.

If the symbol `'landscape` is supplied as an argument to `set-default-paper-size`, pages will be rotated by 90 degrees, and wider line widths will be set accordingly.

```
#(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.

See also

Installed Files: ‘`scm/paper.scm`’.

Snippets: [Section “Spacing” in *Snippets*](#).

4.1.2 Page formatting

Margins, headers, and footers and other layout variables are automatically set according to the paper size.

Default margin values are accessible in ‘`ly/paper-defaults-init.ly`’. They apply to the default paper size (a4, unless specified differently) and are scaled accordingly for other paper sizes.

This section lists and describes a number of paper variables that may be altered.

Vertical dimensions

These variables are used to set different vertical dimensions on a page:

`after-title-spacing`

Specifies how to calculate the space between a title (or top-level markup) and the system that follows it. This is an associative list with five components:

- *space* – the amount of stretchable space between the baseline of a title and the center of the staff that follows it;
- *padding* – the minimum amount of whitespace that must be present between a title and the staff that follows it;
- *stretchability* – the ease with which the stretchable space increases when a page is stretched. If this is zero, the distance to the next staff will not stretch at all;
- *minimum-distance* – the minimum distance to place between the baseline of a title and the center of the staff that follows it. This differs from *padding* in that the height of a staff has no effect on the application of *minimum-distance* (whereas the height of a staff is crucial for *padding*).

For example, the default is:

```
after-title-spacing = #'((space . 2) (padding . 0.5))
```

If a page has a ragged bottom, *space* is not stretched. In particular, the resulting distance on such a page is the largest of

- *space*,
- *minimum-distance*, and
- *padding* plus the smallest distance necessary to eliminate overlap.

`before-title-spacing`

Specifies the spacing between a system and the title (or top-level markup) that follows it. The distances are measured from the center of the last staff in the system to the baseline of the title that follows it. See *after-title-spacing*.

`between-scores-system-spacing`

Specifies the spacing between two systems if they are in different scores, but there is no title between them. See *after-title-spacing*.

`between-system-spacing`

Specifies the spacing between the center of the bottom staff of one system and the center of the top staff of the following system. See *after-title-spacing*.

`between-title-spacing`

Specifies the spacing between two titles (or top-level markups). The distances are measured from the baseline of the first title to the baseline of the second. See *after-title-spacing*.

`bottom-margin`

The margin between footer and bottom of the page. Default: 6\mm.

bottom-system-spacing

Specifies the spacing from the center of the last staff (or the baseline of the last top-level markup) to the bottom of the printable area (ie. the top of the bottom margin). See *after-title-spacing*.

top-title-spacing

Specifies the spacing from the top of the printable area (ie. the bottom of the top margin) to the baseline of the title. See *after-title-spacing*.

top-system-spacing

Specifies the spacing from the top of the printable area (ie. the bottom of the top margin) to the center of the first staff. This only takes effect if there is no title at the top of the page (in which case *top-title-spacing* is used instead). See *after-title-spacing*.

paper-height

The height of the page. Default: the height of the current paper size. For details, see [Section 4.1.1 \[Paper size\]](#), page 357.

top-margin

The margin between header and top of the page. Default: 5\mm.

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-init.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-init.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.

See also

Notation Reference: [Section 4.4.2 \[Vertical spacing between systems\]](#), page 377.

Snippets: [Section “Spacing” in *Snippets*](#).

Horizontal dimensions

There are a few variables that determine the horizontal dimensions on a page:

`binding-offset`

The amount `inner-margin` is increased to make sure nothing will be hidden by the binding. Works only with `two-sided` set to true. Default: 0.

`horizontal-shift`

The amount that all systems (including titles and system separators) are shifted to the right. Default: 0.0.

`indent`

The level of indentation for the first system in a score. Default: 15\mm.

`inner-margin`

The margin all pages have at the inner side if they are part of a book. Works only with `two-sided` set to true. Default: 10\mm.

`left-margin`

The margin between the left edge of the page and the beginning of each system. Default: 10\mm.

`line-width`

The width of music systems. Default: `paper-width` minus `left-margin` and `right-margin`.

`outer-margin`

The margin all pages have at the outer side if they are part of a book. Works only with `two-sided` set to true. Default: 20\mm.

`paper-width`

The width of the page. Default: the width of the current paper size. For details, see [Section 4.1.1 \[Paper size\]](#), page 357.

`right-margin`

The margin between the right edge of the page and the end of each system. Default: 10\mm.

short-indent

The level of indentation for all systems in a score besides the first system. Default: 0.

If some values are not set, defaults will be taken. Their exact value is adjusted, depending on the paper size specified. Currently, the following values are affected by this scaling:

- *left-margin*
- *right-margin*
- *top-margin*
- *bottom-margin*
- *head-separation*
- *foot-separation*
- *indent*
- *short-indent*

The settings for `line-width`, `left-margin`, `right-margin` and `paper-width` depend on each other, but they do not have to be specified completely.

```
\paper {
  left-margin = 30\mm
}
```

In this example, only `left-margin` is set. The value for `right-margin` will remain default, `line-width` is calculated automatically.

```
\paper {
  line-width = 150\mm
}
```

Here `left-margin` and `right-margin` will be set to the same value. Therefore, `line-width` is subtracted from `paper-width` and divided by two. That means systems are centered on the page, if only `line-width` is specified.

Some checks occur to ensure the values are set correctly. If the values do not match or systems would run off the page, a warning is printed and default values are set.

```
\paper {
  paper-width = 210\mm
  left-margin = 20\mm
  right-margin = 30\mm
  line-width = 100\mm
}
```

These checks can be avoided by setting `check-consistency` to false.

```
\paper {
  paper-width = 210\mm
  left-margin = 20\mm
  line-width = 200\mm
  check-consistency = ##f
}
```

Note: If `paper-width` is manually set, `line-width`, `left-margin`, `indent`, and `short-indent` may have to be adjusted as well.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

Other layout variables

These variables can be used to adjust page layout in general.

`auto-first-page-number`

The page breaking algorithm is affected by the first page number being odd or even. If set to true, 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. Default: `##f`.

`blank-last-page-force`

The penalty for ending the score on an odd-numbered page. Default: 0.

`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: 5.

`check-consistency`

If set to true, check whether `left-margin`, `right-margin` and `line-width` fit each other. Also make sure that their combination does not exceed the available `paper-width`. Default: `##t`.

`first-page-number`

The value of the page number on the first page. Default: `#1`.

`max-systems-per-page`

The maximum number of systems that will be placed on a page. This is currently supported only by the `ly:optimal-breaking` algorithm. Default: unset.

`min-systems-per-page`

The minimum number of systems that will be placed on a page. This may cause pages to be overfilled if it is made too large. This is currently supported only by the `ly:optimal-breaking` algorithm. Default: unset.

`page-breaking-between-system-spacing`

Tricks the page breaker into thinking that `between-system-spacing` is set to something different than it really is. For example, if `page-breaking-between-system-spacing #'padding` is set to something substantially larger than `between-system-spacing #'padding`, then the page-breaker will put fewer systems on each page. Default: unset.

`page-count`

The number of pages to be used for a score. Default: unset.

`page-limit-inter-system-space`

If set to true, limits space between systems on a page with a lot of space left. Default: `##f`. For details, see [Section 4.4.2 \[Vertical spacing between systems\]](#), page 377.

`page-limit-inter-system-space-factor`

The factor used by `page-limit-inter-system-space`. Default: 1.4. For details, see [Section 4.4.2 \[Vertical spacing between systems\]](#), page 377.

`page-spacing-weight`

The relative importance of page (vertical) spacing and line (horizontal) spacing. High values will make page spacing more important. Default: `#10`.

`print-all-headers`

If set to true, this will print all headers for each `\score` in the output. Normally only the piece and opus header variables are printed. Default: `##f`.

print-first-page-number

If set to true, a page number is printed on the first page. Default: **##f**.

print-page-number

If set to false, page numbers are not printed. Default: **##t**.

ragged-bottom

If set to true, systems will not spread vertically across the page. This does not affect the last page. Default: **##f**.

This should be set to true for pieces that have only two or three systems per page, for example orchestral scores.

ragged-last

If set to true, the last system in the score will not fill the line width. Instead the last system ends at its natural horizontal length. Default: **##f**.

ragged-last-bottom

If set to false, systems will spread vertically across the last page. Default: **##t**.

Pieces that amply fill two pages or more should have this set to true.

It also affects the last page of book parts, ie parts of a book created with `\bookpart` blocks.

ragged-right

If set to true, systems will not fill the line width. Instead, systems end at their natural horizontal length. Default: **##f**.

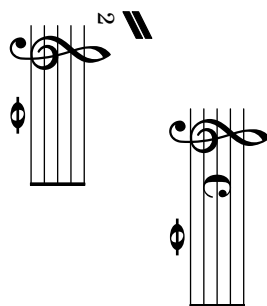
If the score has only one system, the default value is **##t**.

system-separator-markup

A markup object that is inserted between systems. This is often used for orchestral scores. Default: unset.

The markup command `\slashSeparator` is provided as a sensible default, for example

Music engraving by LilyPond 2.13.27—www.lilypond.org



system-count

The number of systems to be used for a score. Default: unset.

systems-per-page

The number of systems that should be placed on each page. This is currently supported only by the `ly:optimal-breaking` algorithm. Default: unset.

two-sided

If set to true, use `inner-margin`, `outer-margin` and `binding-offset` to determine margins depending on whether the page number is odd or even. This overrides `left-margin` and `right-margin`. Default: `##f`.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

Known issues and warnings

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.

4.2 Music layout

4.2.1 Setting the staff size

The default **staff size** is set to 20 points. This may be changed in two ways:

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 [Section “StaffSymbol” in *Internals Reference*](#)) can be used to tune the size for individual staves. The sizes of individual staves are relative to the global size.

See also

Notation Reference: [\[Selecting notation font size\]](#), page 164.

Snippets: [Section “Spacing” in *Snippets*](#).

Known issues and warnings

`layout-set-staff-size` does not change the distance between the staff lines.

4.2.2 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

Notation Reference: [Section 5.1.5 \[Changing context default settings\]](#), page 405.

Snippets: [Section “Spacing” in *Snippets*](#).

4.3 Breaks

4.3.1 Line breaking

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
}

```

For line breaks at regular intervals use `\break` separated by skips and repeated with `\repeat`. For example, this would cause the following 28 measures (assuming 4/4 time) to be broken every 4 measures, and only there:

```

<< \repeat unfold 7 {
  s1 \noBreak s1 \noBreak
  s1 \noBreak s1 \break }
  the real music
>>

```

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 [Section “Spacing” in *Snippets*](#).

Predefined commands

`\break`, `\noBreak`.

See also

Internals Reference: [Section “LineBreakEvent”](#) in *Internals Reference*.

Snippets: [Section “Spacing”](#) in *Snippets*.

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`.

4.3.2 Page breaking

The default page breaking may be overridden 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.

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 [Section 4.4 \[Vertical spacing\]](#), page 373.

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)
}
```

When a book has many scores and pages, the page breaking problem may be difficult to solve, requiring large processing time and memory. To ease the page breaking process, `\bookpart` blocks are used to divide the book into several parts: the page breaking occurs separately on each part. Different page breaking functions may also be used in different book parts.

```
\bookpart {
  \header {
    subtitle = "Preface"
  }
  \paper {
    %% In a part consisting mostly of text,
    %% ly:minimal-breaking may be preferred
    #(define page-breaking ly:minimal-breaking)
  }
  \markup { ... }
  ...
}
\bookpart {
  %% In this part, consisting of music, the default optimal
  %% page breaking function is used.
  \header {
    subtitle = "First movement"
  }
  \score { ... }
  ...
}
```

Predefined commands

`\pageBreak`, `\noPageBreak`.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.3.3 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.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.3.4 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 4.3.2 \[Page breaking\]](#), page 367. 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`.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

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.

4.3.5 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 preferred 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)
}
```

See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.3.6 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 overridden to false, Lily will insert line breaks at explicit `\break` commands and nowhere else. When `page-break-permission` is overridden 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 }
    }
  }
}
```





See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.3.7 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 4.4 \[Vertical spacing\]](#), page 373.

```

\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 }
}
>>
}

```



See also

Notation Reference: [Section 4.4 \[Vertical spacing\]](#), page 373.

Snippets: [Section “Spacing” in *Snippets*](#).

4.4 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.

4.4.1 Vertical spacing inside a system

The height of each system is determined in two steps. First, all of the staves are spaced according to the amount of space available. Then, the non-staff lines (eg. lyrics or chords) are distributed between the staves.

Spacing between staves

Spacing between staves is controlled by the *next-staff-spacing* property of the *VerticalAxisGroup* grob. This property is an alist with four elements: *space*, *minimum-distance*, *padding* and *stretchability*:

- *space* is the size of the stretchable space between the center line of one staff to the center line of the next staff.
- *minimum-distance* provides a lower bound on the final distance between the center line of one staff to the center line of the next staff. That is, if a page has many systems and needs to be compressed, the distance from this staff to the next will never be compressed to less than *minimum-distance*.

- *padding* is the amount of whitespace that must be present between the bottom of one staff and the top of the next. It differs from *minimum-distance* in that the effect of *padding* depends on the height of objects in the staff. For example, *padding* is more likely to come into effect for staves with notes that are far below the staff.
- *stretchability* controls the stretchable space's propensity to stretch when the system is stretched. Large values will cause a system to stretch more, while a value of zero will prevent the space from stretching at all. If unset, *stretchability* defaults to *space - minimum-distance*.

```

#(set-global-staff-size 16)
\new StaffGroup <<
  % Since space is small and there is no minimum-distance, the distance
  % between this staff and the next will be determined by padding.
  \new Staff \with {
    \override VerticalAxisGroup #'next-staff-spacing =
      #'((space . 1) (padding . 1))
  }
  { \clef bass c, }
  % Since space is small and nothing sticks out very far, the distance
  % between this staff and the next will be determined by minimum-distance.
  \new Staff \with {
    \override VerticalAxisGroup #'next-staff-spacing =
      #'((space . 1) (minimum-distance . 12))
  }
  { \clef bass c, }
  % By setting padding to a negative value, staves can be made to collide.
  \new Staff \with {
    \override VerticalAxisGroup #'next-staff-spacing =
      #'((space . 4) (padding . -10))
  }
  { \clef bass c, }
  \new Staff { \clef bass c, }
>>

```



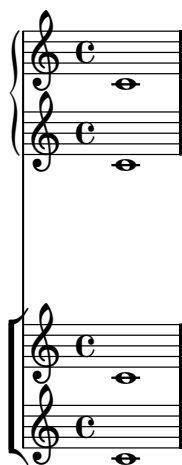
In orchestral and other large scores, it is common to place staves in groups. The space between groups is typically larger than the space between staves of the same group. This spacing can be tweaked with the *StaffGrouper* grob: the default value of *next-staff-spacing* for *VerticalAxisGroup* is a callback function which operates by searching for a *StaffGrouper* grob containing the staff. If it finds a *StaffGrouper* grob and the staff in question is in the middle of a group, it reads the *between-staff-spacing* property of *StaffGrouper* and returns it. If the staff in question is the last staff of a group, the callback reads the *after-last-staff-spacing* property of *StaffGrouper* and returns it. If the callback did not find a *StaffGrouper* grob, it reads *default-next-staff-spacing* from its *VerticalAxisGroup* and returns that.

```

#(set-global-staff-size 16)
<<
  \new PianoStaff \with {
    \override StaffGrouper #'between-staff-spacing #'space = #1
    \override StaffGrouper #'between-staff-spacing #'padding = #0
    \override StaffGrouper #'after-last-staff-spacing #'space = #20
  }
  <<
    \new Staff c'1
    \new Staff c'1
  >>
>>

  \new StaffGroup \with {
    \override StaffGrouper #'between-staff-spacing #'space = #1
    \override StaffGrouper #'between-staff-spacing #'padding = #0
  }
  <<
    \new Staff c'1
    \new Staff c'1
  >>
>>

```



Spacing of non-staff lines

After the positions of the staves are determined, the non-staff lines are distributed between the staves. Each of these lines has a *staff-affinity* property which controls its vertical alignment. For example,

```
\new Lyrics \with { \override VerticalAxisGroup #'staff-affinity = #DOWN }
```

creates a lyrics context that will be placed close to the staff below it. Setting *staff-affinity* to something which is not a number (*#f*, for example) will cause that line to be treated like a staff. Conversely, setting *staff-affinity* for a staff will cause it to be treated like a non-staff.

Non-staff lines admit three properties to control their spacing. Each of these properties is an alist of the same format as *next-staff-spacing*, above.

- If the nearest line in the *staff-affinity* direction is a staff then *inter-staff-spacing* gives the spacing between the non-staff and the staff. If *staff-affinity* is *CENTER*, then *inter-staff-spacing* is used for both directions.
- If the nearest line in the *staff-affinity* direction is a non-staff then *inter-loose-line-spacing* gives the spacing between the two non-staff lines.

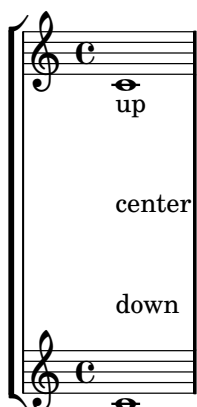
- If the nearest line in the opposite direction to *staff-affinity* is a staff then *non-affinity-spacing* gives the spacing between the non-staff and the staff. This can be used, for example, to require a minimum amount of padding between a Lyrics line and the staff to which it does not belong.

```

#(set-global-staff-size 16)
\layout {
  \context {
    \Lyrics
    % By default, Lyrics are placed close together. Here, we allow them to
    % be stretched more widely.
    \override VerticalAxisGroup
      #'inter-loose-line-spacing #'stretchability = #1000
  }
}

\new StaffGroup
<<
  \new Staff \with {
    \override VerticalAxisGroup #'next-staff-spacing = #'((space . 30)) }
    { c'1 }
  \new Lyrics \with {
    \override VerticalAxisGroup #'staff-affinity = #UP }
    \lyricmode { up }
  \new Lyrics \with {
    \override VerticalAxisGroup #'staff-affinity = #CENTER }
    \lyricmode { center }
  \new Lyrics \with {
    \override VerticalAxisGroup #'staff-affinity = #DOWN }
    \lyricmode { down }
  \new Staff
    { c'1 }
>>

```



See also

Snippets: [Section “Spacing” in *Snippets*](#).

Internals Reference: [Section “VerticalAxisGroup” in *Internals Reference*](#), [Section “VerticalAlignment” in *Internals Reference*](#), [Section “Axis-group-engraver” in *Internals Reference*](#).

Known issues and warnings

Adjacent non-staff lines should have non-increasing *staff-affinity* from top-to-bottom. For example, the behavior of

```
<<
  \new Staff c
  \new Lyrics \with { \override VerticalAxisGroup #'staff-affinity = #DOWN }
  \new Lyrics \with { \override VerticalAxisGroup #'staff-affinity = #UP }
  \new Staff c
>>

is undefined.
```

A non-staff line at the bottom of a system should have *staff-affinity* set to UP. Similarly, a non-staff line at the top of a system should have *staff-affinity* set to DOWN.

4.4.2 Vertical spacing between systems

The mechanisms that control spacing between systems are similar to those that control spacing between staves within a system (see [Section 4.4.1 \[Vertical spacing inside a system\]](#), page 373). The main difference is that the variables to control spacing between systems are set in the `\paper` block, rather than as grob properties. These paper block variables are *between-system-spacing*, *between-scores-system-spacing*, *after-title-spacing*, *before-title-spacing*, *between-title-spacing*, *top-system-spacing*, *top-title-spacing* and *bottom-system-spacing*. Note that these variables ignore non-staff lines. For example, *between-system-spacing* controls the spacing from the middle staff line of the bottom staff from one system to the middle staff line of the top staff of the next system, whether or not there are lyrics below the upper system. See [\[Vertical dimensions\]](#), page 358 for a description of each of these variables.

There are two more `\paper` block variables that affect vertical spacing: if *ragged-bottom* is set to `##t` then no pages will be stretched (which means that neither the space between systems nor the space within systems will be stretched). If *ragged-last-bottom* is set to `##t` then the last page will not be stretched.

See also

Snippets: [Section “Spacing” in Snippets](#).

4.4.3 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 three different settings:

- `X-offset`
- `Y-offset`
- `alignment-distances`

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))

\overrideProperty NonMusicalPaperColumn
  #'line-break-system-details #'((alignment-distances . (15)))

\overrideProperty NonMusicalPaperColumn
  #'line-break-system-details #'((X-offset . 20) (Y-offset . 40)
                                (alignment-distances . (15)))
```

To understand how each of these different settings work, we begin by looking at an example that includes no overrides at all.

The image displays a musical score with three systems, each consisting of two staves. The first system is labeled with a measure number '1' at the beginning. The second system is labeled with a measure number '6' at the beginning. The third system is labeled with a measure number '11' at the beginning. Each system contains a series of eighth notes across four measures. The notation is in treble clef with a common time signature 'C'. The score demonstrates how line and page breaks occur naturally without the use of overrides.

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 4.3.7 \[Using an extra voice for breaks\]](#), page 371.

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:

The image displays three systems of musical notation, each consisting of two staves. The first system is at the top. The second system is preceded by a measure number '6'. The third system is preceded by a measure number '11'. Each system contains six measures of music, with notes evenly spaced across the staves.

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 distances between staves within each system manually. We do this using the `alignment-distances` subproperty of `line-break-system-details`.

The image displays three systems of musical notation, each consisting of two staves. The first system begins at measure 1, the second at measure 6, and the third at measure 11. Each staff contains a sequence of eighth notes, with a bar line appearing after every four notes. The notation is in common time (C) and uses a treble clef. The systems are vertically separated, with the first system at the top, the second in the middle, and the third at the bottom. The measure numbers 1, 6, and 11 are placed to the left of the first staff of each system.

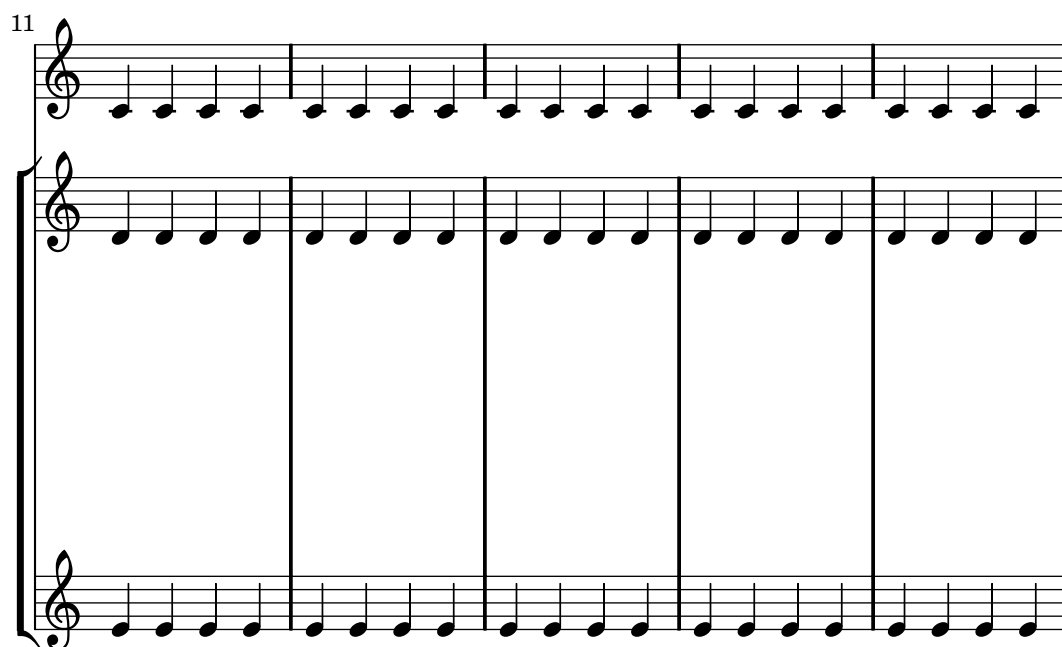
Note that here we assign two different values to the `line-break-system-details` attribute of the `NonMusicalPaperColumn` grob. Though the `line-break-system-details` attribute alist accepts many additional spacing parameters (including, for example, a corresponding `X-offset` pair), we need only set the `Y-offset` and `alignment-distances` pairs to control the vertical startpoint of every system and every staff. Finally, note that `alignment-distances` specifies the vertical positioning of staves but not of staff groups.



First system of musical notation. It consists of three staves. The top staff is a single treble clef staff with a common time signature 'C'. The bottom two staves are grouped by a brace on the left and each has its own treble clef and common time signature 'C'. All staves contain a sequence of eighth notes, with bar lines every four notes.



Second system of musical notation, starting at measure 6. It follows the same three-staff structure as the first system, with a single treble clef staff on top and two staves with a brace on the bottom. All staves contain a sequence of eighth notes, with bar lines every four notes.



Third system of musical notation, starting at measure 11. It follows the same three-staff structure. The top staff contains a sequence of eighth notes. The middle staff (the top staff of the brace group) is empty, with only vertical bar lines. The bottom staff contains a sequence of eighth notes. Bar lines are present in all staves at the same positions.

Some points to consider:

- When using `alignment-distances`, lyrics and other non-staff lines do not count as a staff.
- The units of the numbers passed to `X-offset`, `Y-offset` and `alignment-distances` 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.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.4.4 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
```

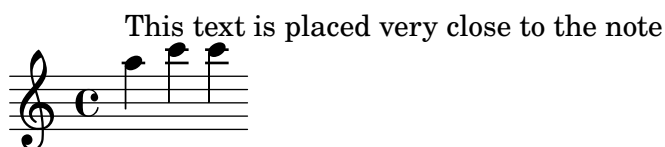


The vertical padding between an outside-staff object and the previously-positioned grobs can be controlled with `outside-staff-padding`.

```
\once \override TextScript #'outside-staff-padding = #0
a'~"This text is placed very close to the note"
```

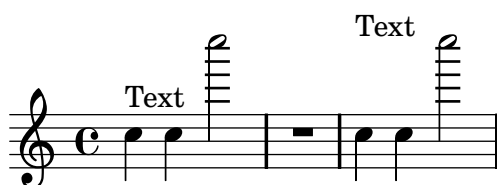
```
\once \override TextScript #'outside-staff-padding = #3
c^"This text is padded away from the previous text"
c^"This text is placed close to the previous text"
```

This text is placed close to the previous text
 This text is padded away from the previous text



By default, outside-staff objects are placed only to avoid a horizontal collision with previously-positioned grobs. This can lead to situations in which objects are placed very close to each other horizontally. The vertical spacing between staves can also be set so that outside staff objects are interleaved. 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
c4^"Text"
c4
c''2
% setting outside-staff-horizontal-padding fixes this
R1
\once \override TextScript #'outside-staff-horizontal-padding = #1
c,,4^"Text"
c4
c''2
```



See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.5 Horizontal spacing

4.5.1 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 [Section “SpacingSpanner” in *Internals Reference*](#) 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 [Section “SpacingSpanner” in *Internals Reference*](#), 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 *Essay on automated music engraving*, it was explained that stem directions influence spacing (see [Section “Optical spacing” in *Essay*](#)). This is controlled with the `stem-spacing-correction` property in the [Section “NoteSpacing” in *Internals Reference*](#), object. These are generated for every [Section “Voice” in *Internals Reference*](#) context. The `StaffSpacing` object (generated in [Section “Staff” in *Internals Reference*](#) 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 4.5.5 \[Proportional notation\]](#), page 388.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

Internals Reference: [Section “SpacingSpanner” in *Internals Reference*](#), [Section “NoteSpacing” in *Internals Reference*](#), [Section “StaffSpacing” in *Internals Reference*](#), [Section “NonMusicalPaperColumn” in *Internals Reference*](#).

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, adjusting the padding value as necessary.

```
\override Score.NonMusicalPaperColumn #'padding = #10
```

No work-around exists for decreasing the amount of space.

4.5.2 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.

See also

Snippets: [Section “Spacing” in *Snippets*](#).

Internals Reference: [Section “SpacingSpanner” in *Internals Reference*](#).

4.5.3 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

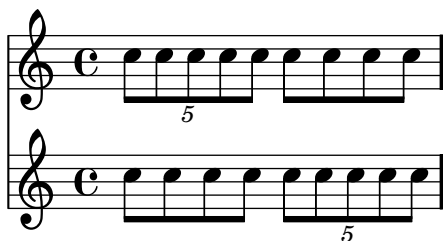
By default, spacing in tuplets depends on various non-duration factors (such as accidentals, clef changes, etc). To disregard such symbols and force uniform equal-duration spacing, use `Score.SpacingSpanner #'uniform-stretching`. This property can only be changed at the beginning of a score,

```
\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] }

```



See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.5.4 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 normal default setting is false, but if the score has only one system the default value is true.

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
}

```

See also

Snippets: [Section “Spacing” in *Snippets*](#).

4.5.5 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 facilitate the placement of timelines or other graphics directly in the score.

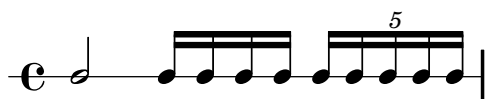
LilyPond supports five different settings for proportional notation, which may be used together or alone:

- `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
    }
  }
>>
```



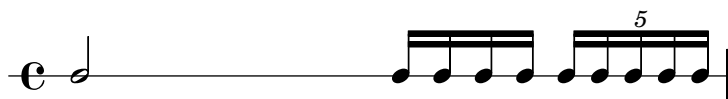
Notice that the half note which begins the measure takes up far less than half of the horizontal space of the measure. Likewise, the sixteenth notes and sixteenth-note quintuplets (or twentieth notes) which end the measure together take up far more than half the horizontal space of the measure.

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 preceded 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
    }
  }
}
>>
```

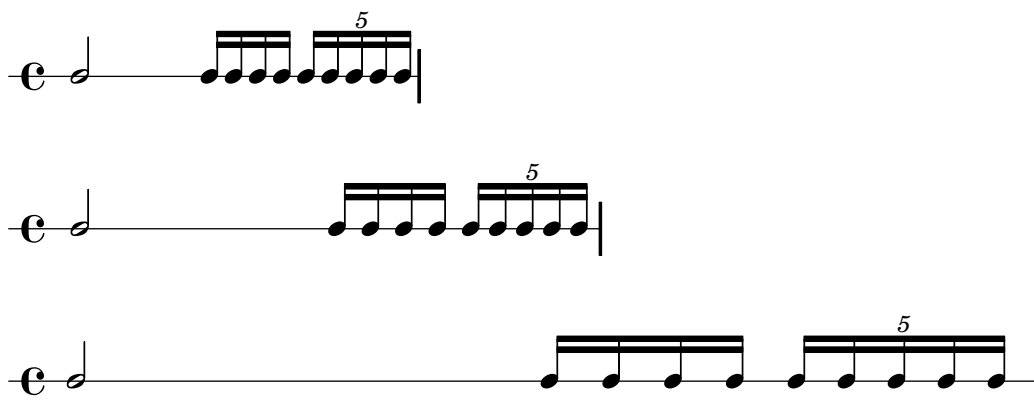
```
\new Score \with {
  proportionalNotationDuration = #(ly:make-moment 1 16)
} <<
  \new RhythmicStaff {
    c'2
    c'16 c'16 c'16 c'16
    \times 4/5 {
      c'16 c'16 c'16 c'16 c'16
    }
  }
}
>>
```

```
\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 than does classical spacing. Proportional spacing provides rhythmic clarity at the expense of horizontal space.

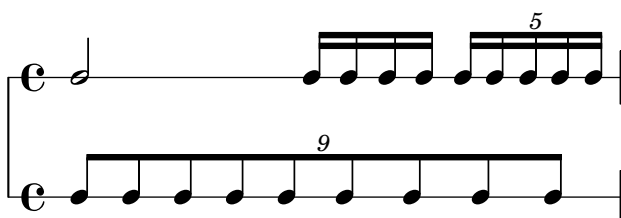
Next we examine how to optimally space overlapping tuplets.

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.

```

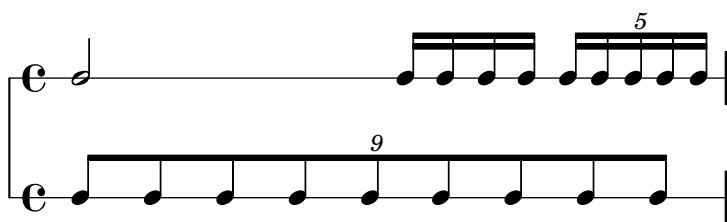
\new Score <<
\new RhythmicStaff {
  c'2
  c'16 c'16 c'16 c'16
  \times 4/5 {
    c'16 c'16 c'16 c'16 c'16
  }
}
\new RhythmicStaff {
  \times 8/9 {
    c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8 c'8
  }
}
>>

```



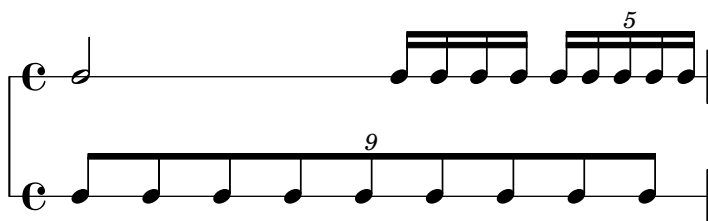
The spacing is bad because the evenly notes of the bottom staff do not stretch uniformly. Classical engraving includes very few complex triplets and so classical engraving rules can generate this type of result. Setting `proportionalNotationDuration` remedies this situation considerably.

```
\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.

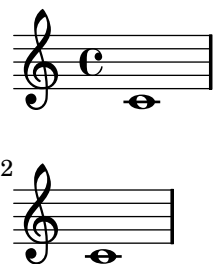
The `SpacingSpanner` is an abstract grob that lives in the `Score` context. As with our settings of `proportionalNotationDuration`, overrides to the `SpacingSpanner` can occur in any of three different places in our input file – in the `Score \with` block, in a `Score \context` block, or in note entry 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 4.5.2 \[New spacing area\]](#), page 385, 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 problematic 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.

See also

Notation Reference: [Section 4.5.2 \[New spacing area\]](#), page 385.

Snippets: [Section “Spacing” in *Snippets*](#).

4.6 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 variables; for more details see the following section, [Section 4.6.1 \[Displaying spacing\]](#), page 394.

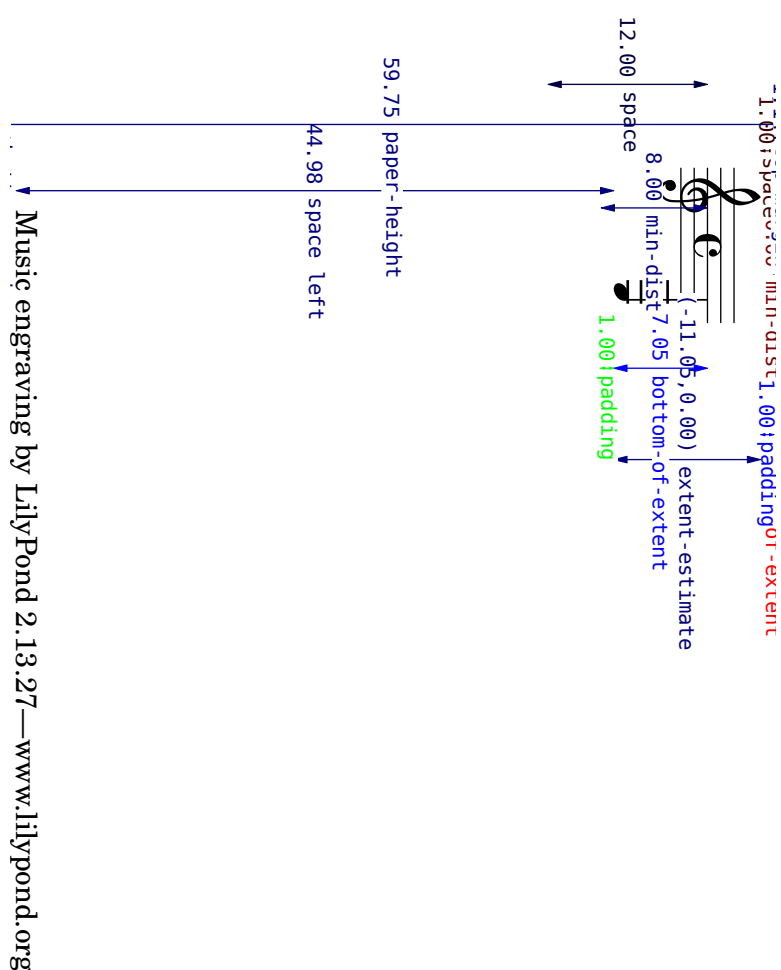
4.6.1 Displaying spacing

To graphically display the dimensions of vertical layout variables that may be altered for page formatting, set `annotate-spacing` in the `\paper` block:

```

#(set-default-paper-size "a6" 'landscape)
\book {
  \score { { c4 } }
  \paper { annotate-spacing = ##t }
}

```



All layout dimensions are displayed in staff-spaces, regardless of the units specified in the `\paper` or `\layout` block. In the above example, `paper-height` has a value of 59.75 `staff-spaces`, and the `staff-size` is 20 points (the default value). Note that:

$$1 \text{ point} = (25.4/72.27) \text{ mm}$$

$$\begin{aligned} 1 \text{ staff-space} &= (\text{staff-size})/4 \text{ pts} \\ &= (\text{staff-size})/4 * \\ &= (25.4/72.27) \text{ mm} \end{aligned}$$

In this case, one `staff-space` is approximately equal to 1.757mm. Thus the `paper-height` measurement of 59.75 `staff-spaces` is equivalent to 105 millimeters, the height of a6 paper in landscape orientation. The pairs (a,b) are intervals, where a is the lower edge and b the upper edge of the interval.

See also

Notation Reference: [Section 4.2.1 \[Setting the staff size\]](#), page 365.

Snippets: [Section “Spacing” in *Snippets*](#).

4.6.2 Changing spacing

The output of `annotate-spacing` reveals vertical dimensions in great detail. For details about modifying margins and other layout variables, see [Section 4.1.2 \[Page formatting\]](#), page 358.

Other than margins, there are a few other options to save space:

- Force systems to move as close together as possible (to fit as many systems as possible onto a page) while being spaced so that there is no blank space at the bottom of the page.

```
\paper {
  between-system-spacing = #'((padding . 0) (space . 0.1))
  ragged-last-bottom = ##f
  ragged-bottom = ##f
}
```

- Force the number of systems. This can help in two ways. Just setting a value, even the same value as the number of systems being typeset by default, will sometimes cause more systems to be fitted onto each page, as an estimation step is then bypassed, giving a more accurate fit to each page. Also, forcing an actual reduction in the number of systems may save a further page. For example, if the default layout has 11 systems, the following assignment will force a layout with 10 systems.

```
\paper {
  system-count = #10
}
```

- Avoid (or reduce) objects that 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. For example, dynamics that ‘stick out’ of a system can be moved closer to the staff:

```
e4 c g\f c
\override DynamicText #'extra-offset = #'(-2.2 . 2.0)
e4 c g\f c
```



- Alter the horizontal spacing via `SpacingSpanner`. For more details, see [Section 4.5.3 \[Changing horizontal spacing\]](#), page 385. The following example illustrates the default spacing:

```
\score {
  \relative c'' {
    g4 e e2 |
    f4 d d2 |
    c4 d e f |
    g4 g g2 |
    g4 e e2 |
  }
}
```



The next example modifies `common-shortest-duration` from a value of $1/4$ to $1/2$. The quarter note is the most common and shortest duration in this example, so by making this duration longer, a ‘squeezing’ effect occurs:

```
\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)
    }
  }
}
```



The `common-shortest-duration` property cannot be modified dynamically, so it must always be placed in a `\context` block so that it applies to the whole score.

See also

Notation Reference: [Section 4.1.2 \[Page formatting\]](#), page 358, [Section 4.5.3 \[Changing horizontal spacing\]](#), page 385.

Snippets: [Section “Spacing” in *Snippets*](#).

5 Changing defaults

The purpose of LilyPond’s design is to provide the finest quality output by default. Nevertheless, it may happen that you need to change this default layout. The layout is controlled through a large number of ‘knobs and switches’ collectively called ‘properties’. A tutorial introduction to accessing and modifying these properties can be found in the Learning Manual, see [Section “Tweaking output” in *Learning Manual*](#). This should be read first. This chapter covers similar ground, but in a style more appropriate to a reference manual.

The definitive description of the controls available for tuning can be found in a separate document: [Section “the Internals Reference” in *Internals Reference*](#). That manual lists all the variables, functions and options available in LilyPond. It is written as a HTML document, which is available [on-line](#), and is also included with the LilyPond documentation package.

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

5.1 Interpretation contexts

This section describes what contexts are, and how to modify them.

See also

Learning Manual: [Section “Contexts and engravers” in *Learning Manual*](#).

Installed files: `‘ly/engraver-init.ly’`, `‘ly/performer-init.ly’`.

Snippets: [Section “Contexts and engravers” in *Snippets*](#).

Internals Reference: [Section “Contexts” in *Internals Reference*](#), [Section “Engravers and Performers” in *Internals Reference*](#).

5.1.1 Contexts explained

Contexts are arranged hierarchically:

Score - the master of all contexts

This is the top level notation context. No other context can contain a Score context. By default the Score context handles the administration of time signatures and makes sure that items such as clefs, time signatures, and key-signatures are aligned across staves.

A Score context is instantiated implicitly when a `\score {...}` or `\layout {...}` block is processed, or explicitly when a `\new Score` command is executed.

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 bar lines of the contained staves are not connected vertically.

GrandStaff

¹ [Section “Scheme tutorial” in *Extending*](#), contains a short tutorial on entering numbers, lists, strings, and symbols in Scheme.

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 support for instrument names to the left of each system.

Intermediate-level contexts - staves

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.

DrumStaff

Handles typesetting for percussion. Can contain **DrumVoice**

VaticanaStaff

Same as **Staff**, except that it is designed for typesetting a piece in gregorian style.

MensuralStaff

Same as **Staff**, except that it is designed for typesetting a piece in mensural style.

Bottom-level contexts - voices

Voice-level contexts initialise certain properties and start appropriate engravers. Being bottom-level contexts, they cannot contain other contexts.

Voice

Corresponds to a voice on a staff. This context handles the conversion of dynamic signs, stems, beams, super- and sub-scripts, slurs, ties, and rests. You have to instantiate this explicitly if you require multiple voices on the same staff.

VaticanaVoice

Same as **Voice**, except that it is designed for typesetting a piece in gregorian style.

MensuralVoice

Same as **Voice**, with modifications for typesetting a piece in mensural style.

Lyrics

Corresponds to a voice with lyrics. Handles the printing of a single line of lyrics.

DrumVoice

The voice context used in a percussion staff.

FiguredBass

The context in which **BassFigure** objects are created from input entered in `\figuremode` mode.

TabVoice

The voice context used within a **TabStaff** context. Usually left to be created implicitly.

CueVoice

A voice context used to render notes of a reduced size, intended primarily for adding cue notes to a staff, see [\[Formatting cue notes\]](#), page 161. Usually left to be created implicitly.

ChordNames

Typesets chord names.

5.1.2 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 “Running a function on all layout objects” in Extending](#)). 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
```

5.1.3 Keeping contexts alive

Contexts are usually terminated at the first musical moment in which they have nothing to do. So *Voice* contexts die as soon as they contain no events; *Staff* contexts die as soon as all the *Voice* contexts within them contain no events; etc. This can cause difficulties if earlier contexts which have died have to be referenced, for example, when changing staves with `\change` commands, associating lyrics with a voice with `\lyricsto` commands, or when adding further musical events to an earlier context.

There is an exception to this general rule: just one of the *Voice* contexts in a *Staff* context or in a `<<...>>` construct will always persist to the end of the enclosing *Staff* context or `<<...>>` construct, even though there may be periods when it has nothing to do. The context to persist in this way will be the first one encountered in the first enclosed `{...}` construct, ignoring any in enclosed `<<...>>` constructs.

Any context can be kept alive by ensuring it has something to do at every musical moment. *Staff* contexts are kept alive by ensuring one of their voices is kept alive. One way of doing this is to add spacer rests to a voice in parallel with the real music. These need to be added to every *Voice* context which needs to be kept alive. If several voices are to be used sporadically it is safest to keep them all alive rather than attempting to rely on the exceptions mentioned above.

In the following example, both voice A and voice B are kept alive in this way for the duration of the piece:

```
musicA = \relative c'' { d4 d d d }
musicB = \relative c'' { g4 g g g }
keepVoicesAlive = {
  <<
    \new Voice = "A" { s1*5 } % Keep Voice "A" alive for 5 bars
    \new Voice = "B" { s1*5 } % Keep Voice "B" alive for 5 bars
  >>
}

music = {
  \context Voice = "A" {
    \voiceOneStyle
    \musicA
  }
  \context Voice = "B" {
```

```

    \voiceTwoStyle
    \musicB
  }
  \context Voice = "A" { \musicA }
  \context Voice = "B" { \musicB }
  \context Voice = "A" { \musicA }
}

\score {
  \new Staff <<
    \keepVoicesAlive
    \music
  >>
}

```



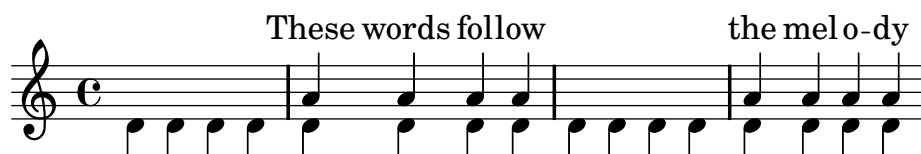
The following example shows how a sporadic melody line with lyrics might be written using this approach. In a real situation the melody and accompaniment would consist of several different sections, of course.

```

melody = \relative c'' { a4 a a a }
accompaniment = \relative c' { d4 d d d }
words = \lyricmode { These words fol -- low the mel -- o -- dy }
\score {
  <<
    \new Staff = "music" {
      <<
        \new Voice = "melody" {
          \voiceOne
          s1*4 % Keep Voice "melody" alive for 4 bars
        }
        {
          \new Voice = "accompaniment" {
            \voiceTwo
            \accompaniment
          }
          <<
            \context Voice = "melody" { \melody }
            \context Voice = "accompaniment" { \accompaniment }
          >>
          \context Voice = "accompaniment" { \accompaniment }
          <<
            \context Voice = "melody" { \melody }
            \context Voice = "accompaniment" { \accompaniment }
          >>
        }
      >>
    }
    \new Lyrics \with { alignAboveContext = #"music" }
    \lyricsto "melody" { \words }
  }
}

```

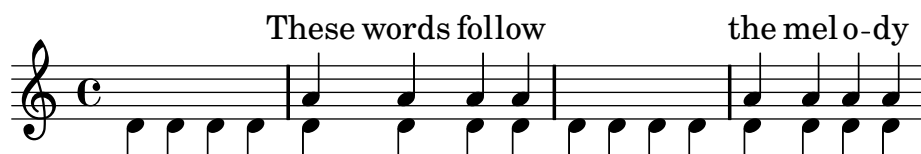
```
>>
}
```



An alternative way, which may be better in many circumstances, is to keep the melody line alive by simply including spacer notes to line it up correctly with the accompaniment:

```
melody = \relative c'' {
  s1 % skip a bar
  a4 a a a
  s1 % skip a bar
  a4 a a a
}
accompaniment = \relative c' {
  d4 d d d
  d4 d d d
  d4 d d d
  d4 d d d
}
words = \lyricmode { These words fol -- low the mel -- o -- dy }

\score {
  <<
    \new Staff = "music" {
      <<
        \new Voice = "melody" {
          \voiceOne
          \melody
        }
        \new Voice = "accompaniment" {
          \voiceTwo
          \accompaniment
        }
      >>
    }
    \new Lyrics \with { alignAboveContext = #"music" }
    \lyricsto "melody" { \words }
  >>
}
```



5.1.4 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* \mapsto Translation \mapsto Engravers. Every context described in *Internals Reference* \mapsto Translation \mapsto 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. More sophisticated methods of blanking objects are shown in [Section “Visibility and color of objects”](#) in *Learning Manual*.

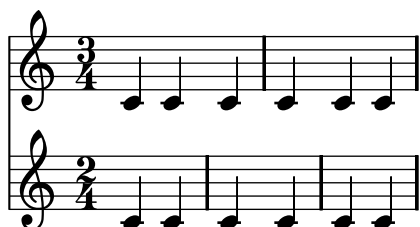
The next example shows a practical application. Bar lines and time signatures are normally synchronized across the score. This is done by the `Timing_translator` and `Default_bar_line_engraver`. This plug-in keeps an administration of time signature, location within the measure, etc. By moving these 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
  }
>>

```



Known issues and warnings

Usually the order in which the engravers are specified does not matter, but in a few special cases the order is important, for example where one engraver writes a property and another reads it, or where one engraver creates a grob and another must process it. The order in which the engravers are specified is the order in which they are called to carry out their processing.

The following orderings are important: the `Bar_engraver` must normally be first, and the `New_fingering_engraver` must come before the `Script_column_engraver`. There may be others with ordering dependencies.

5.1.5 Changing context default settings

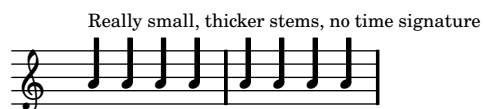
The context settings which are to be used by default in `Score`, `Staff` and `Voice` contexts may be specified in a `\layout` block, as illustrated in the following example. The `\layout` block should be placed within the `\score` block to which it is to apply, but outside any music.

Note that the `\set` command itself and the context must be omitted when the context default values are specified in this way:

```

\score {
  \relative c'' {
    a4^"Really small, thicker stems, no time signature" a a a
    a a a a
  }
  \layout {
    \context {
      \Staff
      fontSize = #-4
      \override Stem #'thickness = #4.0
      \remove "Time_signature_engraver"
    }
  }
}

```



In this example, the `\Staff` command specifies that the subsequent specifications are to be applied to all staves within this score block.

Modifications can be made to the `Score` context or all `Voice` contexts in a similar way.

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

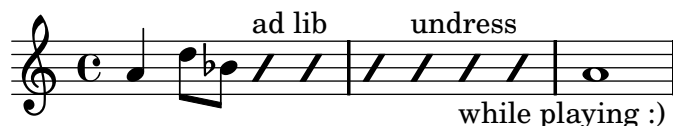
```
\layout {
  \context {
    \RemoveEmptyStaffContext

    \override Stem #'thickness = #4.0
  }
}
```

5.1.6 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) `Voices` 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 Section “Pitch_squash_engraver” in *Internals Reference* modifies note heads (created by Section “Note_heads_engraver” in *Internals Reference*) 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 `Voices`. 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
}

```

5.1.7 Aligning contexts

New contexts may be aligned above or below existing contexts. This could be useful in setting up a vocal staff (Section “Vocal ensembles” in *Learning Manual*) and in ossia,



Contexts like `PianoStaff` can contain other contexts nested within them. Contexts which are acceptable for nesting are defined by the “accepts” list of a context. Contexts which are not in this list are placed below the outer context in the printed score. For example, the `PianoStaff` context is defined by default to accept `Staff` and `FiguredBass` contexts within it, but not (for example) a `Lyrics` context. So in the following structure the lyrics are placed below the piano staff rather than between the two staves:

```

\new PianoStaff
<<
  \new Staff { e4 d c2 }
  \addlyrics { Three blind mice }
  \new Staff {
    \clef "bass"
    { c,1 }
  }
>>

```



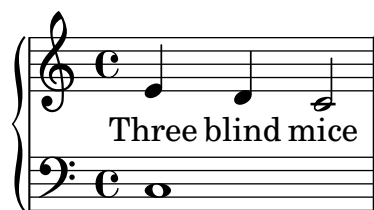
Three blind mice

The “accepts” list of a context can be modified to include additional nested contexts, so if we wanted the lyrics to appear between the two staves we could use:

```

\new PianoStaff \with { \accepts Lyrics }
<<
  \new Staff { e4 d c2 }
  \addlyrics { Three blind mice }
  \new Staff {
    \clef "bass"
    { c,1 }
  }
>>

```



The opposite of `\accepts` is `\denies`; this removes a context from the “accepts” list.

5.2 Explaining the Internals Reference

5.2.1 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 [\[Fingering instructions\]](#), page 165), you will notice:

See also

Internals Reference: [Section “Fingering” in *Internals Reference*](#).

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 [Section “Fingering” in *Internals Reference*](#). At the top of the page, you will see

Fingering objects are created by: [Section “Fingering-engraver” in *Internals Reference*](#) and [Section “New_fingering_engraver” in *Internals Reference*](#).

By following related links inside the program reference, we can follow the flow of information within the program:

- [Section “Fingering” in *Internals Reference*](#): Section “Fingering” in *Internals Reference* objects are created by: [Section “Fingering-engraver” in *Internals Reference*](#)
- [Section “Fingering-engraver” in *Internals Reference*](#): Music types accepted: [Section “fingering-event” in *Internals Reference*](#)
- [Section “fingering-event” in *Internals Reference*](#): Music event type `fingering-event` is in Music expressions named [Section “FingeringEvent” in *Internals Reference*](#)

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 [Section “Translation” in *Internals Reference*](#), and the [Section “Backend” in *Internals Reference*](#). Every chapter lists all the definitions used and all properties that may be tuned.

5.2.2 Layout interfaces

The HTML page that we found in the previous section describes the layout object called [Section “Fingering” in *Internals Reference*](#). 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 [Section “grob-interface” in *Internals Reference*](#).

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 *interfaces*, which are listed on the [Section “Fingering” in *Internals Reference*](#) page at the bottom

This object supports the following interfaces: [Section “item-interface” in *Internals Reference*](#), [Section “self-alignment-interface” in *Internals Reference*](#), [Section “side-position-interface” in *Internals Reference*](#), [Section “text-interface” in *Internals Reference*](#), [Section “text-script-interface” in *Internals Reference*](#), [Section “font-interface” in *Internals Reference*](#), [Section “finger-interface” in *Internals Reference*](#), and [Section “grob-interface” in *Internals Reference*](#).

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 [Section “Other sources of information” in *Learning Manual*](#)) ‘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.

5.2.3 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 [Section “Fingering-engraver”](#) in *Internals Reference* plug-in says

Fingering-engraver is part of contexts: ... [Section “Voice”](#) in *Internals Reference*

5.2.4 Naming conventions

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

Questions to be answered:

- Which of these are conventions and which are rules?
- Which are rules of the underlying language, and which are LP-specific?

5.3 Modifying properties

5.3.1 Overview of modifying properties

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.

There are two different kinds of properties stored in contexts: context properties and grob properties. Context properties are properties that apply to the context as a whole and control how the context itself is displayed. In contrast, grob properties apply to specific grob types that will be displayed in the context.

The `\set` and `\unset` commands are used to change values for context properties. The `\override` and `\revert` commands are used to change values for grob properties.

See also

Internals: Section “Backend” in *Internals Reference*, Section “All layout objects” in *Internals Reference*, Section “OverrideProperty” in *Internals Reference*, Section “RevertProperty” in *Internals Reference*, Section “PropertySet” in *Internals Reference*.

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.

5.3.2 The `\set` command

Each context has a set of *properties*, variables contained in that context. Context properties are changed with the `\set` command, which has the following syntax:

```
\set context.property = #value
```

value is a Scheme object, which is why it must be preceded by the `#` character.

Contexts properties are usually named in `studlyCaps`. They mostly control the translation from music to notation, e.g. `localKeySignature` (for determining whether to print accidentals), or `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`.

For example, multimeasure rests will be combined into a single bar if the context property `skipBars` is set to `#t`:

```
R1*2
\set Score.skipBars = ##t
R1*2
```



If the *context* argument is left out, then the property will be set in the current bottom context (typically `ChordNames`, `Voice`, `TabVoice`, or `Lyrics`).

```
\set Score.autoBeaming = ##f
<<
{
  e8 e e e
  \set autoBeaming = ##t
  e8 e e e
} \ {
  c8 c c c c8 c c c
}
>>
```



The change is applied ‘on-the-fly’, during the music, so that the setting only affects the second group of eighth notes.

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 default bottom context, in this case `Voice`, will have no effect, because `skipBars` is a property of the `Score` context.

```
R1*2
\set skipBars = ##t
R1*2
```



Contexts are hierarchical, so if an enclosing context was specified, for example `Staff`, then the change would also apply to all `Voices` in the current staff.

The `\unset` command:

```
\unset context.property
```

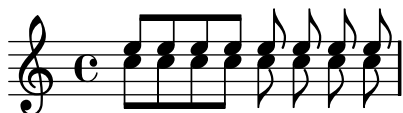
is used to remove the definition of *property* from *context*. This command removes the definition only if it is set in *context*. Properties that have been set in enclosing contexts will not be altered by an unset in an enclosed context:

```
\set Score.autoBeaming = ##t
<<
{
  \unset autoBeaming
  e8 e e e
}
```

```

\unset Score.autoBeaming
e8 e e e
} \ {
c8 c c c c8 c c c
}
>>

```



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 if the current bottom context is `Voice`.

Preceding a `\set` command by `\once` makes the setting apply to only a single time-step:

```

c4
\once \set fontSize = #4.7
c4
c4

```



A full description of all available context properties is in the internals reference, see Translation \mapsto Tunable context properties.

See also

Internals Reference:

Section “Tunable context properties” in *Internals Reference*.

5.3.3 The `\override` command

There is a special type of context property: the grob description. Grob descriptions are named in **StudlyCaps** (starting with capital letters). They contain the ‘default settings’ for a particular kind of grob as an association list. See ‘`scm/define-grobs.scm`’ to see the settings for each grob description. Grob descriptions are modified with `\override`.

`\override` is actually a shorthand;

```

\override context.GrobName #'property = #value

```

is more or less equivalent to

```

\set context.GrobName =
  #(cons (cons 'property value)
    <previous value of context.GrobName>)

```

The value of `context.GrobName` (the alist) is used to initialize the properties of individual grobs. Grobs 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.

For example, we can increase the thickness of a note stem by overriding the **thickness** property of the **Stem** object:

```
c4 c
\override Voice.Stem #'thickness = #3.0
c4 c
```



If no context is specified in an `\override`, the bottom context is used:

```
{ \override Staff.Stem #'thickness = #3.0
  <<
    {
      e4 e
      \override Stem #'thickness = #0.5
      e4 e
    } \ {
      c4 c c c
    }
  >>
}
```



The effects of `\override` can be undone by `\revert`:

```
c4
\override Voice.Stem #'thickness = #3.0
c4 c
\revert Voice.Stem #'thickness
c4
```



The effects of `\override` and `\revert` apply to all grobs in the affected context from the current time forward:

```
{
  <<
    {
      e4
      \override Staff.Stem #'thickness = #3.0
      e4 e e
    } \ {
      c4 c c
      \revert Staff.Stem #'thickness
      c4
    }
  >>
}
```



`\once` can be used with `\override` to affect only the current time step:

```
{
  <<
  {
    \override Stem #'thickness = #3.0
    e4 e e e
  } \ {
    c4
    \once \override Stem #'thickness = #3.0
    c4 c c
  }
  >>
}
```



See also

Internals Reference: [Section “Backend” in *Internals Reference*](#)

5.3.4 The `\tweak` command

Changing grob properties with `\override` causes the changes to apply to all of the given grobs in the context at the moment the change applies. Sometimes, however, it is desirable to have changes apply to just one grob, rather than to all grobs in the affected context. This is accomplished with the `\tweak` command, which has the following syntax:

```
\tweak #'grob-property #value
```

The `\tweak` command applies to the object that immediately follows `value` in the music stream.

For an introduction to the syntax and uses of the `tweak` command see [Section “Tweaking methods” in *Learning Manual*](#).

When several similar items are placed at the same musical moment, the `\override` command cannot be used to modify just one of them – this is where the `\tweak` command must be used. Items which may appear more than once at the same musical moment include the following:

- note heads of notes inside a chord
- articulation signs on a single note
- ties between notes in a chord
- tuplet brackets starting at the same time

In this example, the color of one note head and the type of another note head are modified within a single chord:

```
< c
\tweak #'color #red
d
g
\tweak #'duration-log #1
```

```
a
> 4
```



`\tweak` can be used to modify slurs:

```
c-\tweak #'thickness #5 ( d e f)
```



For the `\tweak` command to work, it must remain immediately adjacent to the object to which it is to apply after the input file has been converted to a music stream. At times, LilyPond may insert additional items into the music stream during the parsing process. For example, when a note that is not explicitly part of a chord will be placed in a chord by LilyPond, so notes to be modified with `\tweak` must be placed inside a chord construct:

```
\tweak #'color #red c4
<\tweak #'color #red c>4
```



The `\tweak` command cannot be used to modify any item that does not appear explicitly in the input file. In particular it cannot be used to modify stems, beams or accidentals directly, since these are generated later by note heads, rather than by music elements in the input stream. Nor can `\tweak` be used to modify clefs or time signatures, since these become separated from any preceding `\tweak` command in the input stream by the automatic insertion of extra elements required to specify the context.

Several `\tweak` commands may be placed before a notational element – all affect it:

```
c
-\tweak #'style #'dashed-line
-\tweak #'dash-fraction #0.2
-\tweak #'thickness #3
-\tweak #'color #red
\glissando
f'
```



The music stream which is generated from a section of an input file, including any automatically inserted elements, may be examined, see [Section “Displaying music expressions” in *Extending*](#). This may be helpful in determining what may be modified by a `\tweak` command, or in determining how to adjust the input to make a `\tweak` apply.

See also

Learning Manual: [Section “Tweaking methods” in *Learning Manual*](#).

Extending: [Section “Displaying music expressions” in *Extending*](#).

Known issues and warnings

The `\tweak` command cannot be used inside a variable.

The `\tweak` commands cannot be used in `\lyricmode`.

The `\tweak` command will apply to only the first of several generated ties in a chord.

5.3.5 `\set` vs. `\override`

TODO – This section is probably unnecessary now.

5.4 Useful concepts and properties

5.4.1 Input modes

The way in which the notation contained within an input file is interpreted is determined by the current input mode.

Chord mode

This is activated with the `\chordmode` command, and causes input to be interpreted with the syntax of chord notation, see [Section 2.7 \[Chord notation\]](#), page 278. Chords are rendered as notes on a staff.

Chord mode is also activated with the `\chords` command. This also creates a new `ChordNames` context and causes the following input to be interpreted with the syntax of chord notation and rendered as chord names in the `ChordNames` context, see [\[Printing chord names\]](#), page 284.

Drum mode

This is activated with the `\drummode` command, and causes input to be interpreted with the syntax of drum notation, see [\[Basic percussion notation\]](#), page 265.

Drum mode is also activated with the `\drums` command. This also creates a new `DrumStaff` context and causes the following input to be interpreted with the syntax of drum notation and rendered as drum symbols on a drum staff, see [\[Basic percussion notation\]](#), page 265.

Figure mode

This is activated with the `\figuremode` command, and causes input to be interpreted with the syntax of figured bass, see [\[Entering figured bass\]](#), page 291.

Figure mode is also activated with the `\figures` command. This also creates a new `FiguredBass` context and causes the following input to be interpreted with the figured bass syntax and rendered as figured bass symbols in the `FiguredBass` context, see [\[Introduction to figured bass\]](#), page 291.

Fret and tab modes

There are no special input modes for entering fret and tab symbols.

To create tab diagrams, enter notes or chords in note mode and render them in a `TabStaff` context, see [\[Default tablatures\]](#), page 233.

To create fret diagrams above a staff, you have two choices. You can either use the `FretBoards` context (see [\[Automatic fret diagrams\]](#), page 258 or you can enter them as a markup above the notes using the `\fret-diagram` command (see [\[Fret diagram markups\]](#), page 241).

Lyrics mode

This is activated with the `\lyricmode` command, and causes input to be interpreted as lyric syllables with optional durations and associated lyric modifiers, see [Section 2.1 \[Vocal music\]](#), page 198.

Lyric mode is also activated with the `\addlyrics` command. This also creates a new `Lyrics` context and an implicit `\lyricsto` command which associates the following lyrics with the preceding music.

Markup mode

This is activated with the `\markup` command, and causes input to be interpreted with the syntax of markup, see [Section A.8 \[Text markup commands\]](#), page 474.

Note mode

This is the default mode or it may be activated with the `\notemode` command. Input is interpreted as pitches, durations, markup, etc and typeset as musical notation on a staff.

It is not normally necessary to specify note mode explicitly, but it may be useful to do so in certain situations, for example if you are in lyric mode, chord mode or any other mode and want to insert something that only can be done with note mode syntax.

For example, to indicate dynamic markings for the verses of a choral pieces it is necessary to enter note mode to interpret the markings:

```
{ c4 c4 c4 c4 }
\addlyrics {
  \notemode{\set stanza = \markup{ \dynamic f 1. } }
  To be sung loudly
}
\addlyrics {
  \notemode{\set stanza = \markup{ \dynamic p 2. } }
  To be sung quietly
}
```



5.4.2 Direction and placement

In typesetting music the direction and placement of many items is a matter of choice. For example, the stems of notes can be directed up or down; lyrics, dynamics, and other expressive marks may be placed above or below the staff; text may be aligned left, right or center; etc. Most of these choices may be left to be determined automatically by LilyPond, but in some cases it may be desirable to force a particular direction or placement.

Articulation direction indicators

By default some directions are always up or always down (e.g. dynamics or fermata), while other things can alternate between up or down based on the stem direction (like slurs or accents).

The default action may be overridden by prefixing the articulation by a *direction indicator*. Three direction indicators are available: `^` (meaning “up”), `_` (meaning “down”) and `-` (meaning “use default direction”). The direction indicator can usually be omitted, in which case `-` is assumed, but a direction indicator is **always** required before

- `\tweak` commands
- `\markup` commands
- `\tag` commands

- string markups, e.g. `-"string"`
- fingering instructions, e.g. `-1`
- articulation shortcuts, e.g. `-. , -> , --`

Direction indicators affect only the next note:

```
c2( c)
c2_( c)
c2( c)
c2^( c)
```



The direction property

The position or direction of many layout objects is controlled by the `direction` property.

The value of the `direction` property may be set to 1, meaning “up” or “above”, or to -1, meaning “down” or “below”. The symbols `UP` and `DOWN` may be used instead of 1 and -1 respectively. The default direction may be specified by setting `direction` to 0 or `CENTER`. Alternatively, in many cases predefined commands exist to specify the direction. These are all of the form

```
\xxxUp, xxxDown, xxxNeutral
```

where `xxxNeutral` means “use the default direction”. See [Section “Within-staff objects” in Learning Manual](#).

In a few cases, arpeggio being the only common example, the value of the `direction` property specifies whether the object is to be placed to the right or left of the parent object. In this case -1 or `LEFT` means “to the left” and 1 or `RIGHT` means “to the right”. 0 or `CENTER` means “use the default” direction, as before.

These indications affect all notes until they are cancelled.

```
c2( c)
\slurDown
c2( c)
c2( c)
\slurNeutral
c2( c)
```



5.4.3 Context layout order

Contexts are normally positioned in a system from top to bottom in the order in which they are encountered in the input file. When contexts are nested, the outer context will include inner nested contexts as specified in the input file, provided the inner contexts are included in the outer context’s “accepts” list. Nested contexts which are not included in the outer context’s “accepts” list will be repositioned below the outer context rather than nested within it.

Note that a context will be silently created implicitly if a command is encountered when there is no suitable context available to contain it. This can give rise to unexpected new staves or scores.

The default order in which contexts are laid out and the “accepts” list can be changed, see [Section 5.1.7 \[Aligning contexts\], page 408](#).

See also

Usage Manual: [Section “An extra staff appears” in Application Usage](#).

5.4.4 Distances and measurements

Distances in LilyPond are of two types: absolute and scaled.

Absolute distances are used for specifying margins, indents, and other page layout details, and are by default specified in millimeters. Distances may be specified in other units by following the quantity by `\mm`, `\cm`, `\in` (inches), or `\pt` (points, 1/72.27 of an inch). Page layout distances can also be specified in scalable units (see the following paragraph) by appending `\staff-space` to the quantity. Page layout is described in detail in [Section 4.1.2 \[Page formatting\], page 358](#).

Scaled distances are always specified in units of the staff-space or, rarely, the half staff-space. The staff-space is the distance between two adjacent staff lines. The default value can be changed globally by setting the global staff size, or it can be overridden locally by changing the `staff-space` property of `StaffSymbol`. Scaled distances automatically scale with any change to the either the global staff size or the `staff-space` property of `StaffSymbol`, but fonts scale automatically only with changes to the global staff size. The global staff size thus enables the overall size of a rendered score to be easily varied. For the methods of setting the global staff size see [Section 4.2.1 \[Setting the staff size\], page 365](#).

If just a section of a score needs to be rendered to a different scale, for example an ossia section or a footnote, the global staff size cannot simply be changed as this would affect the entire score. In such cases the change in size is made by overriding both the `staff-space` property of `StaffSymbol` and the size of the fonts. A Scheme function, `magstep`, is available to convert from a font size change to the equivalent change in `staff-space`. For an explanation and an example of its use, see [Section “Length and thickness of objects” in Learning Manual](#).

See also

Learning Manual: [Section “Length and thickness of objects” in Learning Manual](#).

Notation Reference: [Section 4.1.2 \[Page formatting\], page 358](#), [Section 4.2.1 \[Setting the staff size\], page 365](#).

5.4.5 Staff symbol properties

The vertical position of staff lines and the number of staff lines can be defined at the same time. As the following example shows, note positions are not influenced by the staff line positions.

Note: The `'line-positions` property overrides the `'line-count` property. The number of staff lines is implicitly defined by the number of elements in the list of values for `'line-positions`.

```
\new Staff \with {
  \override StaffSymbol #'line-positions = #'(7 3 0 -4 -6 -7)
}
{ a4 e' f b | d1 }
```



The width of a staff can be modified. The units are staff spaces. The spacing of objects inside the staff is not affected by this setting.

```
\new Staff \with {
  \override StaffSymbol #'width = #23
```

```
}
{ a4 e' f b | d1 }
```



5.4.6 Spanners

Many objects of musical notation extend over several notes or even several bars. Examples are slurs, beams, tuplet brackets, volta repeat brackets, crescendi, trills, and glissandi. Such objects are collectively called “spanners”, and have special properties to control their appearance and behaviour. Some of these properties are common to all spanners; others are restricted to a sub-set of the spanners.

All spanners support the `spanner-interface`. A few, essentially those that draw a straight line between the two objects, support in addition the `line-spanner-interface`.

Using the spanner-interface

This interface provides two properties that apply to several spanners.

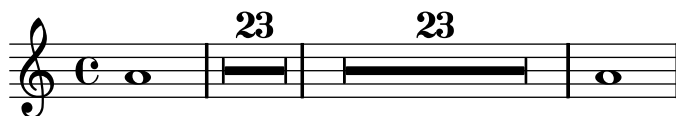
The minimum-length property

The minimum length of the spanner is specified by the `minimum-length` property. Increasing this usually has the necessary effect of increasing the spacing of the notes between the two end points. However, this override has no effect on many spanners, as their length is determined by other considerations. A few examples where it is effective are shown below.

```
a~a
a
% increase the length of the tie
-\tweak #'minimum-length #5
~a
```



```
a1
\compressFullBarRests
R1*23
% increase the length of the rest bar
\once \override MultiMeasureRest #'minimum-length = #20
R1*23
a1
```



```
a \< a a a \!
% increase the length of the hairpin
\override Hairpin #'minimum-length = #20
a \< a a a \!
```



This override can also be used to increase the length of slurs and phrasing slurs:

```
a( a)
a
-\tweak #'minimum-length #5
( a)
```

```
a\ ( a\ )
a
-\tweak #'minimum-length #5
\ ( a\ )
```



For some layout objects, the `minimum-length` property becomes effective only if the `set-spacing-rods` procedure is called explicitly. To do this, the `springs-and-rods` property should be set to `ly:spanner::set-spacing-rods`. For example, the minimum length of a glissando has no effect unless the `springs-and-rods` property is set:

```
% default
e \glissando c'

% not effective alone
\once \override Glissando #'minimum-length = #20
e, \glissando c'

% effective only when both overrides are present
\once \override Glissando #'minimum-length = #20
\once \override Glissando #'springs-and-rods = #ly:spanner::set-spacing-rods
e, \glissando c'
```



The same is true of the Beam object:

```
% not effective alone
\once \override Beam #'minimum-length = #20
e8 e e e

% effective only when both overrides are present
\once \override Beam #'minimum-length = #20
\once \override Beam #'springs-and-rods = #ly:spanner::set-spacing-rods
e8 e e e
```



The to-barline property

The second useful property of the `spanner-interface` is `to-barline`. By default this is true, causing hairpins and other spanners which are terminated on the first note of a measure to end instead on the immediately preceding bar line. If set to false, the spanner will extend beyond the bar line and end on the note itself:

```
a \< a a a a \! a a a \break
\override Hairpin #'to-barline = ##f
a \< a a a a \! a a a
```



This property is not effective for all spanners. For example, setting it to `#t` has no effect on slurs or phrasing slurs or on other spanners for which terminating on the bar line would not be meaningful.

Using the line-spanner-interface

Objects which support the `line-spanner-interface` include

- `DynamicTextSpanner`
- `Glissando`
- `TextSpanner`
- `TrillSpanner`
- `VoiceFollower`

The routine responsible for drawing the stencils for these spanners is `ly:line-interface::print`. This routine determines the exact location of the two end points and draws a line between them, in the style requested. The locations of the two end points of the spanner are computed on-the-fly, but it is possible to override their Y-coordinates. The properties which need to be specified are nested two levels down within the property hierarchy, but the syntax of the `\override` command is quite simple:

```
e2 \glissando b
\once \override Glissando #'(bound-details left Y) = #3
\once \override Glissando #'(bound-details right Y) = #-2
e2 \glissando b
```

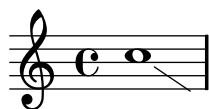


The units for the Y property are `staff-spaces`, with the center line of the staff being the zero point. For the glissando, this is the value for Y at the X-coordinate corresponding to the center point of each note head, if the line is imagined to be extended to there.

If Y is not set, the value is computed from the vertical position of the corresponding attachment point of the spanner.

In case of a line break, the values for the end points are specified by the `left-broken` and `right-broken` sub-lists of `bound-details`. For example:

```
\override Glissando #'breakable = ##t
\override Glissando #'(bound-details right-broken Y) = #-3
c1 \glissando \break
f1
```



A number of further properties of the `left` and `right` sub-lists of the `bound-details` property may be modified in the same way as `Y`:

Y This sets the Y-coordinate of the end point, in `staff-spaces` offset from the staff center line. By default, it is the center of the bound object, so a glissando points to the vertical center of the note head.

For horizontal spanners, such as text spanners and trill spanners, it is hardcoded to 0.

attach-dir

This determines where the line starts and ends in the 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 X-coordinate of the end point. It is usually computed on the fly, and overriding it has little useful effect.

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 that `text` be used instead.

text This is a markup that is evaluated to yield the stencil. It is used to put *cresc.*, *tr* and other text on horizontal spanners.

```
\override TextSpanner #'(bound-details left text)
  = \markup { \small \bold Slower }
c2\startTextSpan b c a\stopTextSpan
```



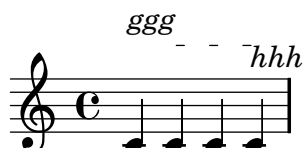
stencil-align-dir-y

stencil-offset

Without setting one of these, the stencil is simply put at the end-point, centered on the line, 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 vertically relative to the end point of the line:

```
\override TextSpanner
  #'(bound-details left stencil-align-dir-y) = #-2
\override TextSpanner
  #'(bound-details right stencil-align-dir-y) = #UP
```

```
\override TextSpanner
  #'(bound-details left text) = #"ggg"
\override TextSpanner
  #'(bound-details right text) = #"hhh"
c4^\startTextSpan c c c \stopTextSpan
```



Note that negative values move the text *up*, contrary to the effect that might be expected, as a value of -1 or DOWN means align the *bottom* edge of the text with the spanner line. A value of 1 or UP aligns the top edge of the text with the spanner line.

- arrow** Setting this sub-property to **#t** produces 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.

The music function **\endSpanners** terminates the spanner which starts on the immediately following note prematurely. It is terminated after exactly one note, or at the following bar line if **to-barline** is true and a bar line occurs before the next note.

```
\endSpanners
c2 \startTextSpan c2 c2
\endSpanners
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

Internals Reference: Section “TextSpanner” in *Internals Reference*, Section “Glissando” in *Internals Reference*, Section “VoiceFollower” in *Internals Reference*, Section “TrillSpanner” in *Internals Reference*, Section “line-spanner-interface” in *Internals Reference*.

5.4.7 Visibility of objects

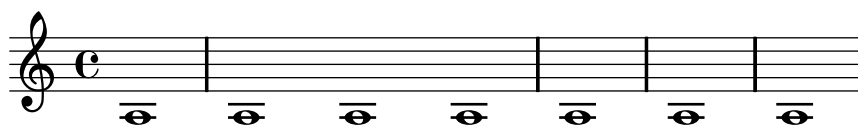
There are four main ways in which the visibility of layout objects can be controlled: their stencil can be removed, they can be made transparent, they can be colored white, or their **break-visibility** property can be overridden. The first three apply to all layout objects; the last to just a few – the *breakable* objects. The Learning Manual introduces these four techniques, see Section “Visibility and color of objects” in *Learning Manual*.

There are also a few other techniques which are specific to certain layout objects. These are covered under Special considerations.

Removing the stencil

Every layout object has a `stencil` property. By default this is set to the specific function which draws that object. If this property is overridden to `#f` no function will be called and the object will not be drawn. The default action can be recovered with `\revert`.

```
a1 a
\override Score.BarLine #'stencil = ##f
a a
\revert Score.BarLine #'stencil
a a a
```



Making objects transparent

Every layout object has a `transparent` property which by default is set to `#f`. If set to `#t` the object still occupies space but is made invisible.

```
a4 a
\once \override NoteHead #'transparent = ##t
a a
```



Painting objects white

Every layout object has a `color` property which by default is set to `black`. If this is overridden to `white` the object will be indistinguishable from the white background. However, if the object crosses other objects the color of the crossing points will be determined by the order in which they are drawn, and this may leave a ghostly image of the white object, as shown here:

```
\override Staff.Clef #'color = #white
a1
```



This may be avoided by changing the order of printing the objects. All layout objects have a `layer` property which should be set to an integer. Objects with the lowest value of `layer` are drawn first, then objects with progressively higher values are drawn, so objects with higher values overwrite objects with lower values. By default most objects are assigned a `layer` value of 1, although a few objects, including `StaffSymbol` and `BarLine`, are assigned a value of 0. The order of printing objects with the same value of `layer` is indeterminate.

In the example above the white clef, with a default `layer` value of 1, is drawn after the staff lines (default `layer` value 0), so overwriting them. To change this, the `Clef` object must be given in a lower value of `layer`, say -1, so that it is drawn earlier:

```
\override Staff.Clef #'color = #white
\override Staff.Clef #'layer = #-1
a1
```



Using break-visibility

Most layout objects are printed only once, but some like bar lines, clefs, time signatures and key signatures, may need to be printed twice when a line break occurs – once at the end of the line and again at the start of the next line. Such objects are called *breakable*, and have a property, the **break-visibility** property to control their visibility at the three positions in which they may appear – at the start of a line, within a line if they are changed, and at the end of a line if a change takes place there.

For example, the time signature by default will be printed at the start of the first line, but nowhere else unless it changes, when it will be printed at the point at which the change occurs. If this change occurs at the end of a line the new time signature will be printed at the start of the next line and a cautionary time signature will be printed at the end of the previous line as well.

This behaviour is controlled by the **break-visibility** property, which is explained in [Section “Visibility and color of objects” in *Learning Manual*](#). This property takes a vector of three booleans which, in order, determine whether the object is printed at the end of, within the body of, or at the beginning of a line. Or to be more precise, before a line break, where there is no line break, or after a line break.

Alternatively, these eight combinations may be specified by pre-defined functions, defined in ‘`scm/output-lib.scm`’, where the last three columns indicate whether the layout objects will be visible in the positions shown at the head of the columns:

Function	Vector	Before	At	After
form	form	no break	break	break
all-visible	'#(#t #t #t)	yes	yes	yes
begin-of-line-visible	'#(#f #f #t)	no	no	yes
center-visible	'#(#f #t #f)	no	yes	no
end-of-line-visible	'#(#t #f #f)	yes	no	no
begin-of-line-invisible	'#(#t #t #f)	yes	yes	no
center-invisible	'#(#t #f #t)	yes	no	yes
end-of-line-invisible	'#(#f #t #t)	no	yes	yes
all-invisible	'#(#f #f #f)	no	no	no

The default settings of **break-visibility** depend on the layout object. The following table shows all the layout objects of interest which are affected by **break-visibility** and the default setting of this property:

Layout object	Usual context	Default setting
BarLine	Score	calculated
BarNumber	Score	begin-of-line-visible
BreathingSign	Voice	begin-of-line-invisible
Clef	Staff	begin-of-line-visible
Custos	Staff	end-of-line-visible
DoublePercentRepeat	Voice	begin-of-line-invisible
KeySignature	Staff	begin-of-line-visible
OctavateEight	Staff	begin-of-line-visible
RehearsalMark	Score	end-of-line-invisible

TimeSignature

Staff

all-visible

The example below shows the use of the vector form to control the visibility of barlines:

```
f4 g a b
f4 g a b
% Remove bar line at the end of the current line
\once \override Score.BarLine #'break-visibility = #'#(#f #t #t)
\break
f4 g a b
f4 g a b
```



Although all three components of the vector used to override **break-visibility** must be present, not all of them are effective with every layout object, and some combinations may even give errors. The following limitations apply:

- Bar lines cannot be printed at start of line.
- A bar number cannot be printed at the start of the first line unless it is set to be different from 1.
- Clef – see below
- Double percent repeats are either all printed or all suppressed. Use `begin-of line-invisible` to print and `all-invisible` to suppress.
- Key signature – see below
- OctavateEight – see below

Special considerations

Visibility following explicit changes

The **break-visibility** property controls the visibility of key signatures and changes of clef only at the start of lines, i.e. after a break. It has no effect on the visibility of the key signature or clef following an explicit key change or an explicit clef change within or at the end of a line. In the following example the key signature following the explicit change to B-flat major is still visible, even though **all-invisible** is set.

```
\key g \major
f4 g a b
% Try to remove all key signatures
\override Staff.KeySignature #'break-visibility = #all-invisible
\key bes \major
f4 g a b
\break
f4 g a b
f4 g a b
```



The visibility of such explicit key signature and clef changes is controlled by the `explicitKeySignatureVisibility` and `explicitClefVisibility` properties. These are the equivalent of the `break-visibility` property and both take a vector of three booleans or the predefined functions listed above, exactly like `break-visibility`. Both are properties of the Staff context, not the layout objects themselves, and so they are set using the `\set` command. Both are set by default to `all-visible`. These properties control only the visibility of key signatures and clefs resulting from explicit changes and do not affect key signatures and clefs at the beginning of lines; `break-visibility` must still be overridden in the appropriate object to remove these.

```
\key g \major
f4 g a b
\set Staff.explicitKeySignatureVisibility = #all-invisible
\override Staff.KeySignature #'break-visibility = #all-invisible
\key bes \major
f4 g a b \break
f4 g a b
f4 g a b
```



Visibility of cautionary accidentals

To remove the cautionary accidentals printed at an explicit key change, set the Staff context property `printKeyCancellation` to `#f`:

```
\key g \major
f4 g a b
\set Staff.explicitKeySignatureVisibility = #all-invisible
\set Staff.printKeyCancellation = ##f
\override Staff.KeySignature #'break-visibility = #all-invisible
\key bes \major
f4 g a b \break
f4 g a b
f4 g a b
```





With these overrides only the accidentals before the notes remain to indicate the change of key.

Automatic bars

As a special case, the printing of bar lines can also be turned off by setting the `automaticBars` property in the Score context. If set to `#f`, bar lines will not be printed automatically; they must be explicitly created with a `\bar` command. Unlike the `\cadenzaOn` predefined command, measures are still counted. Bar generation will resume according to that count if this property is later set to `#t`. When set to `#f`, line breaks can occur only at explicit `\bar` commands.

Octavated clefs

The small octavation symbol on octavated clefs is produced by the `OctavateEight` layout object. Its visibility is controlled independently from that of the `Clef` object, so it is necessary to apply any required `break-visibility` overrides to both the `Clef` and the `OctavateEight` layout objects to fully suppress such clef symbols at the start of each line.

For explicit clef changes, the `explicitClefVisibility` property controls both the clef symbol and any octavation symbol associated with it.

See also

Learning Manual: [Section “Visibility and color of objects”](#) in *Learning Manual*

5.4.8 Line styles

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 between them, in the style requested.

Here is an example showing 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 locations of the end-points of the spanner are 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 value for `Y` is set to `-2` for the right end point. The left side may be similarly adjusted by specifying `left` instead of `right`.

If `Y` is not set, the value is computed from the vertical position of the left and right attachment points of the spanner.

Other adjustments of spanners are possible, for details, see [Section 5.4.6 \[Spanners\]](#), page 422.

5.4.9 Rotating objects

Both layout objects and elements of markup text can be rotated by any angle about any point, but the method of doing so differs.

Rotating layout objects

All layout objects which support the `grob-interface` can be rotated by setting their `rotation` property. This takes a list of three items: the angle of rotation counter-clockwise, and the `x` and `y` coordinates of the point relative to the object's reference point about which the rotation is to be performed. The angle of rotation is specified in degrees and the coordinates in staff-spaces.

The angle of rotation and the coordinates of the rotation point must be determined by trial and error.

There are only a few situations where the rotation of layout objects is useful; the following example shows one situation where they may be:

```
g4\< e' d' f\!
\override Hairpin #'rotation = #'(20 -1 0)
g,,4\< e' d' f\!
```



Rotating markup

All markup text can be rotated to lie at any angle by prefixing it with the `\rotate` command. The command takes two arguments: the angle of rotation in degrees counter-clockwise and the text to be rotated. The extents of the text are not rotated: they take their values from the extremes of the `x` and `y` coordinates of the rotated text. In the following example the `outside-staff-priority` property for text is set to `##f` to disable the automatic collision avoidance, which would push some of the text too high.

```
\override TextScript #'outside-staff-priority = ##f
g4^\markup { \rotate #30 "a G" }
b^\markup { \rotate #30 "a B" }
des^\markup { \rotate #30 "a D-Flat" }
fis^\markup { \rotate #30 "an F-Sharp" }
```



5.5 Advanced tweaks

This section discusses various approaches to fine tuning the appearance of the printed score.

See also

Learning Manual: [Section “Tweaking output”](#) in *Learning Manual*, [Section “Other sources of information”](#) in *Learning Manual*.

Notation Reference: [Section 5.2 \[Explaining the Internals Reference\]](#), page 409, [Section 5.3 \[Modifying properties\]](#), page 412.

Installed Files: ‘`scm/define-grobs.scm`’.

Snippets: [Section “Tweaks and overrides”](#) in *Snippets*.

Extending: [Section “Interfaces for programmers”](#) in *Extending*.

Internals Reference: [Section “All layout objects”](#) in *Internals Reference*.

5.5.1 Aligning objects

Graphical objects which support the `self-alignment-interface` and/or the `side-position-interface` can be aligned to a previously placed object in a variety of ways. For a list of these objects, see [Section “self-alignment-interface”](#) in *Internals Reference* and [Section “side-position-interface”](#) in *Internals Reference*.

All graphical objects have a reference point, a horizontal extent and a vertical extent. The horizontal extent is a pair of numbers giving the displacements from the reference point of the left and right edges, displacements to the left being negative. The vertical extent is a pair of numbers giving the displacement from the reference point to the bottom and top edges, displacements down being negative.

An object’s position on a staff is given by the values of the `X-offset` and `Y-offset` properties. The value of `X-offset` gives the displacement from the X coordinate of the reference point of the parent object, and the value of `Y-offset` gives the displacement from the center line of the staff. The values of `X-offset` and `Y-offset` may be set directly or may be set to be calculated by procedures in order to achieve alignment with the parent object.

Note: Many objects have special positioning considerations which cause any setting of `X-offset` or `Y-offset` to be ignored or modified, even though the object supports the `self-alignment-interface`. Overriding the `X-offset` or `Y-offset` properties to a fixed value causes the respective `self-alignment` property to be disregarded.

For example, an accidental can be repositioned vertically by setting `Y-offset` but any changes to `X-offset` have no effect.

Rehearsal marks may be aligned with breakable objects such as bar lines, clef symbols, time signature symbols and key signatures. There are special properties to be found in the `break-aligned-interface` for positioning rehearsal marks on such objects.

See also

[\[Using the break-alignable-interface\]](#), page 435, [Section “Callback functions”](#) in *Extending*.

Setting X-offset and Y-offset directly

Numerical values may be given to the `X-offset` and `Y-offset` properties of many objects. The following example shows three notes with the default fingering position and the positions with `X-offset` and `Y-offset` modified.

```
a-3
a
```

```

-\tweak #'X-offset #0
-\tweak #'Y-offset #0
-3
a
-\tweak #'X-offset #-1
-\tweak #'Y-offset #1
-3

```



Using the side-position-interface

An object which supports the `side-position-interface` can be placed next to its parent object so that the specified edges of the two objects touch. The object may be placed above, below, to the right or to the left of the parent. The parent cannot be specified; it is determined by the order of elements in the input stream. Most objects have the associated note head as their parent.

The values of the `side-axis` and `direction` properties determine where the object is to be placed, as follows:

side-axis property	direction property	Placement
0	-1	left
0	1	right
1	-1	below
1	1	above

When `side-axis` is 0, `X-offset` should be set to the procedure `ly:side-position-interface::x-aligned-side`. This procedure will return the correct value of `X-offset` to place the object to the left or right side of the parent according to value of `direction`.

When `side-axis` is 1, `Y-offset` should be set to the procedure `ly:side-position-interface::y-aligned-side`. This procedure will return the correct value of `Y-offset` to place the object to the top or bottom of the parent according to value of `direction`.

Using the self-alignment-interface

Self-aligning objects horizontally

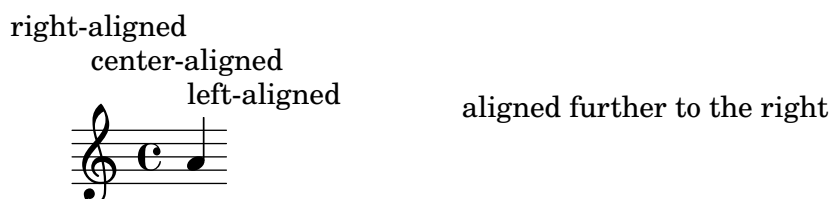
The horizontal alignment of an object which supports the `self-alignment-interface` is controlled by the value of the `self-alignment-X` property, provided the object's `X-offset` property is set to `ly:self-alignment-interface::x-aligned-on-self`. `self-alignment-X` may be given any real value, in units of half the total X extent of the object. Negative values move the object to the right, positive to the left. A value of 0 centers the object on the reference point of its parent, a value of -1 aligns the left edge of the object on the reference point of its parent, and a value of 1 aligns the right edge of the object on the reference point of its parent. The symbols `LEFT`, `CENTER` and `RIGHT` may be used instead of the values -1, 0, 1 respectively.

Normally the `\override` command would be used to modify the value of `self-alignment-X`, but the `\tweak` command can be used to separately align several annotations on a single note:

```

a'
-\tweak #'self-alignment-X #-1
^"left-aligned"
-\tweak #'self-alignment-X #0
^"center-aligned"
-\tweak #'self-alignment-X #RIGHT
^"right-aligned"
-\tweak #'self-alignment-X #-2.5
^"aligned further to the right"

```



Self-aligning objects vertically

Objects may be aligned vertically in an analogous way to aligning them horizontally if the `Y-offset` property is set to `ly:self-alignment-interface:y-aligned-on-self`. However, other mechanisms are often involved in vertical alignment: the value of `Y-offset` is just one variable taken into account. This may make adjusting the value of some objects tricky. The units are just half the vertical extent of the object, which is usually quite small, so quite large numbers may be required. A value of `-1` aligns the lower edge of the object with the reference point of the parent object, a value of `0` aligns the center of the object with the reference point of the parent, and a value of `1` aligns the top edge of the object with the reference point of the parent. The symbols `DOWN`, `CENTER`, `UP` may be substituted for `-1`, `0`, `1` respectively.

Self-aligning objects in both directions

By setting both `X-offset` and `Y-offset`, an object may be aligned in both directions simultaneously.

The following example shows how to adjust a fingering mark so that it nestles close to the note head.

```

a
-\tweak #'self-alignment-X #0.5 % move horizontally left
-\tweak #'Y-offset #ly:self-alignment-interface:y-aligned-on-self
-\tweak #'self-alignment-Y #-1 % move vertically up
-3 % third finger

```



Using the `break-alignable-interface`

Rehearsal marks and bar numbers may be aligned with notation objects other than bar lines. These objects include `ambitus`, `breathing-sign`, `clef`, `custos`, `staff-bar`, `left-edge`, `key-cancellation`, `key-signature`, and `time-signature`.

By default, rehearsal marks and bar numbers will be horizontally centered above the object:

```

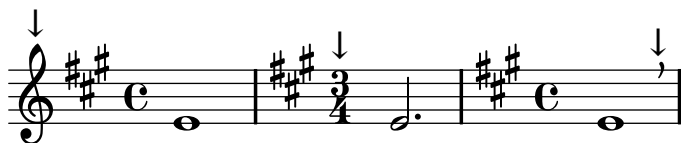
% The rehearsal mark will be centered above the Clef
\override Score.RehearsalMark #'break-align-symbols = #'(clef)
\key a \major

```

```

\clef treble
\mark ""
e1
% The rehearsal mark will be centered above the Time Signature
\override Score.RehearsalMark #'break-align-symbols = #'(time-signature)
\key a \major
\clef treble
\time 3/4
\mark ""
e2.
% The rehearsal mark will be centered above the Breath Mark
\override Score.RehearsalMark #'break-align-symbols = #'(breathing-sign)
\key a \major
\clef treble
\time 4/4
e1
\breathe
\mark ""

```

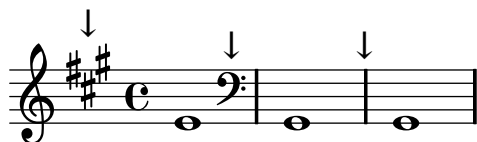


A list of possible target alignment objects may be specified. If some of the objects are invisible at that point due to the setting of `break-visibility` or the explicit visibility settings for keys and clefs, the rehearsal mark or bar number is aligned to the first object in the list which is visible. If no objects in the list are visible the object is aligned to the bar line. If the bar line is invisible the object is aligned to the place where the bar line would be.

```

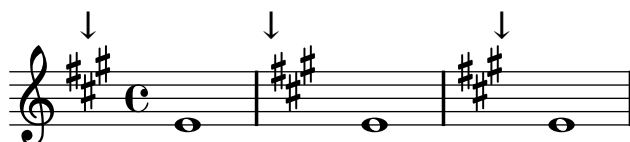
% The rehearsal mark will be centered above the Key Signature
\override Score.RehearsalMark #'break-align-symbols = #'(key-signature clef)
\key a \major
\clef treble
\mark ""
e1
% The rehearsal mark will be centered above the Clef
\set Staff.explicitKeySignatureVisibility = #all-invisible
\override Score.RehearsalMark #'break-align-symbols = #'(key-signature clef)
\key a \major
\clef bass
\mark ""
gis,,1
% The rehearsal mark will be centered above the Bar Line
\set Staff.explicitKeySignatureVisibility = #all-invisible
\set Staff.explicitClefVisibility = #all-invisible
\override Score.RehearsalMark #'break-align-symbols = #'(key-signature clef)
\key a \major
\clef treble
\mark ""
e''1

```



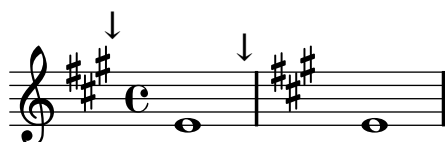
The alignment of the rehearsal mark relative to the notation object can be changed, as shown in the following example. In a score with multiple staves, this setting should be done for all the staves.

```
% The RehearsalMark will be centered above the Key Signature
\override Score.RehearsalMark #'break-align-symbols = #'(key-signature)
\key a \major
\clef treble
\time 4/4
\mark ""
e1
% The RehearsalMark will be aligned with the left edge of the Key Signature
\once \override Score.KeySignature #'break-align-anchor-alignment = #LEFT
\mark ""
\key a \major
e1
% The RehearsalMark will be aligned with the right edge of the Key Signature
\once \override Score.KeySignature #'break-align-anchor-alignment = #RIGHT
\key a \major
\mark ""
e1
```



The rehearsal mark can also be offset to the right or left of the left edge by an arbitrary amount. The units are staff-spaces:

```
% The RehearsalMark will be aligned with the left edge of the Key Signature
% and then shifted right by 3.5 staff-spaces
\override Score.RehearsalMark #'break-align-symbols = #'(key-signature)
\once \override Score.KeySignature #'break-align-anchor = #3.5
\key a \major
\mark ""
e1
% The RehearsalMark will be aligned with the left edge of the Key Signature
% and then shifted left by 2 staff-spaces
\once \override Score.KeySignature #'break-align-anchor = #-2
\key a \major
\mark ""
e1
```



5.5.2 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`.

5.5.3 Modifying stencils

All layout objects have a `stencil` property which is part of the `grob-interface`. By default, this property is usually set to a function specific to the object that is tailor-made to render the symbol which represents it in the output. For example, the standard setting for the `stencil` property of the `MultiMeasureRest` object is `ly:multi-measure-rest::print`.

The standard symbol for any object can be replaced by modifying the `stencil` property to reference a different, specially-written, procedure. This requires a high level of knowledge of the internal workings of LilyPond, but there is an easier way which can often produce adequate results.

This is to set the `stencil` property to the procedure which prints text – `ly:text-interface::print` – and to add a `text` property to the object which is set to contain the markup text which produces the required symbol. Due to the flexibility of markup, much can be achieved – see in particular [\[Graphic notation inside markup\]](#), page 189.

The following example demonstrates this by changing the note head symbol to a cross within a circle.

```
Xin0 = {
  \once \override NoteHead #'stencil = #ly:text-interface::print
  \once \override NoteHead #'text = \markup {
    \combine
      \halign #-0.7 \draw-circle #0.85 #0.2 ##f
      \musicglyph #"noteheads.s2cross"
  }
}
\relative c'' {
  a a \Xin0 a a
}
```



Any of the glyphs in the feta Font can be supplied to the `\musicglyph` markup command – see [Section A.6 \[The Feta font\]](#), page 457.

See also

Notation Reference: [\[Graphic notation inside markup\]](#), page 189, [Section 1.8.2 \[Formatting text\]](#), page 182, [Section A.8 \[Text markup commands\]](#), page 474, [Section A.6 \[The Feta font\]](#), page 457.

5.5.4 Modifying shapes

Modifying ties and slurs

Ties, slurs and phrasing slurs are drawn as third-order Bézier curves. If the shape of the tie or slur which is calculated automatically is not optimum, the shape may be modified manually by explicitly specifying the four control points required to define a third-order Bézier curve.

Third-order or cubic Bézier curves are defined by four control points. The first and fourth control points are precisely the starting and ending points of the curve. The intermediate two control points define the shape. Animations showing how the curve is drawn can be found on the web, but the following description may be helpful. The curve starts from the first control point heading directly towards the second, gradually bending over to head towards the third and continuing to bend over to head towards the fourth, arriving there travelling directly from the third control point. The curve is entirely contained in the quadrilateral defined by the four control points.

Here is an example of a case where the tie is not optimum, and where `\tieDown` would not help.

```
<<
{ e1 ~ e }
\\
{ r4 <g c,> <g c,> <g c,> }
>>
```



One way of improving this tie is to manually modify its control points, as follows.

The coordinates of the Bézier control points are specified in units of staff-spaces. The X coordinate is relative to the reference point of the note to which the tie or slur is attached, and the Y coordinate is relative to the staff center line. The coordinates are entered as a list of four pairs of decimal numbers (reals). One approach is to estimate the coordinates of the two end points, and then guess the two intermediate points. The optimum values are then found by trial and error.

It is useful to remember that a symmetric curve requires symmetric control points, and that Bézier curves have the useful property that transformations of the curve such as translation, rotation and scaling can be achieved by applying the same transformation to the curve's control points.

For the example above the following override gives a satisfactory tie. Note the placement – it has to be immediately before the note to which the start of the tie (or slur) is attached.

```
<<
{
  \once \override Tie
    #'control-points = #'((1 . -1) (3 . 0.6) (12.5 . 0.6) (14.5 . -1))
    e1 ~ e1
}
\\
{ r4 <g c,> <g c,> <g c,>4 }
>>
```



Known issues and warnings

It is not possible to modify shapes of ties or slurs by changing the `control-points` property if there are more than one at the same musical moment, not even by using the `\tweak` command. However, the `tie-configuration` property of `TieColumn` can be overridden to set start line and direction of ties as required.

5.6 Using music functions

Where tweaks need to be reused with different music expressions, it is often convenient to make the tweak part of a *music function*. In this section, we discuss only *substitution* functions, where the object is to substitute a variable into a piece of LilyPond input code. Other more complex functions are described in [Section “Music functions” in *Extending*](#).

5.6.1 Substitution function syntax

Making a function that substitutes a variable into LilyPond code is easy. The general form of these functions is

```
function =
#(define-music-function
  (parser location arg1 arg2 ...)
  (type1? type2? ...)
  #{
    ...music...
  #})
```

where

argN *nth* argument

typeN? a scheme *type predicate* for which *argN* must return `#t`.

...music... normal LilyPond input, using `$` to reference arguments (eg. `'$arg1'`).

The `parser` and `location` arguments are mandatory, and are used in some advanced situations as described in the ‘Extending’ manual (see [Section “Music functions” in *Extending*](#)). For substitution functions, just be sure to include them.

The list of type predicates is also required. Some of the most common type predicates used in music functions are:

```
boolean?
cheap-list?  (use instead of ‘list?’
  for faster processing)
ly:music?
markup?
number?
pair?
string?
symbol?
```

For a list of available type predicates, see [Section A.16 \[Predefined type predicates\]](#), page 548. User-defined type predicates are also allowed.

See also

Notation Reference: [Section A.16 \[Predefined type predicates\]](#), page 548.

Extending: [Section “Music functions” in *Extending*](#).

Installed Files: ‘lily/music-scheme.cc’, ‘scm/c++.scm’, ‘scm/lily.scm’.

5.6.2 Substitution function examples

This section introduces some substitution function examples. These are not intended to be exhaustive, but rather to demonstrate some of the possibilities of simple substitution functions.

In the first example, a function is defined that simplifies setting the padding of a TextScript:

```
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
}
```



In addition to numbers, we can use music expressions such as notes for arguments to music functions:

```
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
  #})

\relative c' { c4 d e f \custosNote g }
```



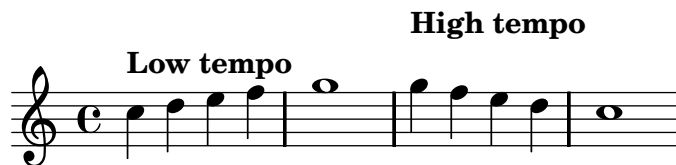
Substitution functions with multiple arguments can be defined:

```

tempoPadded =
#(define-music-function
  (parser location padding tempotext)
  (number? string?)
  #{
    \once \override Score.MetronomeMark #'padding = $padding
    \tempo \markup { \bold $tempotext }
  #})

\relative c'' {
  \tempo \markup { "Low tempo" }
  c4 d e f g1
  \tempoPadded #4.0 #"High tempo"
  g4 f e d c1
}

```



See also

TODO: add missing @ref's here.

Appendix A Notation manual tables

A.1 Chord name chart

The following charts shows two standard systems for printing chord names, along with the pitches they represent.

Ignatzek (default)	C	Cm	C+	C ^o
Alternative	C	C ^{b3}	C ^{#5}	C ^{b3 b5}
Def	C ⁷	Cm ⁷	C ^Δ	C ^{o7}
Alt ⁵	C ⁷	C ^{7 b3}	C ^{#7}	C ^{b3 b5 b7}
Def	C ^{7/#5}	Cm ^Δ	C ^{Δ/#5}	C [∅]
Alt ¹⁰	C ^{7 #5}	C ^{b3 #7}	C ^{#5 #7}	C ^{7 b3 b5}
Def	C ⁶	Cm ⁶	C ⁹	Cm ⁹
Alt ¹⁴	C ⁶	C ^{b3 6}	C ⁹	C ^{9 b3}
Def	Cm ¹³	Cm ¹¹	Cm ^{7/b5/9}	C ^{7/b9}
Alt ¹⁸	C ^{13 b3}	C ^{11 b3}	C ^{9 b3 b5}	C ^{7 b9}
Def	C ^{7/#9}	C ¹¹	C ^{7/#11}	C ¹³
Alt ²²	C ^{7 #9}	C ¹¹	C ^{9 #11}	C ¹³
Def	C ^{7/#11/b13}	C ^{7/#5/#9}	C ^{7/#9/#11}	C ^{7/b13}
Alt ²⁶	C ^{9 #11 b13}	C ^{7 #5 #9}	C ^{7 #9 #11}	C ^{11 b13}

Def

Alt₃₀

C^{7/b9/b13}
C^{11 b9 b13}

C^{7/#11}
C^{9 #11}

C^{△/9}
C^{9 #7}

C^{7/b13}
C^{11 b13}

Def

Alt₃₄

C^{7/b9/b13}
C^{11 b9 b13}

C^{7/b9/13}
C^{13 b9}

C^{△/9}
C^{9 #7}

C^{△/13}
C^{13 #7}

Def

Alt₃₈

C^{△/#11}
C^{9 #7 #11}

C^{7/b9/13}
C^{13 b9}

C^{sus4}
C^{add4 5}

C^{7/sus4}
C^{add4 5 7}

Def

Alt₄₂

C^{9/sus4}
C^{add4 5 7 9}


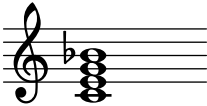
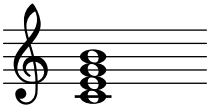


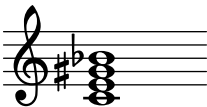


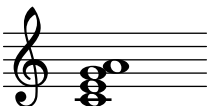
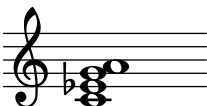
C^{add9}
C^{add9}

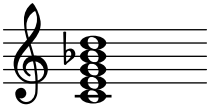
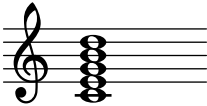
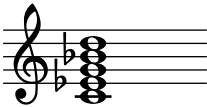
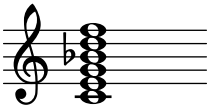
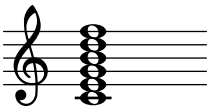
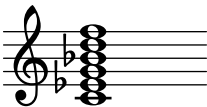
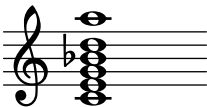
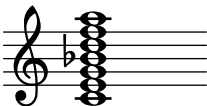
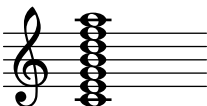
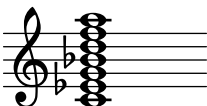
C^{m add11}
C^{b3 add11}

A.2 Common chord modifiers

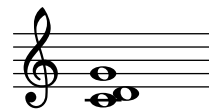
The following table shows chord modifiers that can be used in `\chordmode` to generate standard chord structures.

Chord type	Intervals	Modifier(s)	Example
Major	Major third, perfect fifth	5 or nothing	
Minor	Minor third, perfect fifth	m or m5	
Augmented	Major third, augmented fifth	aug	

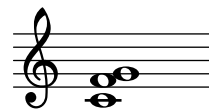
Diminished	Minor third, diminished fifth	dim	
Dominant seventh	Major triad, minor seventh	7	
Major seventh	Major triad, major seventh	maj7 or maj	
Minor seventh	Minor triad, minor seventh	m7	
Diminished seventh	Diminished triad, diminished seventh	dim7	
Augmented seventh	Augmented triad, minor seventh	aug7	
Half-diminished seventh	Diminished triad, minor seventh	m7.5-	
Minor-major seventh	Minor triad, major seventh	maj7.3-	
Major sixth	Major triad, sixth	6	
Minor sixth	Minor triad, sixth	m6	

Dominant ninth	Dominant seventh, major ninth	9	
Major ninth	Major seventh, major ninth	maj9	
Minor ninth	Minor seventh, major ninth	m9	
Dominant eleventh	Dominant ninth, perfect eleventh	11	
Major eleventh	Major ninth, perfect eleventh	maj11	
Minor eleventh	Minor ninth, perfect eleventh	m11	
Dominant thirteenth	Dominant ninth, major thirteenth	13	
Dominant thirteenth	Dominant eleventh, major thirteenth	13.11	
Major thirteenth	Major eleventh, major thirteenth	maj13.11	
Minor thirteenth	Minor eleventh, major thirteenth	m13.11	

Suspended second Major second, perfect fifth sus2



Suspended fourth Perfect fourth, perfect fifth sus4



A.3 Predefined fretboard diagrams

The chart below shows the predefined fretboard diagrams for guitar.

The chart displays predefined fretboard diagrams for guitar chords, organized into four rows of eight chords each. Each chord is represented by a fretboard diagram showing fingerings and a string mute (x) or open string (o) indicator, followed by its musical notation on a treble clef staff.

Row 1 (C major and related):

- C: 3 2 1
- Cm: 1 3 4 2 1
- C+: 2 1 1 4
- C°: 1 3 2 4
- C⁷: 3 2 4 1
- C[△]: 3 2
- Cm⁷: 1 3 1 2 1
- C⁹: 2 1 3 3 3

Row 2 (C# major and related):

- C#: 3 1 2 1
- C#m: 2 1 3
- C#+: 4 3 1 2
- C#°: 1 3 2 4
- C#⁷: 2 3 1 4
- C#[△]: 4 3 1 1 1
- C#m⁷: 4 2 1
- C#⁹: 2 1 3 3 3

Row 3 (D♭ major and related):

- D♭: 3 1 2 1
- D♭m: 2 1 3
- D♭+: 4 3 1 2
- D♭°: 1 3 2 4
- D♭⁷: 2 3 1 4
- D♭[△]: 4 3 1 1 1
- D♭m⁷: 4 2 1
- D♭⁹: 2 1 3 3 3

Row 4 (D major and related):

- D: 1 3 2
- Dm: 2 3 1
- D+: 2 3 1
- D°: 1 2
- D⁷: 2 1 3
- D[△]: 1 2 3
- Dm⁷: 2 1 1
- D⁹: 2 1 3 3 3

33

D[♯]	D[♯]m	D[♯]+	D[♯]⁰	D[♯]⁷	D[♯]^Δ	D[♯]m⁷	D[♯]⁹

41

E^b	E^bm	E^b+	E^b⁰	E^b⁷	E^b^Δ	E^bm⁷	E^b⁹

49

E	Em	E+	E⁰	E⁷	E^Δ	Em⁷	E⁹

57

F	Fm	F+	F⁰	F⁷	F^Δ	Fm⁷	F⁹

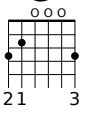
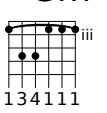
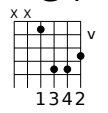
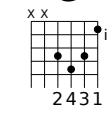
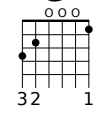
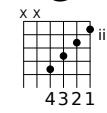
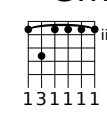
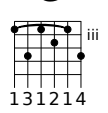
65

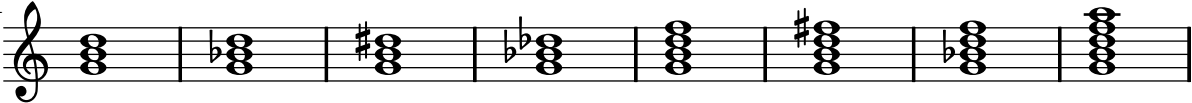
F[♯]	F[♯]m	F[♯]+	F[♯]⁰	F[♯]⁷	F[♯]^Δ	F[♯]m⁷	F[♯]⁹

73

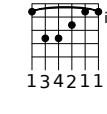
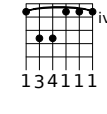
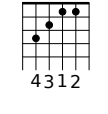
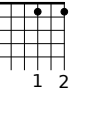
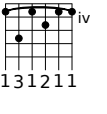
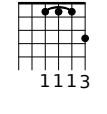
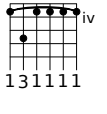
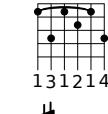
G^b	G^bm	G^b+	G^b⁰	G^b⁷	G^b^Δ	G^bm⁷	G^b⁹


81

G  21 3	Gm  134111	G+  1342	G^o  2431	G⁷  32 1	G^Δ  4321	Gm⁷  131111	G⁹  131214
---	--	--	---	---	---	--	---

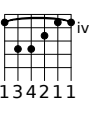
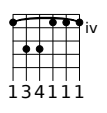
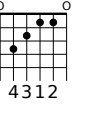
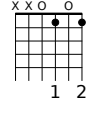
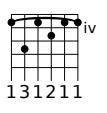
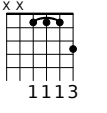
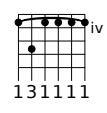
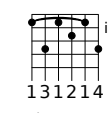


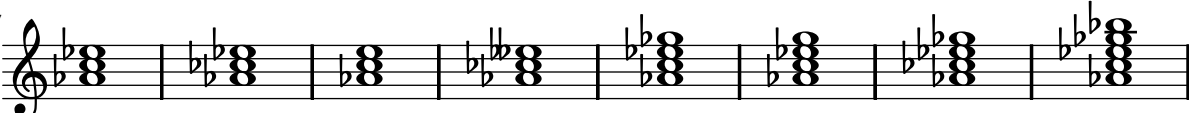
89

G[#]  134211	G[#]m  134111	G[#]+  4312	G^{#o}  1 2	G^{#7}  131211	G^{#Δ}  1113	G[#]m⁷  131111	G^{#9}  131214
---	--	--	---	--	--	--	--

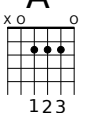
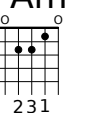
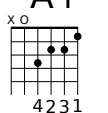
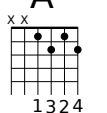
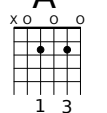
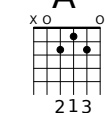
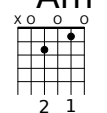
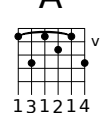



97

A^b  134211	A^bm  134111	A^b+  4312	A^{bo}  1 2	A^{b7}  131211	A^{bΔ}  1113	A^bm⁷  131111	A^{b9}  131214
---	--	--	---	--	--	--	--

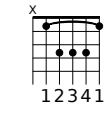
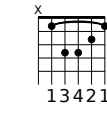
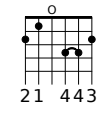
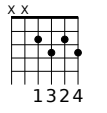
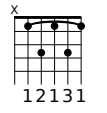
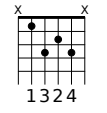
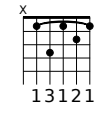
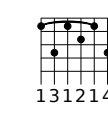



105

A  123	Am  231	A+  4231	A^o  1324	A⁷  1 3	A^Δ  213	Am⁷  2 1	A⁹  131214
--	---	--	---	--	--	---	---

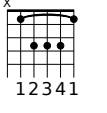
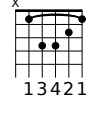
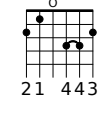
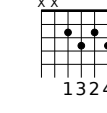
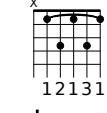
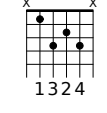
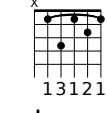
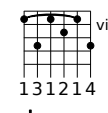


113

A[#]  12341	A[#]m  13421	A[#]+  21 443	A^{#o}  1324	A^{#7}  12131	A^{#Δ}  1324	A[#]m⁷  13121	A^{#9}  131214
--	---	--	--	---	--	---	--



121

B^b  12341	B^bm  13421	B^b+  21 443	B^{bo}  1324	B^{b7}  12131	B^{bΔ}  1324	B^bm⁷  13121	B^{b9}  131214
--	---	--	--	---	--	---	--

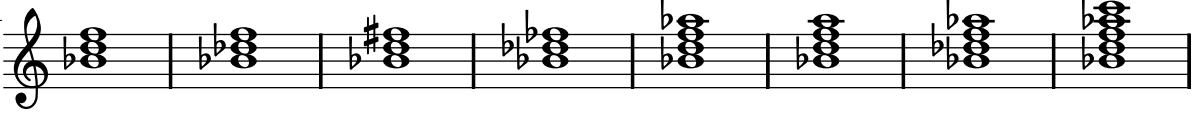


Diagram showing fretboard positions for chords B, Bm, B+, B⁰, B⁷, B^Δ, Bm⁷, and B⁹. Each diagram includes a fretboard grid with fingerings and a corresponding musical staff notation.

129

The musical staff for B chords is in treble clef with a key signature of two sharps (F# and C#). It contains eight measures, each representing a different B chord: B, Bm, B+, B⁰, B⁷, B^Δ, Bm⁷, and B⁹. Each measure contains a whole chord symbol.

The chart below shows the predefined fretboard diagrams for ukulele.

Diagram showing fretboard positions for chords C, Cm, C+, C⁰, C⁷, C^Δ, Cm⁷, C⁶, C^{sus2}, C^{sus4}, C⁹, C#, C#m, C#+, C#⁰, C#⁷, C#^Δ, C#m⁷, C#⁶, C#^{sus2}, C#^{sus4}, C#⁹, D^b, D^bm, D^b+, D^{b0}, D^{b7}, D^{bΔ}, D^bm⁷, D^{b6}, D^bsus2, D^bsus4, and D^{b9}. Each diagram includes a fretboard grid with fingerings and a corresponding musical staff notation.

12

18

23

29

The musical staff for C chords is in treble clef with a key signature of one sharp (F#). It contains 29 measures, each representing a different C chord: C, Cm, C+, C⁰, C⁷, C^Δ, Cm⁷, C⁶, C^{sus2}, C^{sus4}, C⁹, C#, C#m, C#+, C#⁰, C#⁷, C#^Δ, C#m⁷, C#⁶, C#^{sus2}, C#^{sus4}, C#⁹, D^b, D^bm, D^b+, D^{b0}, D^{b7}, D^{bΔ}, D^bm⁷, D^{b6}, D^bsus2, D^bsus4, and D^{b9}. Each measure contains a whole chord symbol.

34

D	Dm	D+	D ^o	D ⁷	D ^Δ	Dm ⁷	D ⁶	D ^{sus2}	D ^{sus4}	D ⁹
123	221	2114	1324	1112	1113	2213	1111	12	12	1312

45

D [#]	D [#] m	D [#] +	D ^{#o}	D ^{#7}	D ^{#Δ}
221	3321	221	1314	1112	1212

51

D [#] m ⁷	D ^{#6}	D ^{#sus2}	D ^{#sus4}	D ^{#9}
2214	1111	2211	2341	111

56

E ^b	E ^b m	E ^b +	E ^{b°}	E ^{b7}	E ^{bΔ}
221	3321	221	1314	1112	1212

62

E ^b m ⁷	E ^{b6}	E ^b sus2	E ^b sus4	E ^{b9}
2214	1111	2211	2341	111

67

E	Em	E+	E ^o	E ⁷	E ^Δ	Em ⁷	E ⁶	E ^{sus2}	E ^{sus4}	E ⁹
2341	3321	1 4	1 2	12 3	13 2	1 2	1111	3311	24 1	1222

78

F	Fm	F+	F^o	F⁷	F^Δ	Fm⁷	F⁶	F^{sus2}	F^{sus4}	F⁹

89

F[#]	F[#]m	F[#]+	F^{#o}	F^{#7}	F^{#Δ}

95

F[#]m⁷	F^{#6}	F^{#sus2}	F^{#sus4}	F^{#9}

100

G^b	G^bm	G^b+	G^{b0}	G^{b7}	G^{bΔ}

106

G^bm⁷	G^{b6}	G^bsus2	G^bsus4	G^{b9}

111

G	Gm	G+	G^o	G⁷	G^Δ	Gm⁷	G⁶	G^{sus2}	G^{sus4}	G⁹

122

G^\sharp	$G^\sharp m$	$G^\sharp +$	G^\sharp°	G^\sharp^7	G^\sharp^Δ

128

$G^\sharp m^7$	G^\sharp^6	G^\sharp^{sus2}	G^\sharp^{sus4}	G^\sharp^9

133

A^\flat	$A^\flat m$	$A^\flat +$	A^\flat°	A^\flat^7	A^\flat^Δ

139

$A^\flat m^7$	A^\flat^6	A^\flat^{sus2}	A^\flat^{sus4}	A^\flat^9

144

A	$A m$	$A +$	A°	A^7	A^Δ

150

$A m^7$	A^6	A^{sus2}	A^{sus4}	A^9

155

A^\sharp	$A^\sharp m$	$A^\sharp +$	A^\sharp°	A^\sharp^7	A^\sharp^Δ

161

$A^\sharp m^7$	A^\sharp^6	A^\sharp^{sus2}	A^\sharp^{sus4}	A^\sharp^9

166

B^\flat	$B^\flat m$	$B^\flat +$	B^\flat°	B^\flat^7	B^\flat^Δ

172

$B^\flat m^7$	B^\flat^6	B^\flat^{sus2}	B^\flat^{sus4}	B^\flat^9

177

B	Bm	$B +$	B°	B^7	B^Δ

183

Bm^7	B^6	B^{sus2}	B^{sus4}	B^9

A.4 MIDI instruments

The following is a list of names that can be used for the `midiInstrument` property. The order of the instruments below, starting in the left-hand column moving down, corresponds to the General MIDI Standard's 128 Program Numbers.

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 harp	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)	

A.5 List of colors

Normal colors

Usage syntax is detailed in [\[Coloring objects\]](#), page 168.

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 capitalization (e.g. ‘LightSlateBlue’) can also be spelled as space separated words without capitalization (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.

A.6 The Feta font

The following symbols are available in the Emmentaler font and may be accessed directly using text markup with the name of the glyph as shown in the tables below, such as `g^\markup {\musicglyph #"scripts.segno" }` or `\markup {\musicglyph #"five"}`. For more information, see [Section 1.8.2 \[Formatting text\]](#), page 182.

Clef glyphs

<code>clefs.C</code>		<code>clefs.C_change</code>	
<code>clefs.F</code>		<code>clefs.F_change</code>	
<code>clefs.G</code>		<code>clefs.G_change</code>	
<code>clefs.percussion</code>		<code>clefs.percussion_change</code>	
<code>clefs.tab</code>		<code>clefs.tab_change</code>	








Time Signature glyphs











<code>timesig.C44</code>		<code>timesig.C22</code>	
--------------------------	---	--------------------------	---

Number glyphs







plus	+	comma	,
hyphen	-	period	.
zero	0	one	1
two	2	three	3
four	4	five	5
six	6	seven	7
eight	8	nine	9

Accidental glyphs











accidentals.sharp	#	accidentals .sharp.arrowup	
accidentals .sharp.arrowdown		accidentals .sharp.arrowboth	
accidentals.sharp .slashslash.stem	‡	accidentals.sharp .slashslashslash.stemstem	‡
accidentals.sharp .slashslashslash.stem	‡	accidentals.sharp .slashslash.stemstemstem	‡
accidentals.natural	♮	accidentals .natural.arrowup	
accidentals .natural.arrowdown		accidentals .natural.arrowboth	
accidentals.flat	♭	accidentals.flat.arrowup	









accidentals .flat.arrowdown		accidentals .flat.arrowboth	
accidentals.flat.slash		accidentals.flat .slashslash	
accidentals .mirroredflat.flat		accidentals.mirroredflat	
accidentals .mirroredflat.backslash		accidentals.flatflat	
accidentals .flatflat.slash		accidentals.doublsharp	
accidentals.rightparen)	accidentals.leftparen	(

Default Notehead glyphs























noteheads.uM2		noteheads.dM2	
noteheads.sM1		noteheads.s0	
noteheads.s1		noteheads.s2	

Special Notehead glyphs

noteheads.sM1double		noteheads.s0diamond	
noteheads.s1diamond		noteheads.s2diamond	
noteheads.s0triangle		noteheads.d1triangle	
noteheads.ultriangle		noteheads.u2triangle	
noteheads.d2triangle		noteheads.s0slash	

<code>noteheads.s1slash</code>		<code>noteheads.s2slash</code>	
<code>noteheads.s0cross</code>		<code>noteheads.s1cross</code>	
<code>noteheads.s2cross</code>		<code>noteheads.s2xcircle</code>	
<code>noteheads.s0harmonic</code>		<code>noteheads.s2harmonic</code>	















Shape-note Notehead glyphs

<code>noteheads.s0do</code>		<code>noteheads.d1do</code>	
<code>noteheads.u1do</code>		<code>noteheads.d2do</code>	
<code>noteheads.u2do</code>		<code>noteheads.s0doThin</code>	
<code>noteheads.d1doThin</code>		<code>noteheads.u1doThin</code>	
<code>noteheads.d2doThin</code>		<code>noteheads.u2doThin</code>	
<code>noteheads.s0re</code>		<code>noteheads.u1re</code>	
<code>noteheads.d1re</code>		<code>noteheads.u2re</code>	
<code>noteheads.d2re</code>		<code>noteheads.s0reThin</code>	
<code>noteheads.u1reThin</code>		<code>noteheads.d1reThin</code>	
<code>noteheads.u2reThin</code>		<code>noteheads.d2reThin</code>	
<code>noteheads.s0mi</code>		<code>noteheads.s1mi</code>	







<code>noteheads.s2mi</code>	◆	<code>noteheads.s0miMirror</code>	◇
<code>noteheads.s1miMirror</code>	◇	<code>noteheads.s2miMirror</code>	◆
<code>noteheads.s0miThin</code>	◇	<code>noteheads.s1miThin</code>	◇
<code>noteheads.s2miThin</code>	◆	<code>noteheads.u0fa</code>	▷
<code>noteheads.d0fa</code>	▷	<code>noteheads.u1fa</code>	▷
<code>noteheads.d1fa</code>	▷	<code>noteheads.u2fa</code>	◀
<code>noteheads.d2fa</code>	◀	<code>noteheads.u0faThin</code>	▷
<code>noteheads.d0faThin</code>	▷	<code>noteheads.u1faThin</code>	▷
<code>noteheads.d1faThin</code>	▷	<code>noteheads.u2faThin</code>	◀
<code>noteheads.d2faThin</code>	◀	<code>noteheads.s0sol</code>	○
<code>noteheads.s1sol</code>	○	<code>noteheads.s2sol</code>	●
<code>noteheads.s0la</code>	□	<code>noteheads.s1la</code>	□
<code>noteheads.s2la</code>	■	<code>noteheads.s0laThin</code>	□
<code>noteheads.s1laThin</code>	□	<code>noteheads.s2laThin</code>	■
<code>noteheads.s0ti</code>	◇	<code>noteheads.ulti</code>	◇

noteheads.d1ti		noteheads.u2ti	
noteheads.d2ti		noteheads.s0tiThin	
noteheads.u1tiThin		noteheads.d1tiThin	
noteheads.u2tiThin		noteheads.d2tiThin	

Rest glyphs

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	

Flag glyphs

flags.u3		flags.u4	
flags.u5		flags.u6	
flags.u7		flags.d3	

flags.ugrace



flags.dgrace



flags.d4



flags.d5



flags.d6



flags.d7



Dot glyphs

dots.dot



Dynamic glyphs

space

f



m



p



r



s



z



Script glyphs

scripts.ufermata



scripts.dfermata



scripts.ushortfermata



scripts.dshortfermata



scripts.ulongfermata



scripts.dlongfermata



scripts.uverylongfermata



scripts.dverylongfermata































scripts.thumb











scripts.sforzato





scripts.espr	$\langle \rangle$	scripts.staccato	.
scripts.ustaccatissimo	†	scripts.dstaccatissimo	‡
scripts.tenuto	—	scripts.uportato	÷
scripts.dportato	÷	scripts.umarcato	^
scripts.dmarcato	v	scripts.open	o
scripts.halfopen	ø	scripts.stopped	+
scripts.upbow	V	scripts.downbow	⌊
scripts.reverseturn	∞	scripts.turn	∞
scripts.trill	<i>tr</i>	scripts.upedalheel	u
scripts.dpedalheel	n	scripts.upedaltoe	V
scripts.dpedaltoe	^	scripts.flageolet	o
scripts.segno	%	scripts.varsegno	<i>sc</i>
scripts.coda	⦶	scripts.varcoda	⦶
scripts.rcomma	,	scripts.lcomma	‘
scripts.rvarcomma	/	scripts.lvarcomma	/

<code>scripts.arpeggio</code>		<code>scripts.trill_element</code>	
<code>scripts.arpeggio</code> <code>.arrow.M1</code>		<code>scripts.arpeggio.arrow.1</code>	
<code>scripts.trilelement</code>		<code>scripts.prall</code>	
<code>scripts.mordent</code>		<code>scripts.prallprall</code>	
<code>scripts.prallmordent</code>		<code>scripts.upprall</code>	
<code>scripts.upmordent</code>		<code>scripts.pralldown</code>	
<code>scripts.downprall</code>		<code>scripts.downmordent</code>	
<code>scripts.prallup</code>		<code>scripts.lineprall</code>	
<code>scripts.caesura.curved</code>		<code>scripts.caesura.straight</code>	
<code>scripts.snappizzicato</code>		<code>scripts.ictus</code>	
<code>scripts.uaccentus</code>		<code>scripts.daccentus</code>	
<code>scripts.usemicirculus</code>		<code>scripts.dsemicirculus</code>	
<code>scripts.circulus</code>		<code>scripts.augmentum</code>	
<code>scripts</code> <code>.usignumcongruentiae</code>		<code>scripts</code> <code>.dsignumcongruentiae</code>	








Arrowhead glyphs

<code>arrowheads.open.01</code>		<code>arrowheads.open.0M1</code>	
<code>arrowheads.open.11</code>		<code>arrowheads.open.1M1</code>	
<code>arrowheads.close.01</code>		<code>arrowheads.close.0M1</code>	
<code>arrowheads.close.11</code>		<code>arrowheads.close.1M1</code>	







Bracket-tip glyphs

<code>brackettips.up</code>		<code>brackettips.down</code>	
-----------------------------	---	-------------------------------	--

Pedal glyphs

<code>pedal.*</code>		<code>pedal.M</code>	
<code>pedal..</code>		<code>pedal.P</code>	
<code>pedal.d</code>		<code>pedal.e</code>	
<code>pedal.Ped</code>			

Accordion glyphs

<code>accordion.accDiscant</code>		<code>accordion.accDot</code>	
<code>accordion.accFreebase</code>		<code>accordion.accStdbase</code>	
<code>accordion.accBayanbase</code>		<code>accordion.accOldEE</code>	

Vaticana glyphs





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	⌞
clefs.vaticana.do	♩	clefs.vaticana.do_change	♩
clefs.vaticana.fa	♯	clefs.vaticana.fa_change	♯
custodes.vaticana.u0	↓	custodes.vaticana.u1	↓
custodes.vaticana.u2	↓	custodes.vaticana.d0	⌞
custodes.vaticana.d1	⌞	custodes.vaticana.d2	⌞
accidentals.vaticanaM1	♭	accidentals.vaticana0	♭
dots.dotvaticana	·		

Medicaea glyphs

























noteheads.smedicaea .inclinatum	◆	noteheads .smedicaea.punctum	■
noteheads .smedicaea.rvirga	└	noteheads .smedicaea.virga	┐
clefs.medicaea.do	⦿	clefs.medicaea.do_change	⦿
clefs.medicaea.fa	⦿	clefs.medicaea.fa_change	⦿
custodes.medicaea.u0	┆	custodes.medicaea.u1	┆
custodes.medicaea.u2	┆	custodes.medicaea.d0	┆
custodes.medicaea.d1	┆	custodes.medicaea.d2	┆
accidentals.medicaeaM1	♭		

Hufnagel glyphs













noteheads .shufnagel.punctum	◆	noteheads .shufnagel.virga	↑
noteheads.shufnagel.lpes	▬	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	✓

<code>custodes.hufnagel.d0</code>		<code>custodes.hufnagel.d1</code>	
<code>custodes.hufnagel.d2</code>		<code>accidentals.hufnagelM1</code>	



















Mensural glyphs











<code>rests.M3mensural</code>		<code>rests.M2mensural</code>	
<code>rests.M1mensural</code>		<code>rests.0mensural</code>	
<code>rests.1mensural</code>		<code>rests.2mensural</code>	
<code>rests.3mensural</code>		<code>rests.4mensural</code>	
<code>noteheads.s1mensural</code>		<code>noteheads.sM3mensural</code>	
<code>noteheads.sM2mensural</code>		<code>noteheads.sM1mensural</code>	
<code>noteheads.s0mensural</code>		<code>noteheads.s1mensural</code>	
<code>noteheads.s2mensural</code>		<code>clefs.mensural.c</code>	
<code>clefs.mensural.c_change</code>		<code>clefs.mensural.f</code>	
<code>clefs.mensural.f_change</code>		<code>clefs.mensural.g</code>	
<code>clefs.mensural.g_change</code>		<code>custodes.mensural.u0</code>	
<code>custodes.mensural.u1</code>		<code>custodes.mensural.u2</code>	

custodes.mensural.d0	↘	custodes.mensural.d1	↘
custodes.mensural.d2	↘	accidentals.mensural1	✕
accidentals.mensuralM1	l	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	

Neomensural glyphs

rests.M3neomensural		rests.M2neomensural	
rests.M1neomensural		rests.0neomensural	
rests.1neomensural		rests.2neomensural	
rests.3neomensural		rests.4neomensural	
noteheads.s1neomensural		noteheads.sM3neomensural	
noteheads.sM2neomensural		noteheads.sM1neomensural	
noteheads.s0neomensural		noteheads.s1neomensural	
noteheads.s2neomensural		clefs.neomensural.c	
clefs.neomensural .c_change		timesig.neomensural44	

<code>timesig.neomensural22</code>		<code>timesig.neomensural32</code>	
<code>timesig.neomensural64</code>		<code>timesig.neomensural94</code>	
<code>timesig.neomensural34</code>		<code>timesig.neomensural68</code>	
<code>timesig.neomensural98</code>		<code>timesig.neomensural48</code>	
<code>timesig.neomensural68alt</code>		<code>timesig.neomensural24</code>	

Petrucchi glyphs

<code>noteheads.s0petrucci</code>		<code>noteheads.s1petrucci</code>	
<code>noteheads.s2petrucci</code>		<code>clefs.petrucchi.c1</code>	
<code>clefs.petrucchi.c1_change</code>		<code>clefs.petrucchi.c2</code>	
<code>clefs.petrucchi.c2_change</code>		<code>clefs.petrucchi.c3</code>	
<code>clefs.petrucchi.c3_change</code>		<code>clefs.petrucchi.c4</code>	
<code>clefs.petrucchi.c4_change</code>		<code>clefs.petrucchi.c5</code>	
<code>clefs.petrucchi.c5_change</code>		<code>clefs.petrucchi.f</code>	
<code>clefs.petrucchi.f_change</code>		<code>clefs.petrucchi.g</code>	
<code>clefs.petrucchi.g_change</code>			

Solesmes glyphs

<code>noteheads.ssolesmes</code> <code>.incl.parvum</code>	,	<code>noteheads</code> <code>.ssolesmes.auct.asc</code>	✓
<code>noteheads</code> <code>.ssolesmes.auct.desc</code>	⌞	<code>noteheads.ssolesmes</code> <code>.incl.auctum</code>	⌞
<code>noteheads</code> <code>.ssolesmes.stropha</code>	⌞	<code>noteheads.ssolesmes</code> <code>.stropha.aucta</code>	⌞
<code>noteheads</code> <code>.ssolesmes.oriscus</code>	⌞		

A.7 Note head styles

The following styles may be used for note heads.

default altdefault



9 baroque neomensural



17 mensural petrucci



25 harmonic harmonic-black



33 harmonic-mixed diamond



41 cross xcircle



49 triangle slash



A.8 Text markup commands

The following commands can all be used inside `\markup { }`.

A.8.1 Font

`\abs-fontsize size (number) arg (markup)`

Use *size* as the absolute font size to display *arg*. Adjusts `baseline-skip` and `word-space` accordingly.

```
\markup {
  default text font size
  \hspace #2
  \abs-fontsize #16 { text font size 16 }
  \hspace #2
  \abs-fontsize #12 { text font size 12 }
}
```

default text font size **text font size 16** text font size 12

`\bold arg (markup)`

Switch to bold font-series.

```
\markup {
  default
  \hspace #2
  \bold
  bold
}
```

default **bold**

`\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.

```
\markup {
  \override #'(box-padding . 0.5)
  \box
  \line { V. S. }
}
```

V. S.

Used properties:

- `box-padding` (0.2)
- `font-size` (0)
- `thickness` (1)

`\caps arg (markup)`

Copy of the `\smallCaps` command.

```
\markup {
  default
  \hspace #2
  \caps {
```

```

    Text in small caps
  }
}

```

```

default    TEXT IN SMALL CAPS

```

`\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.

```

\markup {
  \dynamic {
    sfzp
  }
}

```

sfzp

`\finger` *arg* (markup)

Set *arg* as small numbers.

```

\markup {
  \finger {
    1 2 3 4 5
  }
}

```

1 2 3 4 5

`\fontCaps` *arg* (markup)

Set `font-shape` to caps

Note: `\fontCaps` requires the installation and selection of fonts which support the caps font shape.

`\fontsize` *increment* (number) *arg* (markup)

Add *increment* to the font-size. Adjusts `baseline-skip` accordingly.

```

\markup {
  default
  \hspace #2
  \fontsize #-1.5
  smaller
}

```

```

default    smaller

```

Used properties:

- `baseline-skip` (2)
- `word-space` (1)
- `font-size` (0)

`\huge` *arg* (markup)

Set font size to +2.

```
\markup {
  default
  \hspace #2
  \huge
  huge
}
```

default huge

`\italic arg` (markup)
Use italic font-shape for *arg*.

```
\markup {
  default
  \hspace #2
  \italic
  italic
}
```

default *italic*

`\large arg` (markup)
Set font size to +1.

```
\markup {
  default
  \hspace #2
  \large
  large
}
```

default large

`\larger arg` (markup)
Increase the font size relative to the current setting.

```
\markup {
  default
  \hspace #2
  \larger
  larger
}
```

default larger

`\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.

```
\markup {
  default
  \hspace #2
```

```

\magnify #1.5 {
  50% larger
}

```

default **50% larger**

`\medium arg` (markup)

Switch to medium font-series (in contrast to bold).

```

\markup {
  \bold {
    some bold text
    \hspace #2
    \medium {
      medium font series
    }
    \hspace #2
    bold again
  }
}

```

some bold text medium font series **bold again**

`\normal-size-sub arg` (markup)

Set *arg* in subscript with a normal font size.

```

\markup {
  default
  \normal-size-sub {
    subscript in standard size
  }
}

```

default subscript in standard size

Used properties:

- `baseline-skip`

`\normal-size-super arg` (markup)

Set *arg* in superscript with a normal font size.

```

\markup {
  default
  \normal-size-super {
    superscript in standard size
  }
}

```

default superscript in standard size

Used properties:

- `baseline-skip`

`\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.

```
\markup {
  \huge \bold \sans \caps {
    Some text with font overrides
    \hspace #2
    \normal-text {
      Default text, same font-size
    }
    \hspace #2
    More text as before
  }
}
```

SOME TEXT WITH FONT OVERRIDES Default text, same font-size **MOR**

`\normalsize` *arg* (markup)

Set font size to default.

```
\markup {
  \teeny {
    this is very small
    \hspace #2
    \normalsize {
      normal size
    }
    \hspace #2
    teeny again
  }
}
```

this is very small **normal size** teeny again

`\number` *arg* (markup)

Set font family to `number`, which yields the font used for time signatures and fingerings. This font contains numbers and some punctuation; it has no letters.

```
\markup {
  \number {
    0 1 2 3 4 5 6 7 8 9 . ,
  }
}
```

0123456789.,

`\roman` *arg* (markup)

Set font family to `roman`.

```
\markup {
  \sans \bold {
    sans serif, bold
    \hspace #2
    \roman {
```

```

        text in roman font family
    }
    \hspace #2
    return to sans
}
}

```

sans serif, bold text in roman font family return to sans

`\sans` *arg* (markup)

Switch to the sans serif font family.

```

\markup {
  default
  \hspace #2
  \sans {
    sans serif
  }
}

```

default sans serif

`\simple` *str* (string)

A simple text string; `\markup { foo }` is equivalent with `\markup { \simple #"foo" }`.

Note: for creating standard text markup or defining new markup commands, the use of `\simple` is unnecessary.

```

\markup {
  \simple #"simple"
  \simple #"text"
  \simple #"strings"
}

```

simple text strings

`\small` *arg* (markup)

Set font size to -1.

```

\markup {
  default
  \hspace #2
  \small
  small
}

```

default small

`\smallCaps` *arg* (markup)

Emit *arg* as small caps.

Note: `\smallCaps` does not support accented characters.

```

\markup {
  default
  \hspace #2

```

```

\smallCaps {
  Text in small caps
}

```

default TEXT IN SMALL CAPS

`\smaller arg` (markup)
Decrease the font size relative to the current setting.

```

\markup {
  \fontsize #3.5 {
    some large text
    \hspace #2
    \smaller {
      a bit smaller
    }
    \hspace #2
    more large text
  }
}

```

some large text a bit smaller more large text

`\sub arg` (markup)
Set *arg* in subscript.

```

\markup {
  \concat {
    H
    \sub {
      2
    }
    0
  }
}

```

H₂O

Used properties:

- baseline-skip
- font-size (0)

`\super arg` (markup)
Set *arg* in superscript.

```

\markup {
  E =
  \concat {
    mc
    \super
    2
  }
}

```

$$E = mc^2$$

Used properties:

- `baseline-skip`
- `font-size (0)`

`\teeny arg` (markup)

Set font size to -3.

```
\markup {
  default
  \hspace #2
  \teeny
  teeny
}
```

`default` `teeny`

`\text arg` (markup)

Use a text font instead of music symbol or music alphabet font.

```
\markup {
  \number {
    1, 2,
    \text {
      three, four,
    }
    5
  }
}
```

1, 2, three, four, **5**

`\tiny arg` (markup)

Set font size to -2.

```
\markup {
  default
  \hspace #2
  \tiny
  tiny
}
```

`default` `tiny`

`\typewriter arg` (markup)

Use `font-family typewriter` for `arg`.

```
\markup {
  default
  \hspace #2
  \typewriter
  typewriter
}
```

`default` `typewriter`

`\underline arg` (markup)

Underline *arg*. Looks at **thickness** to determine line thickness and y-offset.

```
\markup {
  default
  \hspace #2
  \override #'(thickness . 2)
  \underline {
    underline
  }
}
```

default underline

Used properties:

- **thickness** (1)

`\upright arg` (markup)

Set font-shape to upright. This is the opposite of *italic*.

```
\markup {
  \italic {
    italic text
    \hspace #2
    \upright {
      upright text
    }
    \hspace #2
    italic again
  }
}
```

italic text upright text *italic again*

A.8.2 Align

`\center-align arg` (markup)

Align *arg* to its X center.

```
\markup {
  \column {
    one
    \center-align
    two
    three
  }
}
```

one
two
three

`\center-column args` (markup list)

Put *args* in a centered column.

```
\markup {
  \center-column {
    one
    two
    three
  }
}
```

```
one
two
three
```

Used properties:

- `baseline-skip`

`\column` *args* (markup list)

Stack the markups in *args* vertically. The property `baseline-skip` determines the space between markups in *args*.

```
\markup {
  \column {
    one
    two
    three
  }
}
```

```
one
two
three
```

Used properties:

- `baseline-skip`

`\combine` *arg1* (markup) *arg2* (markup)

Print two markups on top of each other.

Note: `\combine` cannot take a list of markups enclosed in curly braces as an argument; the follow example will not compile:

```
\combine { a list }
\markup {
  \fontsize #5
  \override #'(thickness . 2)
  \combine
    \draw-line #'(0 . 4)
    \arrow-head #Y #DOWN ##f
}
```



`\concat` *args* (markup list)

Concatenate *args* in a horizontal line, without spaces in between. Strings and simple markups are concatenated on the input level, allowing ligatures. For example, `\concat { "f" \simple #"i" }` is equivalent to `"fi"`.

```
\markup {
  \concat {
    one
    two
    three
  }
}
```

onetwothree

`\dir-column` *args* (markup list)

Make a column of *args*, going up or down, depending on the setting of the `direction` layout property.

```
\markup {
  \override #`(direction . ,UP) {
    \dir-column {
      going up
    }
  }
  \hspace #1
  \dir-column {
    going down
  }
  \hspace #1
  \override #'(direction . 1) {
    \dir-column {
      going up
    }
  }
}
```

```
up          up
going going going
          down
```

Used properties:

- `baseline-skip`
- `direction`

`\fill-line` *args* (markup list)

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.

```
\markup {
  \column {
    \fill-line {
      Words evenly spaced across the page
    }
  }
  \null
  \fill-line {
    \line { Text markups }
    \line {
      \italic { evenly spaced }
    }
  }
}
```

```

    }
    \line { across the page }
  }
}

```

Words	evenly	spaced	across	the	page
Text markups		<i>evenly spaced</i>			across the page

Used properties:

- `line-width` (#f)
- `word-space` (1)
- `text-direction` (1)

`\general-align` *axis* (integer) *dir* (number) *arg* (markup)
Align *arg* in *axis* direction to the *dir* side.

```

\markup {
  \column {
    one
    \general-align #X #LEFT
    two
    three
    \null
    one
    \general-align #X #CENTER
    two
    three
    \null
    \line {
      one
      \general-align #Y #UP
      two
      three
    }
    \null
    \line {
      one
      \general-align #Y #3.2
      two
      three
    }
  }
}

```

```

one
two
three

```

```

one
two
three

```

```

one   three
  two

```

```

one   three
  two

```

`\halign` *dir* (number) *arg* (markup)

Set horizontal alignment. If *dir* is `-1`, then it is left-aligned, while `+1` is right. Values in between interpolate alignment accordingly.

```

\markup {
  \column {
    one
    \halign #LEFT
    two
    three
    \null
    one
    \halign #CENTER
    two
    three
    \null
    one
    \halign #RIGHT
    two
    three
    \null
    one
    \halign #-5
    two
    three
  }
}

```

one
two
three

one
two
three

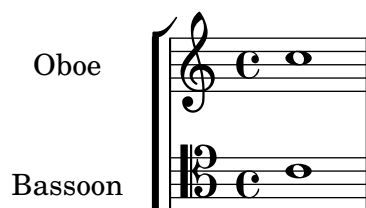
one
two
three

one
two
three

`\hcenter-in` *length* (number) *arg* (markup)

Center *arg* horizontally within a box of extending *length*/2 to the left and right.

```
\new StaffGroup <<
  \new Staff {
    \set Staff.instrumentName = \markup {
      \hcenter-in #12
      Oboe
    }
    c''1
  }
  \new Staff {
    \set Staff.instrumentName = \markup {
      \hcenter-in #12
      Bassoon
    }
    \clef tenor
    c'1
  }
>>
```



`\hspace` *amount* (number)

Create an invisible object taking up horizontal space *amount*.

```
\markup {
  one
  \hspace #2
  two
  \hspace #8
  three
}
```

one two three

`\justify-field` *symbol* (*symbol*)

Justify the data which has been assigned to *symbol*.

```
\header {
  title = "My title"
  myText = "Lorem ipsum dolor sit amet, consectetur adipisicing
    elit, sed do eiusmod tempor incididunt ut labore et dolore magna
    aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
    laboris nisi ut aliquip ex ea commodo consequat."
}

\paper {
  bookTitleMarkup = \markup {
    \column {
      \fill-line { \fromproperty #'header:title }
      \null
      \justify-field #'header:myText
    }
  }
}

\markup {
  \null
}
```

My title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

`\justify` *args* (*markup list*)

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.

```
\markup {
  \justify {
    Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
    Ut enim ad minim veniam, quis nostrud exercitation ullamco
    laboris nisi ut aliquip ex ea commodo consequat.
  }
}
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width` (#f)
- `baseline-skip`

`\justify-string` *arg* (string)

Justify a string. Paragraphs may be separated with double newlines

```
\markup {
  \override #'(line-width . 40)
  \justify-string #"Lorem ipsum dolor sit amet, consectetur
    adipisicing elit, sed do eiusmod tempor incididunt ut labore
    et dolore magna aliqua.
```

Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.

Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum"

```
}
```

Lorem ipsum dolor sit amet,
consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et
dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat.

Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia
deserunt mollit anim id est laborum

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width`
- `baseline-skip`

`\left-align` *arg* (markup)

Align *arg* on its left edge.

```
\markup {
  \column {
    one
    \left-align
    two
    three
  }
}
```

one
two
three

`\left-column` *args* (markup list)

Put *args* in a left-aligned column.

```
\markup {
  \left-column {
    one
    two
    three
  }
}
```

one
two
three

Used properties:

- `baseline-skip`

`\line` *args* (markup list)

Put *args* in a horizontal line. The property `word-space` determines the space between markups in *args*.

```
\markup {
  \line {
    one two three
  }
}
```

one two three

Used properties:

- `text-direction` (1)
- `word-space`

`\lower` *amount* (number) *arg* (markup)

Lower *arg* by the distance *amount*. A negative *amount* indicates raising; see also `\raise`.

```
\markup {
  one
  \lower #3
  two
  three
}
```

one three
two

`\pad-around` *amount* (number) *arg* (markup)

Add padding *amount* all around *arg*.

```
\markup {
  \box {
    default
  }
}
```

```

    }
    \hspace #2
    \box {
      \pad-around #0.5 {
        padded
      }
    }
  }
}

```

default	padded
---------	--------

`\pad-markup` *amount* (number) *arg* (markup)
Add space around a markup object.

```

\markup {
  \box {
    default
  }
  \hspace #2
  \box {
    \pad-markup #1 {
      padded
    }
  }
}

```

default	padded
---------	--------

`\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.

```

\markup {
  \box {
    default
  }
  \hspace #4
  \box {
    \pad-to-box #'(0 . 10) #'(0 . 3) {
      padded
    }
  }
}

```

default	padded
---------	--------

`\pad-x` *amount* (number) *arg* (markup)
Add padding *amount* around *arg* in the X direction.

```

\markup {
  \box {
    default
  }
}

```

```

\hspace #4
\box {
  \pad-x #2 {
    padded
  }
}
}

```

default	padded
---------	--------

`\put-adjacent` *axis* (integer) *dir* (direction) *arg1* (markup) *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`.

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.

```

\markup {
  C
  \small
  \bold
  \raise #1.0
  9/7+
}

```

C 9/7+

`\right-align` *arg* (markup)

Align *arg* on its right edge.

```

\markup {
  \column {
    one
    \right-align
    two
    three
  }
}

```

one
two
three

`\right-column` *args* (markup list)

Put *args* in a right-aligned column.

```
\markup {
  \right-column {
    one
    two
    three
  }
}
```

```
one
two
three
```

Used properties:

- `baseline-skip`

`\rotate` *ang* (number) *arg* (markup)

Rotate object with *ang* degrees around its center.

```
\markup {
  default
  \hspace #2
  \rotate #45
  \line {
    rotated 45°
  }
}
```

```
default
```

rotated 45°

`\translate` *offset* (pair of numbers) *arg* (markup)

Translate *arg* relative to its surroundings. *offset* is a pair of numbers representing the displacement in the X and Y axis.

```
\markup {
  *
  \translate #'(2 . 3)
  \line { translated two spaces right, three up }
}
```

```
translated two spaces right, three up
```

```
*
```

`\translate-scaled` *offset* (pair of numbers) *arg* (markup)

Translate *arg* by *offset*, scaling the offset by the `font-size`.

```
\markup {
  \fontsize #5 {
    * \translate #'(2 . 3) translate
    \hspace #2
    * \translate-scaled #'(2 . 3) translate-scaled
  }
}
```

* **translate** *

translate-scaled

Used properties:

- `font-size` (0)

`\vcenter` *arg* (markup)

Align *arg* to its Y center.

```
\markup {
  one
  \vcenter
  two
  three
}
```

one two three

`\vspace` *amount* (number)

Create an invisible object taking up vertical space of *amount* multiplied by 3.

```
\markup {
  \center-column {
    one
    \vspace #2
    two
    \vspace #5
    three
  }
}
```

one

two

three

`\wordwrap-field` *symbol* (symbol)

Wordwrap the data which has been assigned to *symbol*.

```
\header {
  title = "My title"
  myText = "Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat."
}
```

```
\paper {
  bookTitleMarkup = \markup {
```

```

\column {
  \fill-line { \fromproperty #'header:title }
  \null
  \wordwrap-field #'header:myText
}
}

\markup {
  \null
}

```

My title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

`\wordwrap` *args* (markup list)

Simple wordwrap. Use `\override #'(line-width . X)` to set the line width, where *X* is the number of staff spaces.

```

\markup {
  \wordwrap {
    Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
    Ut enim ad minim veniam, quis nostrud exercitation ullamco
    laboris nisi ut aliquip ex ea commodo consequat.
  }
}

```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width` (*#f*)
- `baseline-skip`

`\wordwrap-string` *arg* (string)

Wordwrap a string. Paragraphs may be separated with double newlines.

```

\markup {
  \override #'(line-width . 40)
  \wordwrap-string #"Lorem ipsum dolor sit amet, consectetur
    adipisicing elit, sed do eiusmod tempor incididunt ut labore

```

et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.

Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum"
}

Lorem ipsum dolor sit amet,
consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore
et dolore magna aliqua.
Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo
consequat.
Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia
deserunt mollit anim id est laborum

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width`
- `baseline-skip`

A.8.3 Graphic

`\arrow-head` *axis* (integer) *dir* (direction) *filled* (boolean)

Produce an arrow head in specified direction and axis. Use the filled head if *filled* is specified.

```
\markup {
  \fontsize #5 {
    \general-align #Y #DOWN {
      \arrow-head #Y #UP ##t
      \arrow-head #Y #DOWN ##f
      \hspace #2
      \arrow-head #X #RIGHT ##f
      \arrow-head #X #LEFT ##f
    }
  }
}
```

▲Y ><

`\beam` *width* (number) *slope* (number) *thickness* (number)

Create a beam with the specified parameters.

```
\markup {
  \beam #5 #1 #2
}
```



`\bracket arg (markup)`
Draw vertical brackets around *arg*.

```
\markup {
  \bracket {
    \note #"2." #UP
  }
}
```



`\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.

```
\markup {
  \circle {
    Hi
  }
}
```



Used properties:

- `circle-padding` (0.2)
- `font-size` (0)
- `thickness` (1)

`\draw-circle radius (number) thickness (number) filled (boolean)`
A circle of radius *radius* and thickness *thickness*, optionally filled.

```
\markup {
  \draw-circle #2 #0.5 ##f
  \hspace #2
  \draw-circle #2 #0 ##t
}
```



`\draw-line dest (pair of numbers)`
A simple line.

```
\markup {
  \draw-line #'(4 . 4)
  \override #'(thickness . 5)
  \draw-line #'(-3 . 0)
```

}



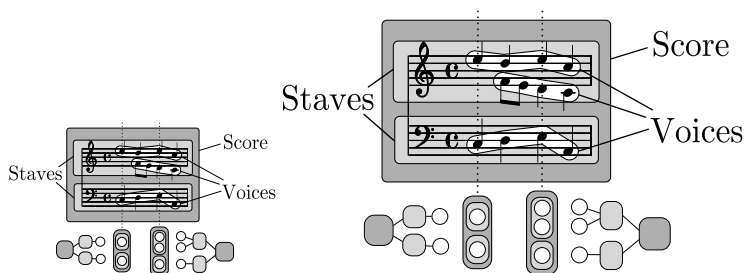
Used properties:

- `thickness` (1)

`\epsfile` *axis* (number) *size* (number) *file-name* (string)

Inline an EPS image. The image is scaled along *axis* to *size*.

```
\markup {
  \general-align #Y #DOWN {
    \epsfile #X #20 #"context-example.eps"
    \epsfile #Y #20 #"context-example.eps"
  }
}
```



`\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).

```
\markup {
  \filled-box #'(0 . 4) #'(0 . 4) #0
  \filled-box #'(0 . 2) #'(-4 . 2) #0.4
  \filled-box #'(1 . 8) #'(0 . 7) #0.2
  \with-color #white
  \filled-box #'(-4.5 . -2.5) #'(3.5 . 5.5) #0.7
}
```



`\hbracket` *arg* (markup)

Draw horizontal brackets around *arg*.

```
\markup {
  \hbracket {
    \line {
      one two three
    }
  }
}
```

}

one two three

`\parenthesize` *arg* (markup)

Draw parentheses around *arg*. This is useful for parenthesizing a column containing several lines of text.

```
\markup {
  \line {
    \parenthesize {
      \column {
        foo
        bar
      }
    }
    \override #'(angularity . 2) {
      \parenthesize {
        \column {
          bah
          baz
        }
      }
    }
  }
}
```

$\left(\begin{smallmatrix} \text{foo} \\ \text{bar} \end{smallmatrix}\right) \left(\begin{smallmatrix} \text{bah} \\ \text{baz} \end{smallmatrix}\right)$

Used properties:

- width (0.25)
- thickness (1)
- size (1)
- padding
- angularity (0)

`\postscript` *str* (string)

This inserts *str* directly into the output as a PostScript command string.

```
ringsps = #"
0.15 setlinewidth
0.9 0.6 moveto
0.4 0.6 0.5 0 361 arc
stroke
1.0 0.6 0.5 0 361 arc
stroke
"
```

```
rings = \markup {
  \with-dimensions #'(-0.2 . 1.6) #'(0 . 1.2)
  \postscript #ringsps
}
```

```
\relative c'' {
  c2^\rings
  a2_\rings
}
```



`\rounded-box` *arg* (markup)

Draw a box with rounded corners around *arg*. Looks at **thickness**, **box-padding** and **font-size** properties to determine line thickness and padding around the markup; the **corner-radius** property makes it possible to define another shape for the corners (default is 1).

```
c4^\markup {
  \rounded-box {
    Overtura
  }
}
c,8. c16 c4 r
```



Used properties:

- **box-padding** (0.5)
- **font-size** (0)
- **corner-radius** (1)
- **thickness** (1)

`\triangle` *filled* (boolean)

A triangle, either filled or empty.

```
\markup {
  \triangle ##t
  \hspace #2
  \triangle ##f
}
```



Used properties:

- **baseline-skip** (2)
- **font-size** (0)
- **thickness** (0.1)

`\with-url` *url* (string) *arg* (markup)

Add a link to URL *url* around *arg*. This only works in the PDF backend.

```
\markup {
  \with-url #"http://lilypond.org/web/" {
    LilyPond ... \italic {
      music notation for everyone
    }
  }
}
```

LilyPond ... *music notation for everyone*

A.8.4 Music

`\customTabClef` *num-strings* (integer) *staff-space* (number)

Draw a tab clef sans-serif style.

`\doubleflat`

Draw a double flat symbol.

```
\markup {
  \doubleflat
}
```



`\doublesharp`

Draw a double sharp symbol.

```
\markup {
  \doublesharp
}
```



`\flat`

Draw a flat symbol.

```
\markup {
  \flat
}
```



`\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 [Section “The Feta font” in *Notation Reference*](#) for a complete listing of the possible glyphs.

```
\markup {
  \musicglyph #"f"
  \musicglyph #"rests.2"
  \musicglyph #"clefs.G_change"
}
```



`\natural`

Draw a natural symbol.

```
\markup {
  \natural
}
```



`\note-by-number` *log* (number) *dot-count* (number) *dir* (number)

Construct a note symbol, with stem. By using fractional values for *dir*, longer or shorter stems can be obtained.

```
\markup {
  \note-by-number #3 #0 #DOWN
  \hspace #2
  \note-by-number #1 #2 #0.8
}
```



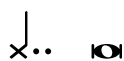
Used properties:

- `style '()`
- `font-size (0)`

`\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.

```
\markup {
  \override #'(style . cross) {
    \note #"4.." #UP
  }
  \hspace #2
  \note #"breve" #0
}
```



Used properties:

- `style '()`
- `font-size (0)`

`\score` *score* (score)

Inline an image of music.

```
\markup {
  \score {
    \new PianoStaff <<
      \new Staff \relative c' {
        \key f \major
        \time 3/4
        \mark \markup { Allegro }
        f2\p( a4)
        c2( a4)
      }
    }
}
```

```

        bes2( g'4)
        f8( e) e4 r
    }
    \new Staff \relative c {
        \clef bass
        \key f \major
        \time 3/4
        f8( a c a c a
        f c' es c es c)
        f,( bes d bes d bes)
        f( g bes g bes g)
    }
>>
\layout {
    indent = 0.0\cm
    \context {
        \Score
        \override RehearsalMark #'break-align-symbols =
            #'(time-signature key-signature)
        \override RehearsalMark #'self-alignment-X = #LEFT
    }
    \context {
        \Staff
        \override TimeSignature #'break-align-anchor-alignment = #LEFT
    }
}
}

```



Used properties:

- baseline-skip

`\semiflat`

Draw a semiflat symbol.

```

\markup {
    \semiflat
}

```

↓

`\semisharp`

Draw a semisharp symbol.

```

\markup {

```

```
\semisharp
}
```

♯

```
\sesquiflat
```

Draw a 3/2 flat symbol.

```
\markup {
  \sesquiflat
}
```

♭

```
\sesquisharp
```

Draw a 3/2 sharp symbol.

```
\markup {
  \sesquisharp
}
```

##

```
\sharp
```

Draw a sharp symbol.

```
\markup {
  \sharp
}
```

#

```
\tied-lyric str (string)
```

Like simple-markup, but use tie characters for ‘~’ tilde symbols.

```
\markup {
  \tied-lyric #"Lasciate~i monti"
}
```

Lasciate~i monti

A.8.5 Instrument Specific Markup

```
\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.

Used properties:

- **thickness** (0.5)
- **fret-diagram-details**
- **size** (1.0)
- **align-dir** (-0.4)

\fret-diagram-terse *definition-string* (string)

Make a fret diagram markup using terse string-based syntax.

Here is 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.

Used properties:

- **thickness** (0.5)
- **fret-diagram-details**
- **size** (1.0)
- **align-dir** (-0.4)

\fret-diagram-verbose *marking-list* (pair)

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*.

```
(capo fret-number)
```

Place a capo indicator (a large solid bar) across the entire fretboard at fret location *fret-number*. Also, set fret *fret-number* to be the lowest fret on the fret diagram.

```
(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.

Used properties:

- *thickness* (0.5)
- *fret-diagram-details*
- *size* (1.0)
- *align-dir* (-0.4)

```
\harp-pedal definition-string (string)
```

Make a harp pedal diagram.

Possible elements in *definition-string*:

```
^          pedal is up
```

```
-          pedal is neutral
```

```
v          pedal is down
```

```
|          vertical divider line
```

```
o          the following pedal should be circled (indicating a change)
```

The function also checks if the string has the typical form of three pedals, then the divider and then the remaining four pedals. If not it prints out a warning. However, in any case, it will also print each symbol in the order as given. This means you can place the divider (even multiple dividers) anywhere you want, but you’ll have to live with the warnings.

The appearance of the diagram can be tweaked inter alia using the *size* property of the TextScript grob (`\override Voice.TextScript #'size = #0.3`) for the overall, the *thickness* property (`\override Voice.TextScript #'thickness = #3`) for

the line thickness of the horizontal line and the divider. The remaining configuration (box sizes, offsets and spaces) is done by the `harp-pedal-details` list of properties (`\override Voice.TextScript #'harp-pedal-details #'box-width = #1`). It contains the following settings: `box-offset` (vertical shift of the box center for up/down pedals), `box-width`, `box-height`, `space-before-divider` (the spacing between two boxes before the divider) and `space-after-divider` (box spacing after the divider).

```
\markup \harp-pedal #"^-v|--ov^"
```



Used properties:

- `thickness` (0.5)
- `harp-pedal-details` ('())
- `size` (1.2)

`\woodwind-diagram` *instrument* (symbol) *input-list* (list)

Make a woodwind-instrument diagram. For example, say

```
\markup \woodwind-diagram #'oboe #'(1.4 0.1 #t ((lh . (d ees)) (cc . (five3qT1q))
```

for an oboe with the left-hand d key, left-hand ees key, and right-hand gis key depressed while the five-hole of the central column effectuates a trill between 1/4 and 3/4 closed.

The following instruments are supported:

- piccolo
- flute
- oboe
- clarinet
- bass-clarinet
- saxophone
- bassoon
- contrabassoon

To see all of the callable keys for a given instrument, include the function (`print-keys 'instrument`) in your `.ly` file, where `instrument` is the instrument whose keys you want to print.

Certain keys allow for special configurations. The entire gamut of configurations possible is as follows:

- 1q (1/4 covered)
- 1h (1/2 covered)
- 3q (3/4 covered)
- R (ring depressed)
- F (fully covered; the default if no state put)

Additionally, these configurations can be used in trills. So, for example, `three3qTR` effectuates a trill between 3/4 full and ring depressed on the three hole. As another example, `threeRT` effectuates a trill between R and open, whereas `threeTR` effectuates a trill between open and shut. To see all of the possibilities for all of the keys of a given instrument, invoke (`print-keys-verbose 'instrument`).

Lastly, substituting an empty list for the pressed-key alist will result in a diagram with all of the keys drawn but none filled. ie...

```
\markup \woodwind-diagram #'oboe #'(1.4 0.1 #t ())
```

A.8.6 Other

`\backslashed-digit` *num* (integer)

A feta number, with backslash. This is for use in the context of figured bass notation.

```
\markup {
  \backslashed-digit #5
  \hspace #2
  \override #'(thickness . 3)
  \backslashed-digit #7
}
```

5 7

Used properties:

- `thickness` (1.6)
- `font-size` (0)

`\char` *num* (integer)

Produce a single character. Characters encoded in hexadecimal format require the prefix `#x`.

```
\markup {
  \char #65 \char ##x00a9
}
```

A ©

`\eyeglasses`

Prints out eyeglasses, indicating strongly to look at the conductor.

```
\markup { \eyeglasses }
```

66

`\fraction` *arg1* (markup) *arg2* (markup)

Make a fraction of two markups.

```
\markup {
  \fraction 355 113
}
```

$\pi \approx \frac{355}{113}$

Used properties:

- `font-size` (0)

`\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.

```
\header {
  myTitle = "myTitle"
  title = \markup {
```

```

        from
        \italic
        \fromproperty #'header:myTitle
    }
}
\markup {
    \null
}

```

from *myTitle*

I

`\left-brace` *size* (number)

A feta brace in point size *size*.

```

\markup {
    \left-brace #35
    \hspace #2
    \left-brace #45
}

```

{ }

`\lookup` *glyph-name* (string)

Lookup a glyph by name.

```

\markup {
    \override #'(font-encoding . fetaBraces) {
        \lookup #"brace200"
        \hspace #2
        \rotate #180
        \lookup #"brace180"
    }
}

```

{ }

`\markalphabet` *num* (integer)

Make a markup letter for *num*. The letters start with A to Z and continue with double letters.

```

\markup {
    \markalphabet #8
    \hspace #2
    \markalphabet #26
}

```

I AA

`\markletter` *num* (integer)

Make a markup letter for *num*. The letters start with A to Z (skipping letter I), and continue with double letters.

```
\markup {
  \markletter #8
  \hspace #2
  \markletter #26
}
```

J AB

`\null`

An empty markup with extents of a single point.

```
\markup {
  \null
}
```

`\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 argument *new-prop* to the property list. Properties may be any property supported by Section “font-interface” in *Internals Reference*, Section “text-interface” in *Internals Reference* and Section “instrument-specific-markup-interface” in *Internals Reference*.

```
\markup {
  \line {
    \column {
      default
      baseline-skip
    }
    \hspace #2
    \override #'(baseline-skip . 4) {
      \column {
        increased
        baseline-skip
      }
    }
  }
}
```

default	increased
baseline-skip	baseline-skip

`\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.

`\right-brace` *size* (number)

A feta brace in point size *size*, rotated 180 degrees.

```
\markup {
  \right-brace #45
  \hspace #2
  \right-brace #35
}
```

$\left\{ \right\}$

`\slashed-digit` *num* (integer)

A feta number, with slash. This is for use in the context of figured bass notation.

```
\markup {
  \slashed-digit #5
  \hspace #2
  \override #'(thickness . 3)
  \slashed-digit #7
}
```

5 7

Used properties:

- `thickness` (1.6)
- `font-size` (0)

`\stencil` *stil* (stencil)

Use a stencil as markup.

```
\markup {
  \stencil #(make-circle-stencil 2 0 #t)
}
```



`\strut`

Create a box of the same height as the space in the current font.

`\transparent` *arg* (markup)

Make *arg* transparent.

```
\markup {
  \transparent {
    invisible text
  }
}
```

`\verbatim-file` *name* (string)

Read the contents of file *name*, and include it verbatim.

```

\markup {
  \verbatim-file #"simple.ly"
}

%% A simple piece in LilyPond, a scale.
\relative c' {
  c d e f g a b c
}
%% Optional helper for automatic updating by convert-ly. May be omitted.
\version "2.12.0"

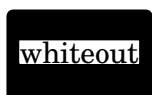
```

`\whiteout` *arg* (markup)
Provide a white background for *arg*.

```

\markup {
  \combine
    \filled-box #'(-1 . 10) #'(-3 . 4) #1
    \whiteout whiteout
}

```



`\with-color` *color* (color) *arg* (markup)
Draw *arg* in color specified by *color*.

```

\markup {
  \with-color #red
  red
  \hspace #2
  \with-color #green
  green
  \hspace #2
  \with-color #blue
  blue
}

```

red green blue

`\with-dimensions` *x* (pair of numbers) *y* (pair of numbers) *arg* (markup)
Set the dimensions of *arg* to *x* and *y*.

A.9 Text markup list commands

The following commands can all be used with `\markuplines`.

`\column-lines` *args* (markup list)

Like `\column`, but return a list of lines instead of a single markup. `baseline-skip` determines the space between each markup in *args*.

Used properties:

- `baseline-skip`

`\justified-lines` *args* (markup list)

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.

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width` (#f)
- `baseline-skip`

`\override-lines` *new-prop* (pair) *args* (markup list)

Like `\override`, for markup lists.

`\table-of-contents`

`\wordwrap-internal` *justify* (boolean) *args* (markup list)

Internal markup list command used to define `\justify` and `\wordwrap`.

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width` (#f)

`\wordwrap-lines` *args* (markup list)

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.

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width` (#f)
- `baseline-skip`

`\wordwrap-string-internal` *justify* (boolean) *arg* (string)

Internal markup list command used to define `\justify-string` and `\wordwrap-string`.

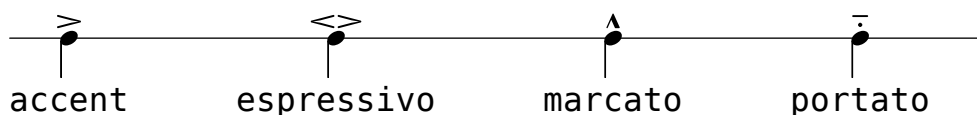
Used properties:

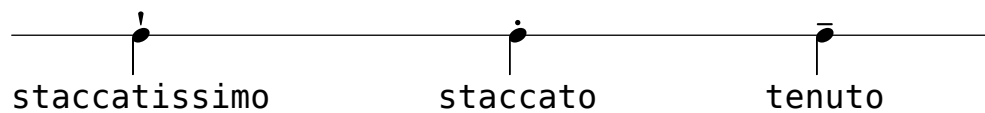
- `text-direction` (1)
- `word-space`
- `line-width`

A.10 List of articulations

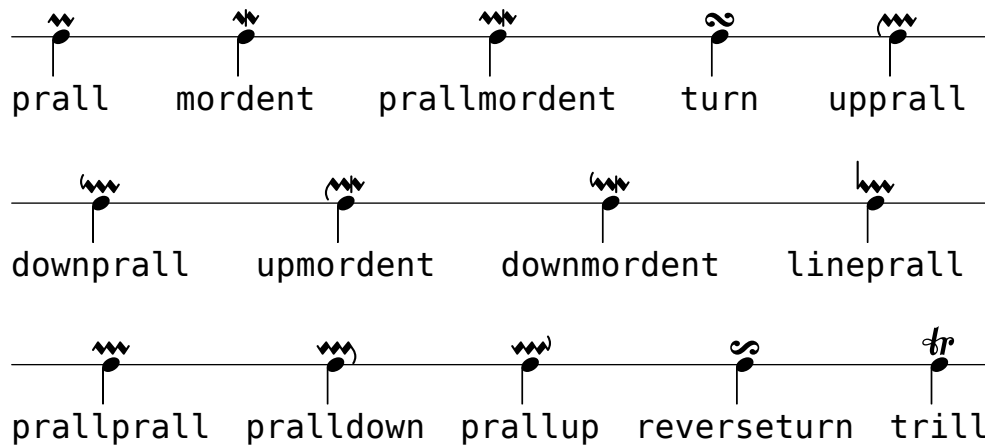
The following scripts are available in the Feta font and may be attached to notes (eg. ‘`c\accent`’).

Articulation scripts

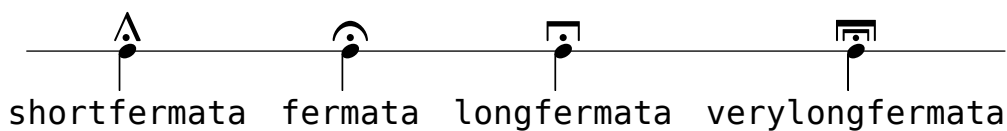




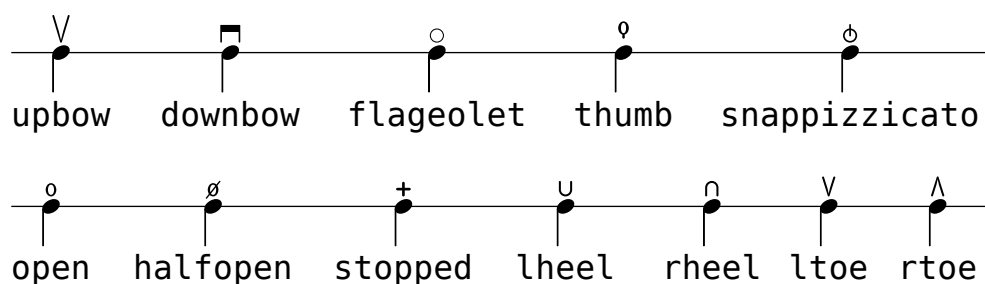
Ornament scripts



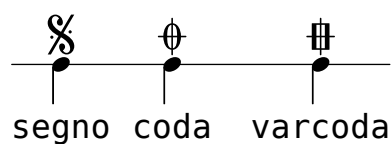
Fermata scripts



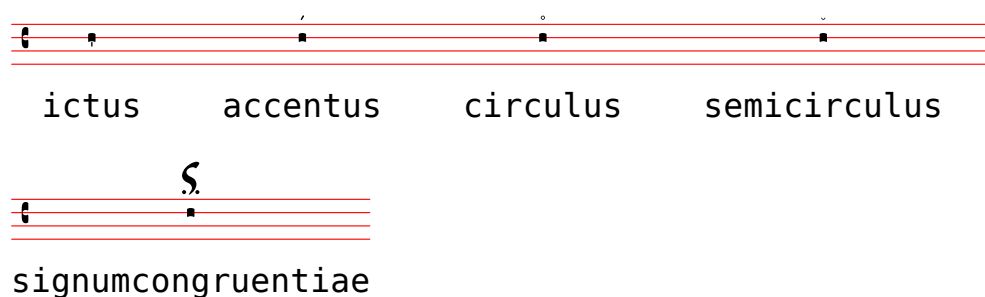
Instrument-specific scripts



Repeat sign scripts

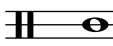
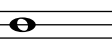
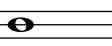
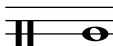
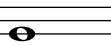
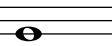

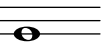
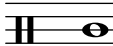
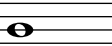
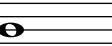
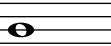
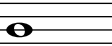
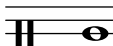
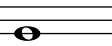
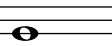




Ancient scripts



A.11 Percussion notes

acousticbassdrum: bda		bassdrum: bd		snare: sn		electricsnare: sne		acousticbassdrum: sna	
lowfloortom: tomfl		highfloortom: tomfh		lowtom: toml		hightom: tomh		lowmidtom: tomml	
closedhihat: hhc		hihat: hh		pedalhihat: hhp		openhihat: hho		halfopenhihat: hhho	
crashcymbala: cymca		crashcymbal: cymc		ridecymbala: cymra		ridecymbal: cymr			
chinesecymbal: cymch		splashcymbal: cymsc		crashcymbal: cymcb		ridecymbal: cymrb		ridebell: rbc	
mutehibongo: boh		mutehibongo: boh		openhibongo: boh		mutelobongo: bol		lobongo: bol	
mutehiconga: cgh		muteloconga: cgl		openhiconga: cgh		openloconga: cgl		loconga: cgl	
hitimbale: timh		lotimbale: timl		hiagogo: agh		loagogo: agl			
hisidestick: ssh		sidestick: ss		losidestick: ssl					
shortguiro: guis		longguiro: guil		guiro: gui		cabasa: cab		maracas: mar	
shortwhistle: whs				longwhistle: whl					
handclap: hc		tambourine: tamb		vibraslap: vib		tamtam: tt			

claves: cl	hiwoodblock: wbh	lowoodblock: wbl		
				
mutecuica: cuim opencuica: cuio mutetriangle: trim triangle: tri opentriangle: trio				
				
oneup: ua	twoup: ub	threeup: uc	fourup: ud	fiveup: ue
				
onedown: da	twodown: db	threedown: dc	fourdown: dd	fivedown: de
				

A.12 Technical glossary

A glossary of the technical terms and concepts used internally in LilyPond. These terms may appear in the manuals, on mailing lists or in the source code.

alist

An association list or **alist** for short is a Scheme pair which associates a value with a key: (key . value). For example, in ‘scm/lily.scm’, the alist “type-p-name-alist” associates certain type predicates (e.g. `ly:music?`) with names (e.g. “music”) so that type-check failures can be reported with a console message that includes the name of the expected type predicate.

callback

A **callback** is a routine, function or method whose reference is passed as an argument in a call to another routine, so allowing the called routine to invoke it. The technique enables a lower-level software layer to call a function defined in a higher layer. Callbacks are used extensively in LilyPond to permit user-level Scheme code to define how many low-level actions are performed.

closure

In Scheme, a **closure** is created when a function, usually a lambda expression, is passed as a variable. The closure contains the function’s code plus references to the lexical bindings of the function’s free variables (i.e. those variables used in the expression but defined outside it). When this function is applied to different arguments later, the free variable bindings that were captured in the closure are used to obtain the values of the free variables to be used in the calculation. One useful property of closures is the retention of internal variable values between invocations, so permitting state to be maintained.

A **simple closure** is a closure whose expression has no free variables and hence no free variable bindings.

A simple closure is represented in LilyPond by a smob containing the expression and a method to apply the expression to a passed list of arguments.

glyph

A **glyph** is a particular graphical representation of a typographic character, or a combination of two characters forming a ligature. A set of glyphs with a single style and shape comprise a font, and a set of fonts covering several styles and sizes comprise a typeface.

See also

Notation Reference: [Section 1.8.3 \[Fonts\]](#), page 195, [Section 3.3.3 \[Text encoding\]](#), page 344.

grob

LilyPond objects which represent items of notation in the printed output such as note heads, stems, slurs, ties, fingering, clefs, etc are called ‘Layout objects’, often known as ‘GRaphical Objects’, or **grobs** for short. They are represented by instances of the `Grob` class.

See also

Learning Manual: [Section “Objects and interfaces”](#) in *Learning Manual*, [Section “Naming conventions of objects and properties”](#) in *Learning Manual*, [Section “Properties of layout objects”](#) in *Learning Manual*.

Internals Reference: [Section “grob-interface”](#) in *Internals Reference*, [Section “All layout objects”](#) in *Internals Reference*.

immutable

An **immutable** object is one whose state cannot be modified after creation, in contrast to a mutable object, which can be modified after creation.

In LilyPond, immutable or shared properties define the default style and behavior of grobs. They are shared between many objects. In apparent contradiction to the name, they can be changed using `\override` and `\revert`.

See also

Notation Reference: [\[mutable\]](#), page 517.

interface

Actions and properties which are common to a number of grobs are grouped together in an object called a `grob-interface`, or just ‘interface’ for short.

See also

Learning Manual: [Section “Objects and interfaces”](#) in *Learning Manual*, [Section “Naming conventions of objects and properties”](#) in *Learning Manual*, [Section “Properties found in interfaces”](#) in *Learning Manual*.

Notation Reference: [Section 5.2.2 \[Layout interfaces\]](#), page 410.

Internals Reference: [Section “Graphical Object Interfaces”](#) in *Internals Reference*.

lexer

A **lexer** is a program which converts a sequence of characters into a sequence of tokens, a process called lexical analysis. The LilyPond lexer converts the stream obtained from an input ‘.ly’ file into a tokenized stream more suited to the next stage of processing - parsing, for which see [\[parser\]](#), page 518. The LilyPond lexer is built with Flex from the lexer file ‘lily/lexer.ll’ which contains the lexical rules. This file is part of the source code and is not included in the LilyPond binary installation.

mutable

A **mutable** object is one whose state can be modified after creation, in contrast to an immutable object, whose state is fixed at the time of creation.

In LilyPond, mutable properties contain values that are specific to one grob. Typically, lists of other objects or results from computations are stored in mutable properties.

See also

Notation Reference: [\[immutable\]](#), page 517.

output-def

An instance of the `Output-def` class contains the methods and data structures associated with an output block. Instances are created for midi, layout and paper blocks.

parser

A **parser** analyzes the sequence of tokens produced by a lexer to determine its grammatical structure, grouping the tokens progressively into larger groupings according to the rules of the grammar. If the sequence of tokens is valid the end product is a tree of tokens whose root is the grammar's start symbol. If this cannot be achieved the file is invalid and an appropriate error message is produced. The syntactic groupings and the rules for constructing the groupings from their parts for the LilyPond syntax are defined in `'lily/parser.yy'` and shown in Backus Normal Form (BNF) in [Appendix C \[LilyPond grammar\]](#), page 575. This file is used to build the parser during the program build by the parser generator, Bison. It is part of the source code and is not included in the LilyPond binary installation.

parser variable

These are variables defined directly in Scheme. Their direct use by users is strongly discouraged, because their scoping semantics can be confusing.

When the value of such a variable is changed in a `.ly` file, the change is global, and unless explicitly reverted, the new value will persist to the end of the file, affecting subsequent `\score` blocks as well as external files added with the `\include` command. This can lead to unintended consequences and in complex typesetting projects the consequent errors can be difficult to track down.

LilyPond uses the following parser variables:

- `afterGraceFraction`
- `musicQuotes`
- `mode`
- `output-count`
- `output-suffix`
- `parseStringResult`
- `partCombineListener`
- `pitchnames`
- `toplevel-bookparts`
- `toplevel-scores`
- `showLastLength`
- `showFirstLength`

prob

`Property Objects`, or **probs** for short, are instances of the `Prob` class, a simple base class for objects which have mutable and immutable property lists and the methods to manipulate them. The `Music` and `Stream_event` classes derive from `Prob`. Instances of the `Prob` class are also created to hold the formatted content of system grobs and titling blocks during page layout.

simple closure

See [\[closure\]](#), page 516.

smob

Smobs, or ScheMe Objects, are part of the mechanism used by Guile to export C and C++ objects to Scheme code. In LilyPond, smobs are created from C++ objects through macros. There are two types of smob objects: simple smobs, intended for simple immutable objects like numbers, and complex smobs, used for objects with identities. If you have access to the LilyPond sources, more information can be found in ‘`lily/includes/smob.hh`’.

stencil

An instance of the **stencil** class holds the information required to print a typographical object. It is a simple smob containing a confining box, which defines the vertical and horizontal extents of the object, and a Scheme expression which will print the object when evaluated. Stencils may be combined to form more complex stencils defined by a tree of Scheme expressions formed from the Scheme expressions of the component stencils.

The **stencil** property, which connects a grob to its stencil, is defined in the **grob-interface** interface.

See also

Internals Reference: [Section “grob-interface” in *Internals Reference*](#).

A.13 All context properties

aDueText (markup)

Text to print at a unisono passage.

alignAboveContext (string)

Where to insert newly created context in vertical alignment.

alignBassFigureAccidentals (boolean)

If true, then the accidentals are aligned in bass figure context.

alignBelowContext (string)

Where to insert newly created context in vertical 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 entry in the list is either a symbol or a procedure.

symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is [Section “Score” in *Internals Reference*](#) then all staves share accidentals, and if *context* is [Section “Staff” in *Internals Reference*](#) then all voices in the same staff share accidentals, but staves do not.

procedure The procedure represents an accidental rule to be applied to the previously specified context.

The procedure takes the following arguments:

context The current context to which the rule should be applied.

pitch The pitch of the note to be evaluated.

barnum The current bar number.

measurepos
The current measure position.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. (**#t** . **#f**) does not make sense.

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.

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 false then bar lines will not be printed automatically; they must be explicitly created with a **\bar** command. Unlike the **\cadenzaOn** keyword, measures are still counted. Bar line 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 **BassFigureEvents**, 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**.

beamSettings (list)

Specifies when automatically generated beams should begin and end, as well as beam subdivision behavior. See [Section “Setting automatic beam behavior” in Notation Reference](#) for more information.

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.
- chordNameLowercaseMinor** (boolean)
Downcase roots of minor chords?
- 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.
- completionBusy** (boolean)
Whether a completion-note head is playing.
- 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’ and ‘text’. 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)
The type of spanner to be used for decrescendi. Available values are ‘hairpin’ and ‘text’. If unset, a hairpin decrescendo is used.

`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 [Section “Timing_translator” in *Internals Reference*](#) at [Section “Score” in *Internals Reference*](#) level.

`doubleRepeatType` (string)

Set the default bar line for double repeats.

`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.

`fretLabels` (list)

A list of strings or Scheme-formatted markups containing, in the correct order, the labels to be used for lettered frets in tablature.

`gridInterval` (moment)

Interval for which to generate `GridPoints`.

`harmonicAccidentals` (boolean)

If set, harmonic notes in chords get accidentals.

`harmonicDots` (boolean)

If set, harmonic notes in dotted chords get dots.

`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 [Section “Lyrics” in *Internals Reference*](#) 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.

`includeGraceNotes` (boolean)

Do not ignore grace notes for [Section “Lyrics” in *Internals Reference*](#).

`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 `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` 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 `\quotes`.

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 four arguments: text, duration, count 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`.

<code>midiInstrument</code> (string)	Name of the MIDI instrument to use.
<code>midiMaximumVolume</code> (number)	Analogous to <code>midiMinimumVolume</code> .
<code>midiMinimumVolume</code> (number)	Set the minimum loudness for MIDI. Ranges from 0 to 1.
<code>minimumFret</code> (number)	The tablature auto string-selecting mechanism selects the highest string with a fret at least <code>minimumFret</code> .
<code>minimumPageTurnLength</code> (moment)	Minimum length of a rest for a page turn to be allowed.
<code>minimumRepeatLengthForPageTurn</code> (moment)	Minimum length of a repeated section for a page turn to be allowed within that section.
<code>noChordSymbol</code> (markup)	Markup to be displayed for rests in a <code>ChordNames</code> context.
<code>noteToFretFunction</code> (procedure)	Convert list of notes and list of defined strings to full list of strings and fret numbers. Parameters: The context, a list of note events, a list of tabstring events, and the fretboard grob if a fretboard is desired.
<code>ottavation</code> (markup)	If set, the text for an ottava spanner. Changing this creates a new text spanner.
<code>output</code> (music output)	The output produced by a score-level translator during music interpretation.
<code>pedalSostenutoStrings</code> (list)	See <code>pedalSustainStrings</code> .
<code>pedalSostenutoStyle</code> (symbol)	See <code>pedalSustainStyle</code> .
<code>pedalSustainStrings</code> (list)	A list of strings to print for sustain-pedal. Format is (<i>up updown down</i>), where each of the three is the string to print when this is done with the pedal.
<code>pedalSustainStyle</code> (symbol)	A symbol that indicates how to print sustain pedals: <code>text</code> , <code>bracket</code> or <code>mixed</code> (both).
<code>pedalUnaCordaStrings</code> (list)	See <code>pedalSustainStrings</code> .
<code>pedalUnaCordaStyle</code> (symbol)	See <code>pedalSustainStyle</code> .
<code>predefinedDiagramTable</code> (hash table)	The hash table of predefined fret diagrams to use in <code>FretBoards</code> .
<code>printKeyCancellation</code> (boolean)	Print restoration alterations before a key signature change.
<code>printOctaveNames</code> (boolean)	Print octave marks for the <code>NoteNames</code> 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.

rehearsalMark (integer)

The last rehearsal mark printed.

repeatCommands (list)

This property is a list of commands of the form (list 'volta x), where x is a string or #f. 'end-repeat is also accepted as a command.

repeatCountVisibility (procedure)

A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when **countPercentRepeats** is set.

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 **instrumentName**.

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 (markup)

The text for the start of a solo for voice ‘two’ when part-combining.

soloText (markup)

The text for the start of a solo when part-combining.

squashedPosition (integer)

Vertical position of squashing for [Section “Pitch_squash_engraver”](#) in *Internals Reference*.

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).

strokeFingeringOrientations (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. Called with three arguments: context, string number and, fret number. It returns the text as a markup.

tabStaffLineLayoutFunction (procedure)

A function determining the staff position of a tablature note head. Called with two arguments: the context and the string.

tempoHideNote (boolean)

Hide the note=count in tempo marks.

tempoText (markup)

Text for tempo marks.

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.

topLevelAlignment (boolean)

If true, the *Vertical-align-engraver* will create a *VerticalAlignment*; otherwise, it will create a *StaffGrouper*

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 [Section “bar-line-interface” in *Internals Reference*](#).

A.14 Layout properties

add-stem-support (boolean)

If set, the **Stem** object is included in this script's support.

after-last-staff-spacing (list)

An alist of spacing variables that controls the spacing after the last staff in this staff group. See *next-staff-spacing* for a description of the elements of this alist.

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 bar line will be created below this bar line.

alteration (number)

Alteration numbers for accidental.

alteration-alist (list)

List of (*pitch* . *accidental*) pairs for key signature.

annotation (string)

Annotate a grob for debug purposes.

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 **inside**, **outside**, **around**, and **ignore**. **inside** adjusts the slur if needed to keep the grob inside the slur. **outside** moves the grob vertically to the outside of the slur. **around** moves the grob vertically to the outside of the slur only if there is a collision. **ignore** does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), **outside** and **around** behave like **ignore**.

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.

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.

beamlet-default-length (pair)

A pair of numbers. The first number specifies the default length of a beamlet that sticks out of the left hand side of this stem; the second number specifies the default length of the beamlet to the right. The actual length of a beamlet is determined by taking either the default length or the length specified by **beamlet-max-length-proportion**, whichever is smaller.

beamlet-max-length-proportion (pair)

The maximum length of a beamlet, as a proportion of the distance between two adjacent stems.

before-line-breaking (boolean)

Dummy property, used to trigger a callback function.

between-cols (pair)

Where to attach a loose column to.

between-staff-spacing (list)

An alist of spacing variables that controls the spacing between staves within this staff group. See *next-staff-spacing* for a description of the elements of this alist.

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 bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

break-align-anchor-alignment (number)

Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's 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). Choices are `left-edge`, `ambitus`, `breathing-sign`, `clef`, `staff-bar`, `key-cancellation`, `key-signature`, `time-signature`, and `custos`.

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.

circled-tip (boolean)

Put a circle at start/end of hairpins (al/del niente).

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 (color)

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-definition (pair)

List of **dash-elements** defining the dash structure. Each **dash-element** has a starting t value, an ending t-value, a **dash-fraction**, and a **dash-period**.

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.

default-next-staff-spacing (list)

An alist of spacing variables that controls the spacing between this staff and the next. See *next-staff-spacing* for a description of the elements of this alist.

details (list)

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a **details** property.

digit-names (vector)

Names for string finger digits.

direction (direction)

If **side-axis** is 0 (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**.

dot-count (integer)

The number of dots.

dot-negative-kern (number)

The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

dot-placement-list (list)

List consisting of (*description string-number fret-number finger-number*) entries used to define fret diagrams.

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-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-height (pair of numbers)

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to (*-inf.0 . +inf.0*).

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*).

extra-X-extent (pair of numbers)

A grob is enlarged in X dimension by this much.

extra-Y-extent (pair of numbers)

A grob is enlarged in Y dimension by this much.

flag (stencil)

A function returning the full flag stencil for the **Stem**, which is passed to the function as the only argument. The default `ly:stem::calc-stencil` function uses the **flag-style** property to determine the correct glyph for the flag. By providing your own function, you can create arbitrary flags.

flag-count (number)

The number of tremolo beams.

flag-style (symbol)

A symbol determining what style of flag glyph is typeset on a **Stem**. Valid options include '()' for standard flags, 'mensural' and 'no-flag', which switches off the flag.

font-encoding (symbol)

The font encoding is the broadest category for selecting a font. Currently, only LilyPond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

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 [Section "note-collision-interface" in *Internals Reference*](#).

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-diagram-details (list)

An alist of detailed grob properties for fret diagrams. Each alist entry consists of a (`property . value`) pair. The properties which can be included in `fret-diagram-details` include the following:

- `barre-type` – Type of barre indication used. Choices include `curved`, `straight`, and `none`. Default `curved`.
- `capo-thickness` – Thickness of capo indicator, in multiples of fret-space. Default value 0.5.
- `dot-color` – Color of dots. Options include `black` and `white`. Default `black`.
- `dot-label-font-mag` – Magnification for font used to label fret dots. Default value 1.
- `dot-position` – Location of dot in fret space. Default 0.6 for dots without labels, `0.95-dot-radius` for dots with labels.
- `dot-radius` – Radius of dots, in terms of fret spaces. Default value 0.425 for labeled dots, 0.25 for unlabeled dots.
- `finger-code` – Code for the type of fingering indication used. Options include `none`, `in-dot`, and `below-string`. Default `none` for markup fret diagrams, `below-string` for FretBoards fret diagrams.
- `fret-count` – The number of frets. Default 4.
- `fret-label-font-mag` – The magnification of the font used to label the lowest fret number. Default 0.5.

- **fret-label-vertical-offset** – The offset of the fret label from the center of the fret in direction parallel to strings. Default 0.
- **label-dir** – Side to which the fret label is attached. -1, #LEFT, or #DOWN for left or down; 1, #RIGHT, or #UP for right or up. Default #RIGHT.
- **mute-string** – Character string to be used to indicate muted string. Default "x".
- **number-type** – Type of numbers to use in fret label. Choices include **roman-lower**, **roman-upper**, and **arabic**. Default **roman-lower**.
- **open-string** – Character string to be used to indicate open string. Default "o".
- **orientation** – Orientation of fret-diagram. Options include **normal**, **landscape**, and **opposing-landscape**. Default **normal**.
- **string-count** – The number of strings. Default 6.
- **string-label-font-mag** – The magnification of the font used to label fingerings at the string, rather than in the dot. Default value 0.6 for **normal** orientation, 0.5 for **landscape** and **opposing-landscape**.
- **string-thickness-factor** – Factor for changing thickness of each string in the fret diagram. Thickness of string k is given by $\text{thickness} * (1 + \text{string-thickness-factor}) ^ (k-1)$. Default 0.
- **top-fret-thickness** – The thickness of the top fret line, as a multiple of the standard thickness. Default value 3.
- **xo-font-magnification** – Magnification used for mute and open string indicators. Default value 0.5.
- **xo-padding** – Padding for open and mute indicators from top fret. Default value 0.25.

full-length-padding (number)

How much padding to use at the right side of a full-length tuplet bracket.

full-length-to-extent (boolean)

Run to the extent of the column for a full-length tuplet bracket.

full-measure-extra-space (number)

Extra space that is allocated at the beginning of a measure with only one note. This property is read from the `NonMusicalPaperColumn` that begins the measure.

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 (string)

The glyph name within the font.

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.

harp-pedal-details (list)

An alist of detailed grob properties for harp pedal diagrams. Each alist entry consists of a (*property* . *value*) pair. The properties which can be included in harp-pedal-details include the following:

- **box-offset** – Vertical shift of the center of flat/sharp pedal boxes above/below the horizontal line. Default value 0.8.
- **box-width** – Width of each pedal box. Default value 0.4.
- **box-height** – Height of each pedal box. Default value 1.0.
- **space-before-divider** – Space between boxes before the first divider (so that the diagram can be made symmetric). Default value 0.8.
- **space-after-divider** – Space between boxes after the first divider. Default value 0.8.
- **circle-thickness** – Thickness (in unit of the line-thickness) of the ellipse around circled pedals. Default value 0.5.
- **circle-x-padding** – Padding in X direction of the ellipse around circled pedals. Default value 0.15.
- **circle-y-padding** – Padding in Y direction of the ellipse around circled pedals. Default value 0.2.

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.

hide-tied-accidental-after-break (boolean)

If set, an accidental that appears on a tied note after a line break will not be displayed.

horizontal-shift (integer)

An integer that identifies ranking of **NoteColumns** for horizontal shifting. This is used by [Section “note-collision-interface”](#) in *Internals Reference*.

horizontal-skylines (pair of skylines)

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?

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.

inter-loose-line-spacing (list)

Specifies how to vertically position a non-spaced line relative to the other non-spaced lines around it. See *next-staff-spacing* for the format of this list.

inter-staff-spacing (list)

Specifies how to vertically position a non-spaced line relative to the staff for which it has affinity. See *next-staff-spacing* for the format of this list.

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 objects 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.

labels (list)

List of labels (symbols) placed on a column.

layer (integer)

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 prefers 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 [Section “Formatting text” in *Notation Reference*](#).
- 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 [Section “note-collision-interface” in *Internals Reference*](#).
merge-differently-headed only applies to opposing stem directions (i.e., voice 1 & 2).
- 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).
- 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)
Minimum size of an object in Y dimension, measured in `staff-space` units.

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 (graphical (layout) object)

Object that is next relation (e.g., the lyric syllable following an extender).

next-staff-spacing (list)

An alist of properties used to position the next staff in the system. The symbols that can be defined in the alist are

- *space* – the amount of stretchable space between the center of this staff and the center of the next staff;
- *padding* – the minimum amount of whitespace that must be present between this staff and the next staff;
- *stretchability* – the ease with which the stretchable space increases when the system to which this staff belongs is stretched. If this is zero, the distance to the next staff will be fixed either at *space* or at *padding* plus the minimum distance to ensure there is no overlap, whichever is larger;
- *minimum-distance* – the minimum distance to place between the center of this staff and the center of the next. This differs from *padding* in that the height of a staff has no effect on the application of *minimum-distance* (whereas the height of a staff is crucial for *padding*).

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-affinity-spacing (list)

An alist of spacing variables that controls the spacing from a loose line (see *staff-affinity*) to the staff for which the loose line does not have affinity. See *next-staff-spacing* for a description of the elements of this alist.

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.

outside-staff-horizontal-padding (number)

By default, an outside-staff-object can be placed so that it is 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 neighbor.

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.

padding-pairs (list)

An alist mapping (*name* . *name*) to distances.

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 prefers 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 prefers 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.

prefer-dotted-right (boolean)

For note collisions, prefer to shift dotted up-note to the right, rather than shifting just the dot.

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 an 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. Other numerical values 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 [Section “spacing-spanner-interface” in *Internals Reference*](#).

shortest-playing-duration (moment)

The duration of the shortest note 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 `0`), the object is placed horizontally next to the other object. If the value is `#Y` or `1`, 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.

skyline-horizontal-padding (number)

For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

slash-negative-kern (number)

The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

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 bar line instead of to the beginning of the non-musical column. If there is a clef change followed by a bar line, 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 [Section “spacing-spanner-interface” in *Internals Reference*](#).

springs-and-rods (boolean)

Dummy variable for triggering spacing routines.

stacking-dir (direction)

Stack objects in which direction?

staff-affinity (direction)

The direction of the staff to which this line should stick.

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 dynamics **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)

An (*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 be a stem over a rest?

stencil (stencil)

The symbol to print.

stencils (list)

Multiple stencils, used as intermediate value.

strict-grace-spacing (boolean)

If set, main notes are spaced normally, then grace notes are put left of the musical columns for the main notes.

strict-note-spacing (boolean)

If set, unbroken columns with non-musical material (clefs, bar lines, etc.) are not spaced separately, but put before musical columns.

- 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 [Section “Formatting text” in *Notation Reference*](#).
- 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.
- 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 the bar line just before it would otherwise stop.
- toward-stem-shift** (number)
Amount by which scripts are shifted toward the stem if their direction coincides with the stem direction. 0.0 means keep the default position (centered on the note head), 1.0 means centered on the stem. Interpolated values are possible.
- 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** (pair of skylines)
Two skylines, one above and one below this grob.
- when** (moment)
Global time step associated with this column happen?
- whiteout** (boolean)
If true, the grob is printed over a white background to white-out underlying material, if the grob is visible. Usually #f by default.
- 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.

- 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)
Hard coded extent in Y direction.
- Y-offset** (number)
The vertical amount that this object is moved relative to its Y-parent.
- 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.

A.15 Available music functions

- acciaccatura** - *music* (music)
Create an acciaccatura from the following music expression
- addChordShape** - *key-symbol* (symbol) *tuning* (pair) *shape-definition* (string or pair)
Add chord shape *shape-definition* to the *chord-shape-table* hash with the key (*cons key-symbol tuning*).
- addInstrumentDefinition** - *name* (string) *lst* (list)
Create instrument *name* with properties *list*.
- addQuote** - *name* (string) *music* (music)
Define *music* as a quotable music expression named *name*
- afterGrace** - *main* (music) *grace* (music)
Create *grace* note(s) after a *main* music expression.
- allowPageTurn**
Allow a page turn. May be used at toplevel (ie between scores or markups), or inside a score.
- applyContext** - *proc* (procedure)
Modify context properties with Scheme procedure *proc*.
- applyMusic** - *func* (procedure) *music* (music)
Apply procedure *func* to *music*.
- applyOutput** - *ctx* (symbol) *proc* (procedure)
Apply function *proc* to every layout object in context *ctx*
- appoggiatura** - *music* (music)
Create an appoggiatura from *music*
- assertBeamQuant** - *l* (pair) *r* (pair)
Testing function: check whether the beam quant *l* and *r* are correct
- assertBeamSlope** - *comp* (procedure)
Testing function: check whether the slope of the beam is the same as *comp*
- autochange** - *music* (music)
Make voices that switch between staves automatically

- balloonGrobText** - *grob-name* (symbol) *offset* (pair of numbers) *text* (markup)
Attach *text* to *grob-name* at offset *offset* (use like `\once`)
- balloonText** - *offset* (pair of numbers) *text* (markup)
Attach *text* at *offset* (use like `\tweak`)
- bar** - *type* (string)
Insert a bar line of type *type*
- barNumberCheck** - *n* (integer)
Print a warning if the current bar number is not *n*.
- bendAfter** - *delta* (real number)
Create a fall or doit of pitch interval *delta*.
- bookOutputName** - *newfilename* (string)
Direct output for the current book block to *newfilename*.
- bookOutputSuffix** - *newsuffix* (string)
Set the output filename suffix for the current book block to *newsuffix*.
- breathe** Insert a breath mark.
- clef** - *type* (string)
Set the current clef to *type*.
- cueDuring** - *what* (string) *dir* (direction) *main-music* (music)
Insert contents of quote *what* corresponding to *main-music*, in a CueVoice oriented by *dir*.
- deadNote** - *note* (music)
Print *note* with a cross-shaped note head.
- defaultNoteHeads**
Revert to the default note head style.
- displayLilyMusic** - *music* (music)
Display the LilyPond input representation of *music* to the console.
- displayMusic** - *music* (music)
Display the internal representation of *music* to the console.
- endSpanners** - *music* (music)
Terminate the next spanner prematurely after exactly one note without the need of a specific end spanner.
- featherDurations** - *factor* (moment) *argument* (music)
Adjust durations of music in *argument* by rational *factor*.
- grace** - *music* (music)
Insert *music* as grace notes.
- harmonicNote** - *note* (music)
Print *note* with a diamond-shaped note head.
- harmonicsOn**
Set the default note head style to a diamond-shaped style.
- instrumentSwitch** - *name* (string)
Switch instrument to *name*, which must be predefined with `\addInstrumentDefinition`.
- keepWithTag** - *tag* (symbol) *music* (music)
Include only elements of *music* that are tagged with *tag*.

- killCues** - *music* (music)
Remove cue notes from *music*.
- label** - *label* (symbol)
Create *label* as a bookmarking label.
- makeClusters** - *arg* (music)
Display chords in *arg* as clusters.
- musicMap** - *proc* (procedure) *mus* (music)
Apply *proc* to *mus* and all of the music it contains.
- noPageBreak**
Forbid a page break. May be used at toplevel (i.e., between scores or markups), or inside a score.
- noPageTurn**
Forbid a page turn. May be used at toplevel (i.e., between scores or markups), or inside a score.
- octaveCheck** - *pitch-note* (music)
Octave check.
- ottava** - *octave* (number)
Set the octavation.
- overrideBeamSettings** - *context* (symbol) *time-signature* (pair) *rule-type* (symbol) *grouping-rule* (pair)
Override beamSettings in *context* for time signatures of *time-signature* and rules of type *rule-type* to have a grouping rule alist *grouping-rule*. *rule-type* can be **end** or **subdivide**, with a potential future value of **begin**. *grouping-rule* is an alist of (*beam-type* . *grouping*) entries. *grouping* is in units of *beam-type*. If *beam-type* is *, grouping is in units of the denominator of *time-signature*.
- overrideProperty** - *name* (string) *property* (symbol) *value* (any type)
Set *property* to *value* in all grobs named *name*. The *name* argument is a string of the form "Context.GrobName" or "GrobName".
- pageBreak**
Force a page break. May be used at toplevel (i.e., between scores or markups), or inside a score.
- pageTurn** Force a page turn between two scores or top-level markups.
- palmMute** - *note* (music)
Print *note* with a triangle-shaped note head.
- palmMuteOn**
Set the default note head style to a triangle-shaped style.
- parallelMusic** - *voice-ids* (list) *music* (music)
Define parallel music sequences, separated by '|' (bar check signs), and assign them to the identifiers provided in *voice-ids*.
voice-ids: a list of music identifiers (symbols containing only letters)
music: a music sequence, containing BarChecks as limiting expressions.
Example:

```
\parallelMusic #'(A B C) {  
  c c | d d | e e |  
  d d | e e | f f |
```

```

    }
<==>
    A = { c c | d d | }
    B = { d d | e e | }
    C = { e e | f f | }

```

parenthesize - *arg* (music)

Tag *arg* to be parenthesized.

partcombine - *part1* (music) *part2* (music)

Take the music in *part1* and *part2* and typeset so that they share a staff.

phrasingSlurDashPattern - *dash-fraction* (number) *dash-period* (number)

Set up a custom style of dash pattern for *dash-fraction* ratio of line to space repeated at *dash-period* interval.

pitchedTrill - *main-note* (music) *secondary-note* (music)

Print a trill with *main-note* as the main note of the trill and print *secondary-note* as a stemless note head in parentheses.

pointAndClickOff

Suppress generating extra code in final-format (e.g. pdf) files to point back to the lilypond source statement.

pointAndClickOn

Enable generation of code in final-format (e.g. pdf) files to reference the originating lilypond source statement; this is helpful when developing a score but generates bigger final-format files.

quoteDuring - *what* (string) *main-music* (music)

Indicate a section of music to be quoted. *what* indicates the name of the quoted voice, as specified in an `\addQuote` command. *main-music* is used to indicate the length of music to be quoted; usually contains spacers or multi-measure rests.

removeWithTag - *tag* (symbol) *music* (music)

Remove elements of *music* that are tagged with *tag*.

resetRelativeOctave - *reference-note* (music)

Set the octave inside a `\relative` section.

revertBeamSettings - *context* (symbol) *time-signature* (pair) *rule-type* (symbol)

Revert beam settings in *context* for time signatures of *time-signature* and groups of type *group-type*. *group-type* can be **end** or **subdivide**.

rightHandFinger - *finger* (number or string)

Apply *finger* as a fingering indication.

scaleDurations - *fraction* (pair of numbers) *music* (music)

Multiply the duration of events in *music* by *fraction*.

setBeatGrouping - *grouping* (pair)

Set the beat grouping in the current time signature to *grouping*.

shiftDurations - *dur* (integer) *dots* (integer) *arg* (music)

Scale *arg* up by a factor of $2^{\text{dur} * (2 - (1/2)^{\text{dots}})}$.

slurDashPattern - *dash-fraction* (number) *dash-period* (number)

(undocumented; fixme)

spacingTweaks - *parameters* (list)

Set the system stretch, by reading the 'system-stretch' property of the 'parameters' assoc list.

- storePredefinedDiagram** - *chord* (music) *tuning* (pair) *diagram-definition* (string or pair)
Add predefined fret diagram defined by *diagram-definition* for the chord pitches *chord* and the stringTuning *tuning*.
- styledNoteHeads** - *style* (symbol) *heads* (list or symbol) *music* (music)
Set *heads* in *music* to *style*.
- tabChordRepetition**
Include the string information in a chord repetition.
- tag** - *tag* (symbol) *arg* (music)
Add *tag* to the **tags** property of *arg*.
- tieDashPattern** - *dash-fraction* (number) *dash-period* (number)
(undocumented; fixme)
- tocItem** - *text* (markup)
Add a line to the table of content, using the **tocItemMarkup** paper variable markup
- transposedCueDuring** - *what* (string) *dir* (direction) *pitch-note* (music) *main-music* (music)
Insert notes from the part *what* into a voice called **cue**, using the transposition defined by *pitch-note*. This happens simultaneously with *main-music*, which is usually a rest. The argument *dir* determines whether the cue notes should be notated as a first or second voice.
- transposition** - *pitch-note* (music)
Set instrument transposition
- tweak** - *sym* (symbol) *val* (any type) *arg* (music)
Add *sym* . *val* to the **tweaks** property of *arg*.
- unfoldRepeats** - *music* (music)
Force any `\repeat volta`, `\repeat tremolo` or `\repeat percent` commands in *music* to be interpreted as `\repeat unfold`.
- withMusicProperty** - *sym* (symbol) *val* (any type) *music* (music)
Set *sym* to *val* in *music*.
- xNote** - *note* (music)
Print *note* with a cross-shaped note head.
- xNotesOn** Set the default note head style to a cross-shaped style.

A.16 Predefined type predicates

R5RS primary predicates

Type predicate	Description
boolean?	boolean
char?	character
number?	number
pair?	pair
port?	port
procedure?	procedure
string?	string
symbol?	symbol
vector?	vector

R5RS secondary predicates

Type predicate	Description
char-alphabetic?	alphabetic character
char-lower-case?	lower-case character
char-numeric?	numeric character
char-upper-case?	upper-case character
char-whitespace?	whitespace character
complex?	complex number
eof-object?	end-of-file object
even?	even number
exact?	exact number
inexact?	inexact number
input-port?	input port
integer?	integer
list?	list (<i>use cheap-list? for faster processing</i>)
negative?	negative number
null?	null
odd?	odd number
output-port?	output port
positive?	positive number
rational?	rational number
real?	real number
zero?	zero

Guile predicates

Type predicate	Description
hash-table?	hash table

LilyPond scheme predicates

Type predicate	Description
boolean-or-symbol?	boolean or symbol
cheap-list?	list (<i>use this instead of list? for faster processing</i>)
color?	color
grob-list?	list of grobs
list-or-symbol?	list or symbol
markup?	markup
markup-command-list?	markup command list
markup-list?	markup list
moment-pair?	pair of moment objects
number-or-grob?	number or grob
number-or-string?	number or string
number-pair?	pair of numbers
rhythmic-location?	rhythmic location
scheme?	any type
string-or-pair?	string or pair
string-or-symbol?	string or symbol

LilyPond exported predicates

Type predicate	Description
<code>ly:box?</code>	box
<code>ly:context?</code>	context
<code>ly:dimension?</code>	dimension, in staff space
<code>ly:dir?</code>	direction
<code>ly:dispatcher?</code>	dispatcher
<code>ly:duration?</code>	duration
<code>ly:font-metric?</code>	font metric
<code>ly:grob?</code>	graphical (layout) object
<code>ly:grob-array?</code>	array of grobs
<code>ly:input-location?</code>	input location
<code>ly:item?</code>	item
<code>ly:iterator?</code>	iterator
<code>ly:lily-lexer?</code>	lily-lexer
<code>ly:lily-parser?</code>	lily-parser
<code>ly:listener?</code>	listener
<code>ly:moment?</code>	moment
<code>ly:music?</code>	music
<code>ly:music-function?</code>	music function
<code>ly:music-list?</code>	list of music objects
<code>ly:music-output?</code>	music output
<code>ly:otf-font?</code>	OpenType font
<code>ly:output-def?</code>	output definition
<code>ly:page-marker?</code>	page marker
<code>ly:pango-font?</code>	pango font
<code>ly:paper-book?</code>	paper book
<code>ly:paper-system?</code>	paper-system Prob
<code>ly:pitch?</code>	pitch
<code>ly:prob?</code>	property object
<code>ly:score?</code>	score
<code>ly:simple-closure?</code>	simple closure
<code>ly:skyline?</code>	skyline
<code>ly:skyline-pair?</code>	pair of skylines
<code>ly:source-file?</code>	source file
<code>ly:spanner?</code>	spanner
<code>ly:stencil?</code>	stencil
<code>ly:stream-event?</code>	stream event
<code>ly:translator?</code>	translator
<code>ly:translator-group?</code>	translator group

A.17 Scheme functions

<code>ly:add-context-mod</code>	<i>contextmods</i> <i>modification</i>	[Function]
Adds the given context <i>modification</i> to the list <i>contextmods</i> of context modifications.		
<code>ly:add-file-name-alist</code>	<i>alist</i>	[Function]
Add mappings for error messages from <i>alist</i> .		
<code>ly:add-interface</code>	<i>iface</i> <i>desc</i> <i>props</i>	[Function]
Add a new grob interface. <i>iface</i> is the interface name, <i>desc</i> is the interface description, and <i>props</i> is the list of user-settable properties for the interface.		

ly:add-listener <i>list disp cl</i>	[Function]
Add the listener <i>list</i> to the dispatcher <i>disp</i> . Whenever <i>disp</i> hears an event of class <i>cl</i> , it is forwarded to <i>list</i> .	
ly:add-option <i>sym val description</i>	[Function]
Add a program option <i>sym</i> . <i>val</i> is the default value and <i>description</i> is a string description.	
ly:all-grob-interfaces	[Function]
Return the hash table with all grob 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 <i>key alist default-value strict-checking</i>	[Function]
Return value if <i>key</i> in <i>alist</i> , else <i>default-value</i> (or #f if not specified). If <i>strict-checking</i> is set to #t and <i>key</i> is not in <i>alist</i> , a <code>programming_error</code> is output.	
ly:axis-group-interface::add-element <i>grob grob-element</i>	[Function]
Set <i>grob</i> the parent of <i>grob-element</i> on all axes of <i>grob</i> .	
ly:beam-grouping <i>settings time-signature rule-type beam-type</i>	[Function]
Return grouping for beams of <i>beam-type</i> in <i>time-signature</i> for <i>rule-type</i> from <i>settings</i> .	
ly:beat-grouping <i>context</i>	[Function]
Return default beat grouping currently active in <i>context</i> .	
ly:book-add-bookpart! <i>book-smob book-part</i>	[Function]
Add <i>book-part</i> to <i>book-smob</i> book part list.	
ly:book-add-score! <i>book-smob score</i>	[Function]
Add <i>score</i> to <i>book-smob</i> score list.	
ly:book-process <i>book-smob default-paper default-layout output</i>	[Function]
Print book. <i>output</i> 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 <i>book-smob default-paper default-layout output</i>	[Function]
Print book. <i>output</i> 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? <i>x</i>	[Function]
Is <i>x</i> a <code>Box</code> object?	
ly:bp <i>num</i>	[Function]
<i>num</i> bigpoints (1/72th inch).	
ly:bracket <i>a iv t p</i>	[Function]
Make a bracket in direction <i>a</i> . The extent of the bracket is given by <i>iv</i> . The wings protrude by an amount of <i>p</i> , which may be negative. The thickness is given by <i>t</i> .	
ly:broadcast <i>disp ev</i>	[Function]
Send the stream event <i>ev</i> to the dispatcher <i>disp</i> .	

<code>ly:camel-case->lisp-identifier</code> <i>name-sym</i>	[Function]
Convert <code>FooBar_Bla</code> to <code>foo-bar-bla</code> style symbol.	
<code>ly:chain-assoc-get</code> <i>key achain default-value strict-checking</i>	[Function]
Return value for <i>key</i> from a list of alists <i>achain</i> . If no entry is found, return <i>default-value</i> or <code>#f</code> if <i>default-value</i> is not specified. With <i>strict-checking</i> set to <code>#t</code> , a <code>programming_error</code> is output in such cases.	
<code>ly:clear-anonymous-modules</code>	[Function]
Plug a GUILE 1.6 and 1.7 memory leak by breaking a weak reference pointer cycle explicitly.	
<code>ly:cm</code> <i>num</i>	[Function]
<i>num</i> cm.	
<code>ly:command-line-code</code>	[Function]
The Scheme code specified on command-line with <code>'-e'</code> .	
<code>ly:command-line-options</code>	[Function]
The Scheme options specified on command-line with <code>'-d'</code> .	
<code>ly:command-line-verbose?</code>	[Function]
Was <code>be_verbose_global</code> set?	
<code>ly:connect-dispatchers</code> <i>to from</i>	[Function]
Make the dispatcher <i>to</i> listen to events from <i>from</i> .	
<code>ly:context?</code> <i>x</i>	[Function]
Is <i>x</i> a <code>Context</code> object?	
<code>ly:context-current-moment</code> <i>context</i>	[Function]
Return the current moment of <i>context</i> .	
<code>ly:context-event-source</code> <i>context</i>	[Function]
Return <code>event-source</code> of context <i>context</i> .	
<code>ly:context-events-below</code> <i>context</i>	[Function]
Return a <code>stream-distributor</code> that distributes all events from <i>context</i> and all its subcontexts.	
<code>ly:context-find</code> <i>context name</i>	[Function]
Find a parent of <i>context</i> that has name or alias <i>name</i> . Return <code>#f</code> if not found.	
<code>ly:context-grob-definition</code> <i>context name</i>	[Function]
Return the definition of <i>name</i> (a symbol) within <i>context</i> as an alist.	
<code>ly:context-id</code> <i>context</i>	[Function]
Return the ID string of <i>context</i> , i.e., for <code>\context Voice = one ...</code> return the string <code>one</code> .	
<code>ly:context-name</code> <i>context</i>	[Function]
Return the name of <i>context</i> , i.e., for <code>\context Voice = one ...</code> return the symbol <code>Voice</code> .	
<code>ly:context-now</code> <i>context</i>	[Function]
Return <code>now-moment</code> of context <i>context</i> .	
<code>ly:context-parent</code> <i>context</i>	[Function]
Return the parent of <i>context</i> , <code>#f</code> if none.	

ly:context-property <i>context sym def</i>	[Function]
Return the value for property <i>sym</i> in <i>context</i> . If <i>def</i> is given, and property value is '()', return <i>def</i> .	
ly:context-property-where-defined <i>context name</i>	[Function]
Return the context above <i>context</i> where <i>name</i> is defined.	
ly:context-pushpop-property <i>context grob eltpop val</i>	[Function]
Do a single <code>\override</code> or <code>\revert</code> operation in <i>context</i> . The grob definition <i>grob</i> is extended with <i>eltpop</i> (if <i>val</i> is specified) or reverted (if unspecified).	
ly:context-set-property! <i>context name val</i>	[Function]
Set value of property <i>name</i> in context <i>context</i> to <i>val</i> .	
ly:context-unset-property <i>context name</i>	[Function]
Unset value of property <i>name</i> in context <i>context</i> .	
ly:default-scale	[Function]
Get the global default scale.	
ly:dimension? <i>d</i>	[Function]
Return <i>d</i> as a number. Used to distinguish length variables from normal numbers.	
ly:dir? <i>s</i>	[Function]
Is <i>s</i> a direction? Valid directions are -1, 0, or 1, where -1 represents left or down, 1 represents right or up, and 0 represents a neutral direction.	
ly:dispatcher? <i>x</i>	[Function]
Is <i>x</i> a <code>Dispatcher</code> object?	
ly:duration? <i>x</i>	[Function]
Is <i>x</i> a <code>Duration</code> object?	
ly:duration<? <i>p1 p2</i>	[Function]
Is <i>p1</i> shorter than <i>p2</i> ?	
ly:duration->string <i>dur</i>	[Function]
Convert <i>dur</i> to a string.	
ly:duration-dot-count <i>dur</i>	[Function]
Extract the dot count from <i>dur</i> .	
ly:duration-factor <i>dur</i>	[Function]
Extract the compression factor from <i>dur</i> . Return it as a pair.	
ly:duration-length <i>dur</i>	[Function]
The length of the duration as a <code>moment</code> .	
ly:duration-log <i>dur</i>	[Function]
Extract the duration log from <i>dur</i> .	
ly:effective-prefix	[Function]
Return effective prefix.	
ly:engraver-make-grob <i>engraver grob-name cause</i>	[Function]
Creates a grob originating from given engraver instance, with give <code>grob_name</code> , a symbol. <code>cause</code> should either be another grob or a music event.	

ly:error <i>str rest</i>	[Function]
A Scheme callable function to issue the error <i>str</i> . The error is formatted with format and <i>rest</i> .	
ly:eval-simple-closure <i>delayed closure scm-start scm-end</i>	[Function]
Evaluate a simple <i>closure</i> with the given <i>delayed</i> argument. If <i>scm-start</i> and <i>scm-end</i> are defined, evaluate it purely with those start and end points.	
ly:event-deep-copy <i>m</i>	[Function]
Copy <i>m</i> and all sub expressions of <i>m</i> .	
ly:event-property <i>sev sym</i>	[Function]
Get the property <i>sym</i> of stream event <i>mus</i> . If <i>sym</i> is undefined, return '().	
ly:event-set-property! <i>ev sym val</i>	[Function]
Set property <i>sym</i> in event <i>ev</i> to <i>val</i> .	
ly:expand-environment <i>str</i>	[Function]
Expand \$VAR and \${VAR} in <i>str</i> .	
ly:export <i>arg</i>	[Function]
Export a Scheme object to the parser so it is treated as an identifier.	
ly:find-file <i>name</i>	[Function]
Return the absolute file name of <i>name</i> , or #f if not found.	
ly:font-config-add-directory <i>dir</i>	[Function]
Add directory <i>dir</i> to FontConfig.	
ly:font-config-add-font <i>font</i>	[Function]
Add font <i>font</i> to FontConfig.	
ly:font-config-display-fonts	[Function]
Dump a list of all fonts visible to FontConfig.	
ly:font-config-get-font-file <i>name</i>	[Function]
Get the file for font <i>name</i> .	
ly:font-design-size <i>font</i>	[Function]
Given the font metric <i>font</i> , return the design size, relative to the current output-scale.	
ly:font-file-name <i>font</i>	[Function]
Given the font metric <i>font</i> , return the corresponding file name.	
ly:font-get-glyph <i>font name</i>	[Function]
Return a stencil from <i>font</i> for the glyph named <i>name</i> . If the glyph is not available, return an empty stencil.	
Note that this command can only be used to access glyphs from fonts loaded with ly:system-font-load ; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings fetaMusic and fetaBraces , respectively.	
ly:font-glyph-name-to-charcode <i>font name</i>	[Function]
Return the character code for glyph <i>name</i> in <i>font</i> .	
Note that this command can only be used to access glyphs from fonts loaded with ly:system-font-load ; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings fetaMusic and fetaBraces , respectively.	

- ly:font-glyph-name-to-index** *font name* [Function]
 Return the index for *name* in *font*.
 Note that this command can only be used to access glyphs from fonts loaded with **ly:system-font-load**; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings **fetaMusic** and **fetaBraces**, respectively.
- ly:font-index-to-charcode** *font index* [Function]
 Return the character code for *index* in *font*.
 Note that this command can only be used to access glyphs from fonts loaded with **ly:system-font-load**; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings **fetaMusic** and **fetaBraces**, respectively.
- 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**. Basic support for **~s** is also provided.
- 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-context-mods** *contextmod* [Function]
 Returns the list of context modifications stored in *contextmod*.
- 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** *original* [Function]
 A Scheme wrapper function for **gettext**.
- ly:grob?** *x* [Function]
 Is *x* a **Grob** object?
- 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? <i>x</i>	[Function]
Is <i>x</i> a <code>Grob_array</code> object?	
ly:grob-array-length <i>grob-arr</i>	[Function]
Return the length of <i>grob-arr</i> .	
ly:grob-array-ref <i>grob-arr index</i>	[Function]
Retrieve the <i>index</i> th element of <i>grob-arr</i> .	
ly:grob-basic-properties <i>grob</i>	[Function]
Get the immutable properties of <i>grob</i> .	
ly:grob-common-refpoint <i>grob other axis</i>	[Function]
Find the common refpoint of <i>grob</i> and <i>other</i> for <i>axis</i> .	
ly:grob-common-refpoint-of-array <i>grob others axis</i>	[Function]
Find the common refpoint of <i>grob</i> and <i>others</i> (a grob-array) for <i>axis</i> .	
ly:grob-default-font <i>grob</i>	[Function]
Return the default font for grob <i>gr</i> .	
ly:grob-extent <i>grob refp axis</i>	[Function]
Get the extent in <i>axis</i> direction of <i>grob</i> relative to the grob <i>refp</i> .	
ly:grob-interfaces <i>grob</i>	[Function]
Return the interfaces list of grob <i>grob</i> .	
ly:grob-layout <i>grob</i>	[Function]
Get <code>\layout</code> definition from grob <i>grob</i> .	
ly:grob-object <i>grob sym</i>	[Function]
Return the value of a pointer in grob <i>g</i> of property <i>sym</i> . It returns '() (end-of-list) if <i>sym</i> is undefined in <i>g</i> .	
ly:grob-original <i>grob</i>	[Function]
Return the unbroken original grob of <i>grob</i> .	
ly:grob-parent <i>grob axis</i>	[Function]
Get the parent of <i>grob</i> . <i>axis</i> is 0 for the X-axis, 1 for the Y-axis.	
ly:grob-pq<? <i>a b</i>	[Function]
Compare two grob priority queue entries. This is an internal function.	
ly:grob-properties <i>grob</i>	[Function]
Get the mutable properties of <i>grob</i> .	
ly:grob-property <i>grob sym val</i>	[Function]
Return the value for property <i>sym</i> of <i>grob</i> . If no value is found, return <i>val</i> or '()' if <i>val</i> is not specified.	
ly:grob-property-data <i>grob sym</i>	[Function]
Return the value for property <i>sym</i> of <i>grob</i> , but do not process callbacks.	
ly:grob-relative-coordinate <i>grob refp axis</i>	[Function]
Get the coordinate in <i>axis</i> direction of <i>grob</i> relative to the grob <i>refp</i> .	
ly:grob-robust-relative-extent <i>grob refp axis</i>	[Function]
Get the extent in <i>axis</i> direction of <i>grob</i> relative to the grob <i>refp</i> , or (0,0) if empty.	

ly:grob-script-priority-less <i>a b</i>	[Function]
Compare two grobs by script priority. For internal use.	
ly:grob-set-nested-property! <i>grob symlist val</i>	[Function]
Set nested property <i>symlist</i> in grob <i>grob</i> to value <i>val</i> .	
ly:grob-set-object! <i>grob sym val</i>	[Function]
Set <i>sym</i> in grob <i>grob</i> to value <i>val</i> .	
ly:grob-set-parent! <i>grob axis parent-grob</i>	[Function]
Set <i>parent-grob</i> the parent of grob <i>grob</i> in axis <i>axis</i> .	
ly:grob-set-property! <i>grob sym val</i>	[Function]
Set <i>sym</i> in grob <i>grob</i> to value <i>val</i> .	
ly:grob-staff-position <i>sg</i>	[Function]
Return the Y-position of <i>sg</i> relative to the staff.	
ly:grob-suicide! <i>grob</i>	[Function]
Kill <i>grob</i> .	
ly:grob-system <i>grob</i>	[Function]
Return the system grob of <i>grob</i> .	
ly:grob-translate-axis! <i>grob d a</i>	[Function]
Translate <i>g</i> on axis <i>a</i> over distance <i>d</i> .	
ly:grouping-rules <i>settings time-signature rule-type</i>	[Function]
Return grouping rules for <i>time-signature</i> and <i>rule-type</i> from <i>settings</i> .	
ly:gulp-file <i>name size</i>	[Function]
Read the file <i>name</i> , and return its contents in a string. The file is looked up using the search path.	
ly:hash-table-keys <i>tab</i>	[Function]
Return a list of keys in <i>tab</i> .	
ly:inch <i>num</i>	[Function]
<i>num</i> inches.	
ly:input-both-locations <i>sip</i>	[Function]
Return input location in <i>sip</i> as (file-name first-line first-column last-line last-column).	
ly:input-file-line-char-column <i>sip</i>	[Function]
Return input location in <i>sip</i> as (file-name line char column).	
ly:input-location? <i>x</i>	[Function]
Is <i>x</i> an input-location?	
ly:input-message <i>sip msg rest</i>	[Function]
Print <i>msg</i> as a GNU compliant error message, pointing to the location in <i>sip</i> . <i>msg</i> is interpreted similar to format 's argument, using <i>rest</i> .	
ly:interpret-music-expression <i>mus ctx</i>	[Function]
Interpret the music expression <i>mus</i> in the global context <i>ctx</i> . The context is returned in its final state.	

ly:interpret-stencil-expression <i>expr func arg1 offset</i>	[Function]
Parse <i>expr</i> , feed bits to <i>func</i> with first arg <i>arg1</i> having offset <i>offset</i> .	
ly:intlog2 <i>d</i>	[Function]
The 2-logarithm of $1/d$.	
ly:is-listened-event-class <i>sym</i>	[Function]
Is <i>sym</i> a listened event class?	
ly:item? <i>g</i>	[Function]
Is <i>g</i> an <code>Item</code> object?	
ly:item-break-dir <i>it</i>	[Function]
The break status direction of item <i>it</i> . -1 means end of line, 0 unbroken, and 1 beginning of line.	
ly:iterator? <i>x</i>	[Function]
Is <i>x</i> a <code>Music_iterator</code> object?	
ly:lexer-keywords <i>lexer</i>	[Function]
Return a list of (KEY . CODE) pairs, signifying the LilyPond reserved words list.	
ly:lily-lexer? <i>x</i>	[Function]
Is <i>x</i> a <code>Lily_lexer</code> object?	
ly:lily-parser? <i>x</i>	[Function]
Is <i>x</i> a <code>Lily_parser</code> object?	
ly:listener? <i>x</i>	[Function]
Is <i>x</i> a <code>Listener</code> object?	
ly:make-book <i>paper header scores</i>	[Function]
Make a \book of <i>paper</i> and <i>header</i> (which may be <code>#f</code> as well) containing \scores.	
ly:make-book-part <i>scores</i>	[Function]
Make a \bookpart containing \scores.	
ly:make-dispatcher	[Function]
Return a newly created dispatcher.	
ly:make-duration <i>length dotcount num den</i>	[Function]
<i>length</i> 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 <i>dotcount</i> .	
The duration factor is optionally given by <i>num</i> and <i>den</i> .	
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 <i>output-def</i>	[Function]
Set up a global interpretation context, using the output block <i>output-def</i> . The context is returned.	
ly:make-global-translator <i>global</i>	[Function]
Create a translator group and connect it to the global context <i>global</i> . 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 indexing the global default scale, with 0 corresponding to pitch C and 6 usually corresponding to pitch B. *alter* is a rational number of 200-cent whole tones for alteration.
- ly:make-prob** *type init rest* [Function]
 Create a Prob object.
- ly:make-scale** *steps* [Function]
 Create a scale. The argument is a vector of rational numbers, each of which represents the number of 200 cent tones of a pitch above the tonic.
- 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 <i>expr text yext</i>	[Function]
Stencils are device independent output expressions. They carry two pieces of information:	
<ol style="list-style-type: none"> 1. A specification of how to print this object. This specification is processed by the output backends, for example ‘<code>scm/output-ps.scm</code>’. 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 <i>cl proplist</i>	[Function]
Create a stream event of class <i>cl</i> with the given mutable property list.	
ly:message <i>str rest</i>	[Function]
A Scheme callable function to issue the message <i>str</i> . The message is formatted with format and <i>rest</i> .	
ly:minimal-breaking <i>pb</i>	[Function]
Break (pages and lines) the Paper_book object <i>pb</i> without looking for optimal spacing: stack as many lines on a page before moving to the next one.	
ly:mm <i>num</i>	[Function]
<i>num</i> mm.	
ly:module->alist <i>mod</i>	[Function]
Dump the contents of module <i>mod</i> as an alist.	
ly:module-copy <i>dest src</i>	[Function]
Copy all bindings from module <i>src</i> into <i>dest</i> .	
ly:modules-lookup <i>modules sym def</i>	[Function]
Look up <i>sym</i> in the list <i>modules</i> , returning the first occurrence. If not found, return <i>def</i> or #f if <i>def</i> isn’t specified.	
ly:moment? <i>x</i>	[Function]
Is <i>x</i> a Moment object?	
ly:moment<? <i>a b</i>	[Function]
Compare two moments.	
ly:moment-add <i>a b</i>	[Function]
Add two moments.	
ly:moment-div <i>a b</i>	[Function]
Divide two moments.	
ly:moment-grace-denominator <i>mom</i>	[Function]
Extract denominator from grace timing.	
ly:moment-grace-numerator <i>mom</i>	[Function]
Extract numerator from grace timing.	
ly:moment-main-denominator <i>mom</i>	[Function]
Extract denominator from main timing.	
ly:moment-main-numerator <i>mom</i>	[Function]
Extract numerator from main timing.	
ly:moment-mod <i>a b</i>	[Function]
Modulo of two moments.	

<code>ly:moment-mul</code> <i>a b</i>	[Function]
Multiply two moments.	
<code>ly:moment-sub</code> <i>a b</i>	[Function]
Subtract two moments.	
<code>ly:music?</code> <i>obj</i>	[Function]
Is <i>obj</i> a music object?	
<code>ly:music-compress</code> <i>m factor</i>	[Function]
Compress music object <i>m</i> by moment <i>factor</i> .	
<code>ly:music-deep-copy</code> <i>m</i>	[Function]
Copy <i>m</i> and all sub expressions of <i>m</i> .	
<code>ly:music-duration-compress</code> <i>mus fact</i>	[Function]
Compress <i>mus</i> by factor <i>fact</i> , which is a Moment .	
<code>ly:music-duration-length</code> <i>mus</i>	[Function]
Extract the duration field from <i>mus</i> and return the length.	
<code>ly:music-function?</code> <i>x</i>	[Function]
Is <i>x</i> a <code>music-function</code> ?	
<code>ly:music-function-extract</code> <i>x</i>	[Function]
Return the Scheme function inside <i>x</i> .	
<code>ly:music-length</code> <i>mus</i>	[Function]
Get the length of music expression <i>mus</i> and return it as a Moment object.	
<code>ly:music-list?</code> <i>lst</i>	[Function]
Is <i>lst</i> a list of music objects?	
<code>ly:music-mutable-properties</code> <i>mus</i>	[Function]
Return an alist containing the mutable properties of <i>mus</i> . The immutable properties are not available, since they are constant and initialized by the <code>make-music</code> function.	
<code>ly:music-output?</code> <i>x</i>	[Function]
Is <i>x</i> a <code>Music_output</code> object?	
<code>ly:music-property</code> <i>mus sym val</i>	[Function]
Return the value for property <i>sym</i> of music expression <i>mus</i> . If no value is found, return <i>val</i> or '()' if <i>val</i> is not specified.	
<code>ly:music-set-property!</code> <i>mus sym val</i>	[Function]
Set property <i>sym</i> in music expression <i>mus</i> to <i>val</i> .	
<code>ly:music-transpose</code> <i>m p</i>	[Function]
Transpose <i>m</i> such that central C is mapped to <i>p</i> . Return <i>m</i> .	
<code>ly:note-column-accidentals</code> <i>note-column</i>	[Function]
Return the <code>AccidentalPlacement</code> grob from <i>note-column</i> if any, or <code>SCM_EOL</code> otherwise.	
<code>ly:note-column-dot-column</code> <i>note-column</i>	[Function]
Return the <code>DotColumn</code> grob from <i>note-column</i> if any, or <code>SCM_EOL</code> otherwise.	
<code>ly:note-head::stem-attachment</code> <i>font-metric glyph-name</i>	[Function]
Get attachment in <i>font-metric</i> for attaching a stem to notehead <i>glyph-name</i> .	

ly:number->string <i>s</i>	[Function]
Convert <i>num</i> to a string without generating many decimals.	
ly:optimal-breaking <i>pb</i>	[Function]
Optimally break (pages and lines) the <code>Paper_book</code> object <i>pb</i> to minimize badness in both vertical and horizontal spacing.	
ly:option-usage	[Function]
Print <code>ly:set-option</code> usage.	
ly:otf->cff <i>otf-file-name</i>	[Function]
Convert the contents of an OTF file to a CFF file, returning it as a string.	
ly:otf-font? <i>font</i>	[Function]
Is <i>font</i> an OpenType font?	
ly:otf-font-glyph-info <i>font glyph</i>	[Function]
Given the font metric <i>font</i> of an OpenType font, return the information about named glyph <i>glyph</i> (a string).	
ly:otf-font-table-data <i>font tag</i>	[Function]
Extract a table <i>tag</i> from <i>font</i> . Return empty string for non-existent <i>tag</i> .	
ly:otf-glyph-count <i>font</i>	[Function]
Return the number of glyphs in <i>font</i> .	
ly:otf-glyph-list <i>font</i>	[Function]
Return a list of glyph names for <i>font</i> .	
ly:output-def? <i>def</i>	[Function]
Is <i>def</i> an output definition?	
ly:output-def-clone <i>def</i>	[Function]
Clone output definition <i>def</i> .	
ly:output-def-lookup <i>def sym val</i>	[Function]
Return the value of <i>sym</i> in output definition <i>def</i> (e.g., <code>\paper</code>). If no value is found, return <i>val</i> or <code>()</code> if <i>val</i> is undefined.	
ly:output-def-parent <i>def</i>	[Function]
Return the parent output definition of <i>def</i> .	
ly:output-def-scope <i>def</i>	[Function]
Return the variable scope inside <i>def</i> .	
ly:output-def-set-variable! <i>def sym val</i>	[Function]
Set an output definition <i>def</i> variable <i>sym</i> to <i>val</i> .	
ly:output-description <i>output-def</i>	[Function]
Return the description of translators in <i>output-def</i> .	
ly:output-formats	[Function]
Formats passed to <code>--format</code> as a list of strings, used for the output.	
ly:outputter-close <i>outputter</i>	[Function]
Close port of <i>outputter</i> .	

<code>ly:outputter-dump-stencil</code> <i>outputter stencil</i>	[Function]
Dump stencil <i>expr</i> onto <i>outputter</i> .	
<code>ly:outputter-dump-string</code> <i>outputter str</i>	[Function]
Dump <i>str</i> onto <i>outputter</i> .	
<code>ly:outputter-module</code> <i>outputter</i>	[Function]
Return output module of <i>outputter</i> .	
<code>ly:outputter-output-scheme</code> <i>outputter expr</i>	[Function]
Eval <i>expr</i> in module of <i>outputter</i> .	
<code>ly:outputter-port</code> <i>outputter</i>	[Function]
Return output port for <i>outputter</i> .	
<code>ly:page-marker?</code> <i>x</i>	[Function]
Is <i>x</i> a <code>Page_marker</code> object?	
<code>ly:page-turn-breaking</code> <i>pb</i>	[Function]
Optimally break (pages and lines) the <code>Paper_book</code> object <i>pb</i> such that page turns only happen in specified places, returning its pages.	
<code>ly:pango-font?</code> <i>f</i>	[Function]
Is <i>f</i> a pango font?	
<code>ly:pango-font-physical-fonts</code> <i>f</i>	[Function]
Return alist of (ps-name file-name font-index) lists for Pango font <i>f</i> .	
<code>ly:paper-book?</code> <i>x</i>	[Function]
Is <i>x</i> a <code>Paper_book</code> object?	
<code>ly:paper-book-pages</code> <i>pb</i>	[Function]
Return pages in <code>Paper_book</code> object <i>pb</i> .	
<code>ly:paper-book-paper</code> <i>pb</i>	[Function]
Return the paper output definition (<code>\paper</code>) in <code>Paper_book</code> object <i>pb</i> .	
<code>ly:paper-book-performances</code> <i>pb</i>	[Function]
Return performances in <code>Paper_book</code> object <i>pb</i> .	
<code>ly:paper-book-scopes</code> <i>pb</i>	[Function]
Return scopes in <code>Paper_book</code> object <i>pb</i> .	
<code>ly:paper-book-systems</code> <i>pb</i>	[Function]
Return systems in <code>Paper_book</code> object <i>pb</i> .	
<code>ly:paper-fonts</code> <i>def</i>	[Function]
Return a list containing the fonts from output definition <i>def</i> (e.g., <code>\paper</code>).	
<code>ly:paper-get-font</code> <i>def chain</i>	[Function]
Find a font metric in output definition <i>def</i> satisfying the font-qualifiers in alist chain <i>chain</i> , and return it. (An alist chain is a list of alists, containing grob properties.)	
<code>ly:paper-get-number</code> <i>def sym</i>	[Function]
Return the value of variable <i>sym</i> in output definition <i>def</i> as a double.	
<code>ly:paper-outputscales</code> <i>def</i>	[Function]
Return the output-scale for output definition <i>def</i> .	

<code>ly:paper-score-paper-systems</code> <i>paper-score</i>	[Function]
Return vector of <code>paper_system</code> objects from <i>paper-score</i> .	
<code>ly:paper-system?</code> <i>obj</i>	[Function]
Is <i>obj</i> a C++ Prob object of type <code>paper-system</code> ?	
<code>ly:paper-system-minimum-distance</code> <i>sys1 sys2</i>	[Function]
Measure the minimum distance between these two paper-systems, using their stored skylines if possible and falling back to their extents otherwise.	
<code>ly:parse-file</code> <i>name</i>	[Function]
Parse a single .ly file. Upon failure, throw <code>ly-file-failed</code> key.	
<code>ly:parser-clear-error</code> <i>parser</i>	[Function]
Clear the error flag for the parser.	
<code>ly:parser-clone</code> <i>parser-smob</i>	[Function]
Return a clone of <i>parser-smob</i> .	
<code>ly:parser-define!</code> <i>parser-smob symbol val</i>	[Function]
Bind <i>symbol</i> to <i>val</i> in <i>parser-smob</i> 's module.	
<code>ly:parser-error</code> <i>parser msg input</i>	[Function]
Display an error message and make the parser fail.	
<code>ly:parser-has-error?</code> <i>parser</i>	[Function]
Does <i>parser</i> have an error flag?	
<code>ly:parser-lexer</code> <i>parser-smob</i>	[Function]
Return the lexer for <i>parser-smob</i> .	
<code>ly:parser-lookup</code> <i>parser-smob symbol</i>	[Function]
Look up <i>symbol</i> in <i>parser-smob</i> 's module. Return '()' if not defined.	
<code>ly:parser-output-name</code> <i>parser</i>	[Function]
Return the base name of the output file.	
<code>ly:parser-parse-string</code> <i>parser-smob ly-code</i>	[Function]
Parse the string <i>ly-code</i> with <i>parser-smob</i> . Upon failure, throw <code>ly-file-failed</code> key.	
<code>ly:parser-set-note-names</code> <i>parser names</i>	[Function]
Replace current note names in <i>parser</i> . <i>names</i> is an alist of symbols. This only has effect if the current mode is notes.	
<code>ly:parser-set-repetition-function</code> <i>parser fun</i>	[Function]
Replace the current repetition function in <i>parser</i> . <i>fun</i> is the new repetition function.	
<code>ly:parser-set-repetition-symbol</code> <i>parser sym</i>	[Function]
Replace the current repetition symbol in <i>parser</i> . <i>sym</i> is the new repetition symbol.	
<code>ly:performance-write</code> <i>performance filename</i>	[Function]
Write <i>performance</i> to <i>filename</i> .	
<code>ly:pfb->pfa</code> <i>pfb-file-name</i>	[Function]
Convert the contents of a PFB file to PFA.	
<code>ly:pitch?</code> <i>x</i>	[Function]
Is <i>x</i> a Pitch object?	

ly:pitch<? <i>p1 p2</i>	[Function]
Is <i>p1</i> lexicographically smaller than <i>p2</i> ?	
ly:pitch-alteration <i>pp</i>	[Function]
Extract the alteration from pitch <i>pp</i> .	
ly:pitch-diff <i>pitch root</i>	[Function]
Return pitch <i>delta</i> such that <i>pitch</i> transposed by <i>delta</i> equals <i>root</i> .	
ly:pitch-negate <i>p</i>	[Function]
Negate <i>p</i> .	
ly:pitch-notename <i>pp</i>	[Function]
Extract the note name from pitch <i>pp</i> .	
ly:pitch-octave <i>pp</i>	[Function]
Extract the octave from pitch <i>pp</i> .	
ly:pitch-quartertones <i>pp</i>	[Function]
Calculate the number of quarter tones of <i>pp</i> from middle C.	
ly:pitch-semitones <i>pp</i>	[Function]
Calculate the number of semitones of <i>pp</i> from middle C.	
ly:pitch-steps <i>p</i>	[Function]
Number of steps counted from middle C of the pitch <i>p</i> .	
ly:pitch-transpose <i>p delta</i>	[Function]
Transpose <i>p</i> by the amount <i>delta</i> , where <i>delta</i> is relative to middle C.	
ly:pointer-group-interface::add-grob <i>grob sym grob-element</i>	[Function]
Add <i>grob-element</i> to <i>grob</i> 's <i>sym</i> grob array.	
ly:position-on-line? <i>sg spos</i>	[Function]
Return whether <i>spos</i> is on a line of the staff associated with the grob <i>sg</i> (even on an extender line).	
ly:prob? <i>x</i>	[Function]
Is <i>x</i> a Prob object?	
ly:prob-immutable-properties <i>prob</i>	[Function]
Retrieve an alist of mutable properties.	
ly:prob-mutable-properties <i>prob</i>	[Function]
Retrieve an alist of mutable properties.	
ly:prob-property <i>prob sym val</i>	[Function]
Return the value for property <i>sym</i> of Prob object <i>prob</i> . If no value is found, return <i>val</i> or '()' if <i>val</i> is not specified.	
ly:prob-property? <i>obj sym</i>	[Function]
Is boolean prop <i>sym</i> set?	
ly:prob-set-property! <i>obj sym value</i>	[Function]
Set property <i>sym</i> of <i>obj</i> to <i>value</i> .	
ly:prob-type? <i>obj type</i>	[Function]
Is <i>obj</i> the specified prob-type?	

ly:programming-error <i>str rest</i>	[Function]
A Scheme callable function to issue the internal warning <i>str</i> . The message is formatted with format and <i>rest</i> .	
ly:progress <i>str rest</i>	[Function]
A Scheme callable function to print progress <i>str</i> . The message is formatted with format and <i>rest</i> .	
ly:property-lookup-stats <i>sym</i>	[Function]
Return hash table with a property access corresponding to <i>sym</i> . Choices are prob , grob , and context .	
ly:protects	[Function]
Return hash of protected objects.	
ly:pt <i>num</i>	[Function]
<i>num</i> printer points.	
ly:register-stencil-expression <i>symbol</i>	[Function]
Add <i>symbol</i> as head of a stencil expression.	
ly:relative-group-extent <i>elements common axis</i>	[Function]
Determine the extent of <i>elements</i> relative to <i>common</i> in the <i>axis</i> direction.	
ly:reset-all-fonts	[Function]
Forget all about previously loaded fonts.	
ly:round-filled-box <i>xext yext blot</i>	[Function]
Make a Stencil object that prints a black box of dimensions <i>xext</i> , <i>yext</i> and roundness <i>blot</i> .	
ly:round-filled-polygon <i>points blot</i>	[Function]
Make a Stencil object that prints a black polygon with corners at the points defined by <i>points</i> (list of coordinate pairs) and roundness <i>blot</i> .	
ly:run-translator <i>mus output-def</i>	[Function]
Process <i>mus</i> according to <i>output-def</i> . An interpretation context is set up, and <i>mus</i> is interpreted with it. The context is returned in its final state.	
Optionally, this routine takes an object-key to uniquely identify the score block containing it.	
ly:score? <i>x</i>	[Function]
Is <i>x</i> a Score object?	
ly:score-add-output-def! <i>score def</i>	[Function]
Add an output definition <i>def</i> to <i>score</i> .	
ly:score-embedded-format <i>score layout</i>	[Function]
Run <i>score</i> through <i>layout</i> (an output definition) scaled to correct output-scale already, returning a list of layout-lines.	
ly:score-error? <i>score</i>	[Function]
Was there an error in the score?	
ly:score-header <i>score</i>	[Function]
Return score header.	
ly:score-music <i>score</i>	[Function]
Return score music.	









ly:score-output-defs <i>score</i>	[Function]
All output definitions in a score.	
ly:score-set-header! <i>score module</i>	[Function]
Set the score header.	
ly:set-default-scale <i>scale</i>	[Function]
Set the global default scale. This determines the tuning of pitches with no accidentals or key signatures. The first pitch is C. Alterations are calculated relative to this scale. The number of pitches in this scale determines the number of scale steps that make up an octave. Usually the 7-note major scale.	
ly:set-grob-modification-callback <i>cb</i>	[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! <i>context</i>	[Function]
Set the <code>middleCPosition</code> variable in <i>context</i> based on the variables <code>middleCClefPosition</code> and <code>middleCOffset</code> .	
ly:set-option <i>var val</i>	[Function]
Set a program option.	
ly:set-property-cache-callback <i>cb</i>	[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? <i>clos</i>	[Function]
Is <i>clos</i> a simple closure?	
ly:skyline? <i>x</i>	[Function]
Is <i>x</i> a Skyline object?	
ly:skyline-pair? <i>x</i>	[Function]
Is <i>x</i> a Skyline_pair object?	
ly:smob-protects	[Function]
Return LilyPond's internal smob protection list.	
ly:solve-spring-rod-problem <i>springs rods length ragged</i>	[Function]
Solve a spring and rod problem for <i>count</i> objects, that are connected by <i>count</i> -1 <i>springs</i> , and an arbitrary number of <i>rods</i> . <i>count</i> is implicitly given by <i>springs</i> and <i>rods</i> . The <i>springs</i> argument has the format (<i>ideal</i> , <i>inverse_hook</i>) and <i>rods</i> is of the form (<i>idx1</i> , <i>idx2</i> , <i>distance</i>).	
<i>length</i> is a number, <i>ragged</i> a boolean.	
The function returns a list containing the force (positive for stretching, negative for compressing and <i>#f</i> for non-satisfied constraints) followed by <i>spring-count</i> +1 positions of the objects.	
ly:source-file? <i>x</i>	[Function]
Is <i>x</i> a Source_file object?	

ly:spanner? <i>g</i>	[Function]
Is <i>g</i> a spanner object?	
ly:spanner-bound <i>spanner dir</i>	[Function]
Get one of the bounds of <i>spanner</i> . <i>dir</i> is -1 for left, and 1 for right.	
ly:spanner-broken-into <i>spanner</i>	[Function]
Return broken-into list for <i>spanner</i> .	
ly:staff-symbol-line-thickness <i>grob</i>	[Function]
Returns the line-thickness of the staff associated with <i>grob</i> .	
ly:staff-symbol-staff-space <i>grob</i>	[Function]
Returns the staff-space of the staff associated with <i>grob</i> .	
ly:start-environment	[Function]
Return the environment (a list of strings) that was in effect at program start.	
ly:stderr-redirect <i>file-name mode</i>	[Function]
Redirect stderr to <i>file-name</i> , opened with <i>mode</i> .	
ly:stencil? <i>x</i>	[Function]
Is <i>x</i> a Stencil object?	
ly:stencil-add <i>args</i>	[Function]
Combine stencils. Takes any number of arguments.	
ly:stencil-aligned-to <i>stil axis dir</i>	[Function]
Align <i>stil</i> using its own extents. <i>dir</i> 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 <i>first axis direction second padding minimum</i>	[Function]
Construct a stencil by putting <i>second</i> next to <i>first</i> . <i>axis</i> can be 0 (x-axis) or 1 (y-axis). <i>direction</i> can be -1 (left or down) or 1 (right or up). The stencils are juxtaposed with <i>padding</i> as extra space. If this puts the reference points closer than <i>minimum</i> , they are moved by the latter amount. <i>first</i> and <i>second</i> may also be '()' or #f.	
ly:stencil-empty? <i>stil</i>	[Function]
Return whether <i>stil</i> is empty.	
ly:stencil-expr <i>stil</i>	[Function]
Return the expression of <i>stil</i> .	
ly:stencil-extent <i>stil axis</i>	[Function]
Return a pair of numbers signifying the extent of <i>stil</i> in <i>axis</i> direction (0 or 1 for x and y axis, respectively).	
ly:stencil-fonts <i>s</i>	[Function]
Analyze <i>s</i> , and return a list of fonts used in <i>s</i> .	
ly:stencil-in-color <i>stc r g b</i>	[Function]
Put <i>stc</i> in a different color.	
ly:stencil-rotate <i>stil angle x y</i>	[Function]
Return a stencil <i>stil</i> rotated <i>angle</i> degrees around the relative offset (x, y). E.g. an offset of (-1, 1) will rotate the stencil around the left upper corner.	

- ly:stencil-rotate-absolute** *stil angle x y* [Function]
Return a stencil *stil* rotated *angle* degrees around point (x, y), given in absolute coordinates.
- 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:stream-event?** *x* [Function]
Is *x* a `Stream_event` object?
- ly:string-percent-encode** *str* [Function]
Encode all characters in string *str* with hexadecimal percent escape sequences, with the following exceptions: characters -, ., /, and _; and characters in ranges 0-9, A-Z, and a-z.
- ly:string-substitute** *a b s* [Function]
Replace string *a* by string *b* in string *s*.
- ly:success** *str rest* [Function]
A Scheme callable function to issue a success message *str*. The message is formatted with *format* and *rest*.
- ly:system-font-load** *name* [Function]
Load the OpenType system font '*name.otf*'. Fonts loaded with this command must contain three additional SFNT font tables called LILC, LILF, and LILY, needed for typesetting musical elements. Currently, only the Emmentaler and the Emmentaler-Brace fonts fulfill these requirements.
Note that only **ly:font-get-glyph** and derived code (like `\lookup`) can access glyphs from the system fonts; text strings are handled exclusively via the Pango interface.
- 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 an alist chain, i.e. a list of alists. This is typically obtained with **(ly:grob-alist-chain grob (ly:output-def-lookup layout 'text-font-defaults))**. *markup* is the markup text to be processed.
- ly:translator?** *x* [Function]
Is *x* a `Translator` object?
- ly:translator-context** *trans* [Function]
Return the context of the translator object *trans*.
- 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: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 idx* [Function]
Convert the contents of a TrueType font file to PostScript Type 42 font, returning it as a string. The optional *idx* argument is useful for TrueType collections (TTC) only; it specifies the font index within the TTC. The default value of *idx* is 0.
- ly:ttf-ps-name** *ttf-file-name idx* [Function]
Extract the PostScript name from a TrueType font. The optional *idx* argument is useful for TrueType collections (TTC) only; it specifies the font index within the TTC. The default value of *idx* is 0.
- 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 B Cheat sheet

Syntax	Description	Example
<code>1 2 8 16</code>	durations	
<code>c4. c4..</code>	augmentation dots	
<code>c d e f g a b</code>	scale	
<code>fis bes</code>	alteration	
<code>\clef treble \clef bass</code>	clefs	
<code>\time 3/4 \time 4/4</code>	time signature	
<code>r4 r8</code>	rest	
<code>d ~ d</code>	tie	
<code>\key es \major</code>	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\s fz`

dynamics

`a\< a a\!`

crescendo



`a\> a a\!`

decrescendo

`< >`

chord

`\partial 8`

pickup / upbeat

`\times 2/3 {f g a}`

triplets

`\grace`

grace notes

`\lyricmode { twinkle }`

entering lyrics

twinkle

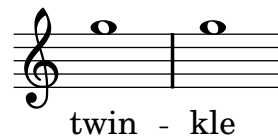
`\new Lyrics`

printing lyrics

twinkle

`twin -- kle`

lyric hyphen

`\chordmode { c:dim f:maj7 }`

chords

`\context ChordNames`

printing chord names

 $C^{\circ} F^{\Delta}$ `<<{e f} \ \ {c d}>>`

polyphony



s4 s8 s16

spacer rests

Appendix C LilyPond grammar

This appendix contains a description of the LilyPond grammar, as output from the parser.

Grammar

```

1 lilypond:  /* empty */
2           | lilypond toplevel_expression
3           | lilypond assignment
4           | lilypond error
5           | lilypond "\invalid"

6 toplevel_expression:  lilypond_header
7                       | book_block
8                       | bookpart_block
9                       | score_block
10                      | composite_music
11                      | full_markup
12                      | full_markup_list
13                      | output_def

14 embedded_scm:  SCM_TOKEN
15              | SCM_IDENTIFIER

16 lilypond_header_body:  /* empty */
17                      | lilypond_header_body assignment

18 lilypond_header:  "\header" '{' lilypond_header_body '}'

19 assignment_id:  STRING
20               | LYRICS_STRING

21 assignment:  assignment_id '=' identifier_init
22             | assignment_id property_path '=' identifier_init
23             | embedded_scm

24 identifier_init:  score_block
25                  | book_block
26                  | bookpart_block
27                  | output_def
28                  | context_def_spec_block
29                  | music
30                  | post_event
31                  | number_expression
32                  | string
33                  | embedded_scm
34                  | full_markup
35                  | DIGIT
36                  | context_modification

37 context_def_spec_block:  "\context" '{' context_def_spec_body '}'

38 context_def_spec_body:  /* empty */

```

```

39             | CONTEXT_DEF_IDENTIFIER
40             | context_def_spec_body
               "\grobdescriptions"
               embedded_scm
41             | context_def_spec_body context_mod
42             | context_def_spec_body context_modification

43 book_block:  "\book" '{' book_body '}'

44 book_body:  /* empty */
45             | BOOK_IDENTIFIER
46             | book_body paper_block
47             | book_body bookpart_block
48             | book_body score_block
49             | book_body composite_music
50             | book_body full_markup
51             | book_body full_markup_list
52             | book_body lilypond_header
53             | book_body error

54 bookpart_block:  "\bookpart" '{' bookpart_body '}'

55 bookpart_body:  /* empty */
56                 | BOOK_IDENTIFIER
57                 | bookpart_body paper_block
58                 | bookpart_body score_block
59                 | bookpart_body composite_music
60                 | bookpart_body full_markup
61                 | bookpart_body full_markup_list
62                 | bookpart_body lilypond_header
63                 | bookpart_body error

64 score_block:  "\score" '{' score_body '}'

65 score_body:  music
66             | SCORE_IDENTIFIER
67             | score_body lilypond_header
68             | score_body output_def
69             | score_body error

70 paper_block:  output_def

71 output_def:  output_def_body '}'

72 output_def_head:  "\paper"
73                 | "\midi"
74                 | "\layout"

75 output_def_head_with_mode_switch:  output_def_head

76 output_def_body:  output_def_head_with_mode_switch '{'
77                 | output_def_head_with_mode_switch

```

```

        '{'
        OUTPUT_DEF_IDENTIFIER
78         | output_def_body assignment
79         | output_def_body context_def_spec_block
80         | output_def_body error

81 tempo_event:  "\tempo" steno_duration '=' bare_unsigned
82             | "\tempo" string steno_duration '=' bare_unsigned
83             | "\tempo" full_markup steno_duration '=' bare_unsigned
84             | "\tempo" string
85             | "\tempo" full_markup

86 music_list:  /* empty */
87             | music_list music
88             | music_list embedded_scm
89             | music_list error

90 music:  simple_music
91        | composite_music

92 alternative_music:  /* empty */
93                    | "\alternative" '{' music_list '}'

94 repeated_music:  "\repeat"
                   simple_string
                   unsigned_number
                   music
                   alternative_music

95 sequential_music:  "\sequential" '{' music_list '}'
96                    | '{' music_list '}'

97 simultaneous_music:  "\simultaneous" '{' music_list '}'
98                      | "<<" music_list ">>"

99 simple_music:  event_chord
100              | MUSIC_IDENTIFIER
101              | music_property_def
102              | context_change

104 context_modification:  "\with" '{' context_mod_list '}'
105                       | "\with" CONTEXT_MOD_IDENTIFIER
106                       | CONTEXT_MOD_IDENTIFIER

107 optional_context_mod:  /* empty */
108                       | context_modification

109 context_mod_list:  /* empty */
110                  | context_mod_list context_mod
111                  | context_mod_list CONTEXT_MOD_IDENTIFIER

112 composite_music:  prefix_composite_music

```

```

113             | grouped_music_list

114 grouped_music_list: simultaneous_music
115             | sequential_music

116 function_scm_argument: embedded_scm
117             | simple_string

118 function_arglist_music_last: EXPECT_MUSIC function_arglist music

119 function_arglist_nonmusic_last: EXPECT_MARKUP
                                function_arglist
                                full_markup
120             | EXPECT_MARKUP
                                function_arglist
                                simple_string
121             | EXPECT_SCM
                                function_arglist
                                function_scm_argument

122 function_arglist_nonmusic: EXPECT_NO_MORE_ARGS
123             | EXPECT_MARKUP
                                function_arglist_nonmusic
                                full_markup
124             | EXPECT_MARKUP
                                function_arglist_nonmusic
                                simple_string
125             | EXPECT_SCM
                                function_arglist_nonmusic
                                function_scm_argument

126 function_arglist: EXPECT_NO_MORE_ARGS
127             | function_arglist_music_last
128             | function_arglist_nonmusic_last

129 generic_prefix_music_scm: MUSIC_FUNCTION function_arglist

130 optional_id: /* empty */
131             | '=' simple_string

132 prefix_composite_music: generic_prefix_music_scm
133             | "\context"
                                simple_string
                                optional_id
                                optional_context_mod
                                music
134             | "\new"
                                simple_string
                                optional_id
                                optional_context_mod
                                music
135             | "\times" fraction music

```

```

136          | repeated_music
137          | "\transpose"
              pitch_also_in_chords
              pitch_also_in_chords
              music
138          | mode_changing_head grouped_music_list
139          | mode_changing_head_with_context
              optional_context_mod
              grouped_music_list
140          | relative_music
141          | re_rhythmed_music

142 mode_changing_head:  "\notemode"
143                   | "\drummode"
144                   | "\figuremode"
145                   | "\chordmode"
146                   | "\lyricmode"

147 mode_changing_head_with_context:  "\drums"
148                                | "\figures"
149                                | "\chords"
150                                | "\lyrics"

151 relative_music:  "\relative" absolute_pitch music
152                | "\relative" composite_music

154 new_lyrics:  "\addlyrics" grouped_music_list

156 new_lyrics:  new_lyrics "\addlyrics" grouped_music_list

157 re_rhythmed_music:  grouped_music_list new_lyrics

159 re_rhythmed_music:  "\lyricsto" simple_string music

160 context_change:  "\change" STRING '=' STRING

161 property_path_revved:  embedded_scm
162                   | property_path_revved embedded_scm

163 property_path:  property_path_revved

164 property_operation:  STRING '=' scalar
165                   | "\unset" simple_string
166                   | "\override" simple_string property_path '=' scalar
167                   | "\revert" simple_string embedded_scm

168 context_def_mod:  "\consists"
169                 | "\remove"
170                 | "\accepts"
171                 | "\defaultchild"
172                 | "\denies"
173                 | "\alias"

```

```

174             | "\type"
175             | "\description"
176             | "\name"

177 context_mod:  property_operation
178             | context_def_mod STRING
179             | context_def_mod embedded_scm

180 context_prop_spec:  simple_string
181                 | simple_string '.' simple_string

182 simple_music_property_def:  "\override"
                             context_prop_spec
                             property_path
                             '='
                             scalar
183                 | "\revert" context_prop_spec embedded_scm
184                 | "\set" context_prop_spec '=' scalar
185                 | "\unset" context_prop_spec

186 music_property_def:  simple_music_property_def
187                 | "\once" simple_music_property_def

188 string:  STRING
189         | STRING_IDENTIFIER
190         | string '+' string

191 simple_string:  STRING
192               | LYRICS_STRING
193               | STRING_IDENTIFIER

194 scalar:  string
195         | LYRICS_STRING
196         | bare_number
197         | embedded_scm
198         | full_markup
199         | DIGIT

200 event_chord:  simple_chord_elements post_events
201             | CHORD_REPETITION optional_notemode_duration post_events
202             | MULTI_MEASURE_REST optional_notemode_duration post_events
203             | command_element
204             | note_chord_element

205 note_chord_element:  chord_body optional_notemode_duration post_events

206 chord_body:  "<" chord_body_elements ">"

207 chord_body_elements:  /* empty */
208                 | chord_body_elements chord_body_element

209 chord_body_element:  pitch

```

```

                                exclamations
                                questions
                                octave_check
                                post_events
210                            | DRUM_PITCH post_events
211                            | music_function_chord_body

212 music_function_identifier_musicless_prefix: MUSIC_FUNCTION

213 music_function_chord_body: music_function_identifier_musicless_prefix
                                EXPECT_MUSIC
                                function_arglist_nonmusic
                                chord_body_element
214                            | music_function_identifier_musicless_prefix
                                function_arglist_nonmusic

215 music_function_event: music_function_identifier_musicless_prefix
                                EXPECT_MUSIC
                                function_arglist_nonmusic
                                post_event
216                            | music_function_identifier_musicless_prefix
                                function_arglist_nonmusic

217 command_element: command_event
218                    | "\skip" duration_length
219                    | "["
220                    | "]"
221                    | "\"
222                    | '|'
223                    | "\partial" duration_length
224                    | "\time" fraction
225                    | "\mark" scalar

226 command_event: "~"
227                | "\mark" "\default"
228                | tempo_event
229                | "\key" "\default"
230                | "\key" NOTENAME_PITCH SCM_IDENTIFIER

231 post_events: /* empty */
232                | post_events post_event

233 post_event: direction_less_event
234             | '-' music_function_event
235             | "--"
236             | "=="
237             | script_dir direction_reqd_event
238             | script_dir direction_less_event
239             | string_number_event

240 string_number_event: E_UNSIGNED

```



```

282             | '_'

283 script_dir:  '_'
284             | '^'
285             | '-'

286 absolute_pitch:  steno_pitch

287 duration_length:  multiplied_duration

288 optional_notemode_duration:  /* empty */
289                             | multiplied_duration

290 steno_duration:  bare_unsigned dots
291                 | DURATION_IDENTIFIER dots

292 multiplied_duration:  steno_duration
293                     | multiplied_duration '*' bare_unsigned
294                     | multiplied_duration '*' FRACTION

295 fraction:  FRACTION
296           | UNSIGNED '/' UNSIGNED

297 dots:  /* empty */
298       | dots '.'

299 tremolo_type:  ':'
300             | ':' bare_unsigned

301 bass_number:  DIGIT
302             | UNSIGNED
303             | STRING
304             | full_markup

305 figured_bass_alteration:  '-'
306                         | '+'
307                         | '!'

308 bass_figure:  "_"
309             | bass_number
310             | bass_figure ']'
311             | bass_figure figured_bass_alteration
312             | bass_figure figured_bass_modification

313 figured_bass_modification:  "\+"
314                         | "\!"
315                         | '/'
316                         | "\"

317 br_bass_figure:  bass_figure
318               | '[' bass_figure

```

```

319 figure_list: /* empty */
320             | figure_list br_bass_figure

321 figure_spec:  FIGURE_OPEN figure_list FIGURE_CLOSE

322 optional_rest: /* empty */
323             | "\rest"

324 simple_element: pitch
                  exclamations
                  questions
                  octave_check
                  optional_notemode_duration
                  optional_rest
325             | DRUM_PITCH optional_notemode_duration
326             | RESTNAME optional_notemode_duration
327             | lyric_element optional_notemode_duration

328 simple_chord_elements: simple_element
329                       | new_chord
330                       | figure_spec optional_notemode_duration

331 lyric_element: lyric_markup
332             | LYRICS_STRING

333 new_chord: steno_tonic_pitch optional_notemode_duration
334          | steno_tonic_pitch
              optional_notemode_duration
              chord_separator
              chord_items

335 chord_items: /* empty */
336          | chord_items chord_item

337 chord_separator: ":"
338                | "^"
339                | "/" steno_tonic_pitch
340                | "/" steno_tonic_pitch

341 chord_item: chord_separator
342           | step_numbers
343           | CHORD_MODIFIER

344 step_numbers: step_number
345           | step_numbers '.' step_number

346 step_number: bare_unsigned
347           | bare_unsigned '+'
348           | bare_unsigned "-"

349 number_expression: number_expression '+' number_term
350                 | number_expression '-' number_term

```

```

351          | number_term

352 number_term:  number_factor
353             | number_factor '*' number_factor
354             | number_factor '/' number_factor

355 number_factor:  '-' number_factor
356               | bare_number

357 bare_number:  UNSIGNED
358             | REAL
359             | NUMBER_IDENTIFIER
360             | REAL NUMBER_IDENTIFIER
361             | UNSIGNED NUMBER_IDENTIFIER

362 bare_unsigned:  UNSIGNED
363               | DIGIT

364 unsigned_number:  bare_unsigned
365                 | NUMBER_IDENTIFIER

366 exclamations:  /* empty */
367               | exclamations '!'

368 questions:  /* empty */
369             | questions '?'

370 lyric_markup:  LYRIC_MARKUP_IDENTIFIER

372 lyric_markup:  LYRIC_MARKUP markup_top

374 full_markup_list:  "\markuptlines" markup_list

375 full_markup:  MARKUP_IDENTIFIER

377 full_markup:  "\markup" markup_top

378 markup_top:  markup_list
379             | markup_head_1_list simple_markup
380             | simple_markup

381 markup_list:  markup_composed_list
382             | markup_braced_list
383             | markup_command_list

384 markup_composed_list:  markup_head_1_list markup_braced_list

385 markup_braced_list:  '{' markup_braced_list_body '}'

386 markup_braced_list_body:  /* empty */
387                           | markup_braced_list_body markup
388                           | markup_braced_list_body markup_list

```

```

389 markup_command_list:  MARKUP_LIST_FUNCTION markup_command_list_arguments

390 markup_command_basic_arguments:  EXPECT_MARKUP_LIST
                                   markup_command_list_arguments
                                   markup_list
391                               | EXPECT_SCM
                                   markup_command_list_arguments
                                   embedded_scm
392                               | EXPECT_NO_MORE_ARGS

393 markup_command_list_arguments:  markup_command_basic_arguments
394                               | EXPECT_MARKUP
                                   markup_command_list_arguments
                                   markup

395 markup_head_1_item:  MARKUP_FUNCTION
                       EXPECT_MARKUP
                       markup_command_list_arguments

396 markup_head_1_list:  markup_head_1_item
397                     | markup_head_1_list markup_head_1_item

398 simple_markup:  STRING
399               | MARKUP_IDENTIFIER
400               | LYRIC_MARKUP_IDENTIFIER
401               | STRING_IDENTIFIER

403 simple_markup:  "\score" '{' score_body '}'
404               | MARKUP_FUNCTION markup_command_basic_arguments

405 markup:  markup_head_1_list simple_markup
406         | simple_markup

```

Terminals, with rules where they appear

```

"-" (319) 348
"--" (340) 235
"/" (320) 339
"/+" (316) 340
":" (318) 337
"<" (321) 206
"<<" (323) 98
">" (322) 206 280
">>" (324) 98
"\!" (329) 246 314
\" (325) 221 316
\" (331) 247
\" \" (328) 248

```

"\+" (334) 313
"\<" (333) 250
"\>" (326) 249
"\[" (330) 219
"\]" (332) 220
"\accepts" (261) 170
"\addlyrics" (259) 154 156
"\alias" (262) 173
"\alternative" (263) 93
"\book" (264) 43
"\bookpart" (265) 54
"\C[haracter]" (327)
"\change" (266) 160
"\chordmode" (267) 145
"\chords" (268) 149
"\consists" (269) 168
"\context" (270) 37 133
"\default" (271) 227 229
"\defaultchild" (272) 171
"\denies" (273) 172
"\description" (274) 175
"\drummode" (275) 143
"\drums" (276) 147
"\figuremode" (277) 144
"\figures" (278) 148
"\grobdescriptions" (279) 40
"\header" (280) 18
"\invalid" (281) 5
"\key" (282) 229 230
"\layout" (283) 74
"\lyricmode" (284) 146
"\lyrics" (285) 150
"\lyricsto" (286) 159
"\mark" (287) 225 227
"\markup" (288) 377
"\markuplines" (289) 374
"\midi" (290) 73
"\name" (291) 176
"\new" (315) 134
"\notemode" (292) 142
"\octave" (293)
"\once" (294) 187
"\override" (295) 166 182
"\paper" (296) 72
"\partial" (297) 223
"\relative" (298) 151 152
"\remove" (299) 169
"\repeat" (300) 94
"\rest" (301) 323
"\revert" (302) 167 183
"\score" (303) 64 403
"\sequential" (304) 95

```

"\set" (305) 184
"\simultaneous" (306) 97
"\skip" (307) 218
"\tempo" (308) 81 82 83 84 85
"\time" (314) 224
"\times" (309) 135
"\transpose" (310) 137
"\type" (311) 174
"\unset" (312) 165 185
"\with" (313) 104 105
"\~" (335) 226
"^" (317) 338
"_" (339) 308
"__" (336) 236
$end (0) 0
'!' (33) 307 367
''' (39) 260 261
'(' (40) 244
')' (41) 245
'*' (42) 293 294 353
'+' (43) 190 277 306 347 349
',' (44) 262 263
'-' (45) 234 278 285 305 350 355
'.' (46) 181 281 298 345
'/' (47) 296 315 354
':' (58) 299 300
'=' (61) 21 22 81 82 83 131 160 164 166 182 184 257 258 259
'?' (63) 369
'[' (91) 241 318
']' (93) 242 310
'^' (94) 276 284
'_' (95) 282 283
'{' (123) 18 37 43 54 64 76 77 93 95 96 97 104 385 403
'|' (124) 222 279
'}' (125) 18 37 43 54 64 71 93 95 96 97 104 385 403
'~' (126) 243
BOOK_IDENTIFIER (352) 45 56
CHORD_MODIFIER (354) 343
CHORD_REPETITION (355) 201
CHORDMODIFIER_PITCH (353)
CHORDMODIFIERS (341)
CONTEXT_DEF_IDENTIFIER (356) 39
CONTEXT_MOD_IDENTIFIER (357) 105 106 111
DIGIT (344) 35 199 275 301 363
DRUM_PITCH (358) 210 325
DURATION_IDENTIFIER (359) 291
E_UNSIGNED (345) 240
error (256) 4 53 63 69 80 89
EVENT_IDENTIFIER (360) 252
EXPECT_MARKUP (347) 119 120 123 124 394 395
EXPECT_MARKUP_LIST (350) 390
EXPECT_MUSIC (348) 118 213 215

```

EXPECT_NO_MORE_ARGS (351) 122 126 392
 EXPECT_SCM (349) 121 125 391
 FIGURE_CLOSE (337) 321
 FIGURE_OPEN (338) 321
 FRACTION (361) 294 295
 LYRIC_MARKUP (342) 372
 LYRIC_MARKUP_IDENTIFIER (363) 370 400
 LYRICS_STRING (362) 20 192 195 332
 MARKUP_FUNCTION (364) 395 404
 MARKUP_IDENTIFIER (366) 375 399
 MARKUP_LIST_FUNCTION (365) 389
 MULTI_MEASURE_REST (343) 202
 MUSIC_FUNCTION (367) 129 212
 MUSIC_IDENTIFIER (368) 100
 NOTENAME_PITCH (369) 230 264 265 266
 NUMBER_IDENTIFIER (370) 359 360 361 365
 OUTPUT_DEF_IDENTIFIER (371) 77
 PREC_BOT (260)
 PREC_TOP (258)
 REAL (372) 358 360
 RESTNAME (373) 326
 SCM_IDENTIFIER (374) 15 230
 SCM_TOKEN (375) 14
 SCORE_IDENTIFIER (376) 66
 STRING (377) 19 160 164 178 188 191 303 398
 STRING_IDENTIFIER (378) 189 193 401
 TONICNAME_PITCH (379) 267 268 269
 UNARY_MINUS (380)
 UNSIGNED (346) 296 302 357 361 362

Nonterminals, with rules where they appear

absolute_pitch (237)
 on left: 286, on right: 151
 alternative_music (172)
 on left: 92 93, on right: 94
 assignment (154)
 on left: 21 22 23, on right: 3 17 78
 assignment_id (153)
 on left: 19 20, on right: 21 22
 bare_number (265)
 on left: 357 358 359 360 361, on right: 196 356
 bass_number (245)
 on left: 301 302 303 304, on right: 309
 book_block (158)
 on left: 43, on right: 7 25
 bookpart_block (160)
 on left: 54, on right: 8 26 47
 br_bass_figure (249)
 on left: 317 318, on right: 320
 chord_body (213)

```

    on left: 206, on right: 205
chord_body_element (215)
    on left: 209 210 211, on right: 208 213
chord_body_elements (214)
    on left: 207 208, on right: 206 208
chord_item (259)
    on left: 341 342 343, on right: 336
chord_items (257)
    on left: 335 336, on right: 334 336
chord_separator (258)
    on left: 337 338 339 340, on right: 334 341
command_event (220)
    on left: 226 227 228 229 230, on right: 217
composite_music (181)
    on left: 112 113, on right: 10 49 59 91 152
context_change (199)
    on left: 160, on right: 102
context_def_spec_block (156)
    on left: 37, on right: 28 79
context_mod (204)
    on left: 177 178 179, on right: 41 110
context_mod_list (180)
    on left: 109 110 111, on right: 104 110 111
context_modification (177)
    on left: 104 105 106, on right: 36 42 108
context_prop_spec (205)
    on left: 180 181, on right: 182 183 184 185
direction_less_event (225)
    on left: 251 252 253, on right: 233 238
direction_reqd_event (226)
    on left: 254 255, on right: 237
dots (243)
    on left: 297 298, on right: 290 291 298
duration_length (238)
    on left: 287, on right: 218 223
event_chord (211)
    on left: 200 201 202 203 204, on right: 99
exclamations (268)
    on left: 366 367, on right: 209 324 367
figure_list (250)
    on left: 319 320, on right: 320 321
figure_spec (251)
    on left: 321, on right: 330
figured_bass_alteration (246)
    on left: 305 306 307, on right: 311
figured_bass_modification (248)
    on left: 313 314 315 316, on right: 312
fraction (242)
    on left: 295 296, on right: 135 224
full_markup_list (272)
    on left: 374, on right: 12 51 61
function_arglist_music_last (184)

```

```

    on left: 118, on right: 127
function_arglist_nonmusic_last (185)
    on left: 119 120 121, on right: 128
function_scm_argument (183)
    on left: 116 117, on right: 121 125
gen_text_def (234)
    on left: 273 274 275, on right: 254
generic_prefix_music_scm (188)
    on left: 129, on right: 132
lilypond (148)
    on left: 1 2 3 4 5, on right: 0 2 3 4 5
lilypond_header (152)
    on left: 18, on right: 6 52 62 67
lilypond_header_body (151)
    on left: 16 17, on right: 17 18
lyric_element (255)
    on left: 331 332, on right: 327
lyric_markup (270)
    on left: 370 372, on right: 331
markup (288)
    on left: 405 406, on right: 387 394
markup_braced_list (279)
    on left: 385, on right: 382 384
markup_braced_list_body (280)
    on left: 386 387 388, on right: 385 387 388
markup_command_list (281)
    on left: 389, on right: 383
markup_composed_list (278)
    on left: 384, on right: 381
markup_head_1_item (284)
    on left: 395, on right: 396 397
markup_head_1_list (285)
    on left: 396 397, on right: 379 384 397 405
markup_list (277)
    on left: 381 382 383, on right: 374 378 388 390
markup_top (276)
    on left: 378 379 380, on right: 372 377
mode_changing_head (191)
    on left: 142 143 144 145 146, on right: 138
multiplied_duration (241)
    on left: 292 293 294, on right: 287 289 293 294
music_function_chord_body (217)
    on left: 213 214, on right: 211
music_function_event (218)
    on left: 215 216, on right: 234
music_list (170)
    on left: 86 87 88 89, on right: 87 88 89 93 95 96 97 98
music_property_def (207)
    on left: 186 187, on right: 101
new_chord (256)
    on left: 333 334, on right: 329
new_lyrics (194)

```

```

    on left: 154 156, on right: 156 157
note_chord_element (212)
    on left: 205, on right: 204
number_expression (262)
    on left: 349 350 351, on right: 31 349 350
number_factor (264)
    on left: 355 356, on right: 352 353 354 355
number_term (263)
    on left: 352 353 354, on right: 349 350 351
octave_check (227)
    on left: 256 257 258 259, on right: 209 324
optional_context_mod (179)
    on left: 107 108, on right: 133 134 139
optional_id (189)
    on left: 130 131, on right: 133 134
optional_rest (252)
    on left: 322 323, on right: 324
output_def (165)
    on left: 71, on right: 13 27 68 70
output_def_body (168)
    on left: 76 77 78 79 80, on right: 71 78 79 80
output_def_head (166)
    on left: 72 73 74, on right: 75
output_def_head_with_mode_switch (167)
    on left: 75, on right: 76 77
paper_block (164)
    on left: 70, on right: 46 57
pitch (232)
    on left: 270, on right: 209 271 324
pitch_also_in_chords (233)
    on left: 271 272, on right: 137
post_events (221)
    on left: 231 232, on right: 200 201 202 205 209 210 232
property_operation (202)
    on left: 164 165 166 167, on right: 177
property_path (201)
    on left: 163, on right: 22 166 182
property_path_revved (200)
    on left: 161 162, on right: 162 163
questions (269)
    on left: 368 369, on right: 209 324 369
re_rhythmed_music (197)
    on left: 157 159, on right: 141
relative_music (193)
    on left: 151 152, on right: 140
repeated_music (173)
    on left: 94, on right: 136
score_block (162)
    on left: 64, on right: 9 24 48 58
score_body (163)
    on left: 65 66 67 68 69, on right: 64 67 68 69 403
script_dir (236)

```

```

    on left: 283 284 285, on right: 237 238
sequential_music (174)
    on left: 95 96, on right: 115
simple_chord_elements (254)
    on left: 328 329 330, on right: 200
simple_element (253)
    on left: 324 325 326 327, on right: 328
simple_music (176)
    on left: 99 100 101 102, on right: 90
simultaneous_music (175)
    on left: 97 98, on right: 114
steno_duration (240)
    on left: 290 291, on right: 81 82 83 292
steno_pitch (230)
    on left: 264 265 266, on right: 270 286
step_number (261)
    on left: 346 347 348, on right: 344 345
step_numbers (260)
    on left: 344 345, on right: 342 345
string (208)
    on left: 188 189 190, on right: 32 82 84 190 194 274
string_number_event (223)
    on left: 240, on right: 239
sub_quotes (229)
    on left: 262 263, on right: 258 263 266 269
sup_quotes (228)
    on left: 260 261, on right: 259 261 265 268
tempo_event (169)
    on left: 81 82 83 84 85, on right: 228
toplevel_expression (149)
    on left: 6 7 8 9 10 11 12 13, on right: 2
tremolo_type (244)
    on left: 299 300, on right: 253
unsigned_number (267)
    on left: 364 365, on right: 94

```

Appendix D GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

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 functional and useful 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. Secondly, 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, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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 fewer than five), unless they release you from this requirement.
- 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", Preserve 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. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements 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 to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

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, and that you preserve all their Warranty Disclaimers.

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 “Acknowledgements”, 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, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which 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 half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket 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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

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 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

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 lists all the LilyPond commands and keywords with links to those sections of the manual which describe or discuss their use. Each link is in two parts. The first part points to the exact location in the manual where the command or keyword appears; the second part points to the start of the section of the manual in which the command or keyword appears.

!	~
! 5	~ 282
,	-
' 1	- 200, 204
,	\
, 1	\! 93
-	\(..... 101
- 90	\) 101
.	\< 93
. 36	\> 93
/	\abs-fontsize 474
/ 283	\accent 90
/+ 283	\accepts 406, 407, 408
:	\addChordShape 252
: 119	\addInstrumentDefinition 156
<	\addlyrics 202
< 121	\addQuote 157
<...> 121	\aeolian 16
=	\afterGrace 84
= 9	\aikenHeads 32
>	\aikenHeadsMinor 32
> 121	\alias 406
?	\allowPageTurn 369
? 5	\alternative 110
[\AncientRemoveEmptyStaffContext 149
[..... 69	\arpeggio 105
]	\arpeggioArrowDown 105
] 69	\arpeggioArrowUp 105
	\arpeggioBracket 106
	\arpeggioNormal 105
	\arpeggioParenthesis 106
	\arpeggioParenthesisDashed 106
	\arrow-head 191
	\arrow-head 496
	\ascendens 312, 318
	\auctum 312, 318
	\augmentum 318
	\autoBeamOff 63
	\autoBeamOn 63
	\autochange 218
	\backslashed-digit 508
	\balloonGrobText 171
	\balloonLengthOff 171
	\balloonLengthOn 171
	\balloonText 171
	\bar 72, 77
	\barNumberCheck 81
	\beam 496
	\bendAfter 104
	\bold 184, 474
	\book 329, 330

<code>\bookpart</code>	330, 368	<code>\dynamicNeutral</code>	94
<code>\box</code>	189, 474	<code>\dynamicUp</code>	94
<code>\bracket</code>	97, 189, 497	<code>\easyHeadsOff</code>	30
<code>\break</code>	366	<code>\easyHeadsOn</code>	30
<code>\breathe</code>	102	<code>\epsfile</code>	191
<code>\breve</code>	35, 45	<code>\epsfile</code>	498
<code>\cadenzaOff</code>	55	<code>\espressivo</code>	90, 93
<code>\cadenzaOn</code>	55	<code>\expandFullBarRests</code>	49, 50
<code>\caesura</code>	310	<code>\eyeglasses</code>	508
<code>\caps</code>	474	<code>\f</code>	93
<code>\cavum</code>	312, 318	<code>\featherDurations</code>	72
<code>\center-align</code>	186, 482	<code>\fermata</code>	90
<code>\center-column</code>	188, 482	<code>\fermataMarkup</code>	49, 50
<code>\change</code>	217	<code>\ff</code>	93
<code>\char</code>	508	<code>\fff</code>	93
<code>\chordmode</code>	4, 13, 250	<code>\ffff</code>	93
<code>\chords</code>	285	<code>\fffff</code>	93
<code>\circle</code>	189, 497	<code>\fill-line</code>	188, 484
<code>\clef</code>	13	<code>\filled-box</code>	191
<code>\cm</code>	421	<code>\filled-box</code>	498
<code>\coda</code>	90	<code>\finalis</code>	310
<code>\column</code>	188, 483	<code>\finger</code>	165, 475
<code>\column-lines</code>	512	<code>\flageolet</code>	90
<code>\combine</code>	191	<code>\flat</code>	501
<code>\combine</code>	483	<code>\flexa</code>	318
<code>\compressFullBarRests</code>	49, 50	<code>\fontCaps</code>	475
<code>\concat</code>	483	<code>\fontsize</code>	184, 475
<code>\consists</code>	406	<code>\fp</code>	93
<code>\context</code>	400	<code>\fraction</code>	508
<code>\cr</code>	93	<code>\frenchChords</code>	287
<code>\crescHairpin</code>	94	<code>\fret-diagram</code>	242, 504
<code>\crescTextCresc</code>	94	<code>\fret-diagram-terse</code>	244, 505
<code>\cueDuring</code>	161	<code>\fret-diagram-verbose</code>	245, 505
<code>\customTabClef</code>	501	<code>\fromproperty</code>	508
<code>\decr</code>	93	<code>\general-align</code>	187, 485
<code>\defaultTimeSignature</code>	53	<code>\germanChords</code>	287
<code>\deminutum</code>	312, 318	<code>\glissando</code>	104
<code>\denies</code>	406, 407, 408	<code>\grace</code>	84
<code>\descendens</code>	312, 318	<code>\halfopen</code>	90
<code>\dimHairpin</code>	94	<code>\halign</code>	186, 486
<code>\dimTextDecr</code>	94	<code>\harmonic</code>	229
<code>\dimTextDecresc</code>	94	<code>\harmonicsOff</code>	229
<code>\dimTextDim</code>	94	<code>\harmonicsOn</code>	229
<code>\dir-column</code>	484	<code>\harp-pedal</code>	506
<code>\displayLilyMusic</code>	345	<code>\hbracket</code>	189, 498
<code>\divisioMaior</code>	310	<code>\hcenter-in</code>	487
<code>\divisioMaxima</code>	310	<code>\header</code>	330
<code>\divisioMinima</code>	310	<code>\hideKeySignature</code>	276
<code>\dorian</code>	16	<code>\hideNotes</code>	168
<code>\dotsDown</code>	36	<code>\hideSplitTiedTabNotes</code>	236
<code>\dotsNeutral</code>	36	<code>\hideStaffSwitch</code>	219
<code>\dotsUp</code>	36	<code>\hspace</code>	487
<code>\doubleflat</code>	501	<code>\huge</code>	164, 186, 475
<code>\doublesharp</code>	501	<code>\improvisationOff</code>	34
<code>\downbow</code>	90	<code>\improvisationOff</code>	60
<code>\downbow</code>	228	<code>\improvisationOn</code>	34
<code>\downmordent</code>	90	<code>\improvisationOn</code>	60
<code>\downprall</code>	90	<code>\in</code>	421
<code>\draw-circle</code>	191	<code>\inclinatum</code>	312, 318
<code>\draw-circle</code>	497	<code>\include</code>	339
<code>\draw-line</code>	191	<code>\instrumentSwitch</code>	156
<code>\draw-line</code>	497	<code>\ionian</code>	16
<code>\drummode</code>	136	<code>\italianChords</code>	287
<code>\dynamic</code>	97, 475	<code>\italic</code>	184, 476
<code>\dynamicDown</code>	94	<code>\justified-lines</code>	194

<code>\justified-lines</code>	513	<code>\normal-text</code>	478
<code>\justify</code>	188, 488	<code>\normalsize</code>	164, 186, 478
<code>\justify-field</code>	488	<code>\note</code>	502
<code>\justify-string</code>	489	<code>\note-by-number</code>	502
<code>\keepWithTag</code>	341	<code>\null</code>	186, 510
<code>\key</code>	16, 32	<code>\number</code>	478
<code>\killCues</code>	162	<code>\numericTimeSignature</code>	53
<code>\label</code>	337	<code>\octaveCheck</code>	9
<code>\laissezVibrer</code>	43	<code>\on-the-fly</code>	510
<code>\large</code>	164, 186, 476	<code>\once</code>	416
<code>\larger</code>	184, 186, 476	<code>\oneVoice</code>	123
<code>\layout</code>	330, 365	<code>\open</code>	90
<code>\left-align</code>	186, 489	<code>\open</code>	228
<code>\left-brace</code>	509	<code>\oriscus</code>	312, 318
<code>\left-column</code>	490	<code>\ottava</code>	18
<code>\lheel</code>	90	<code>\override</code>	414, 510
<code>\line</code>	490	<code>\override-lines</code>	513
<code>\linea</code>	312, 318	<code>\p</code>	93
<code>\lineprall</code>	90	<code>\pad-around</code>	190, 490
<code>\locrian</code>	16	<code>\pad-markup</code>	190, 491
<code>\longa</code>	35, 45	<code>\pad-to-box</code>	190, 491
<code>\longfermata</code>	90	<code>\pad-x</code>	190, 491
<code>\lookup</code>	509	<code>\page-ref</code>	337, 510
<code>\lower</code>	186, 490	<code>\pageBreak</code>	368
<code>\ltoe</code>	90	<code>\pageTurn</code>	369
<code>\lydian</code>	16	<code>\paper</code>	330, 358
<code>\lyricmode</code>	199, 202	<code>\parallelMusic</code>	133
<code>\lyricsto</code>	202	<code>\parenthesize</code>	170, 499
<code>\magnify</code>	184, 476	<code>\partcombine</code>	130
<code>\major</code>	16	<code>\partial</code>	55, 110, 111
<code>\makeClusters</code>	123	<code>\pes</code>	318
<code>\marcato</code>	90	<code>\phrasingSlurDashed</code>	101
<code>\mark</code>	81, 177	<code>\phrasingSlurDashPattern</code>	102
<code>\markalphabet</code>	509	<code>\phrasingSlurDotted</code>	101
<code>\markletter</code>	510	<code>\phrasingSlurDown</code>	101
<code>\markup</code>	177	<code>\phrasingSlurHalfDashed</code>	102
<code>\markup</code>	181	<code>\phrasingSlurHalfSolid</code>	102
<code>\markup</code>	182	<code>\phrasingSlurNeutral</code>	101
<code>\markuplines</code>	181, 194	<code>\phrasingSlurSolid</code>	101
<code>\markuplines</code>	195	<code>\phrasingSlurUp</code>	101
<code>\maxima</code>	35, 45	<code>\phrygian</code>	16
<code>\medium</code>	477	<code>\pitchedTrill</code>	109
<code>\melisma</code>	206	<code>\portato</code>	90
<code>\melismaEnd</code>	206	<code>\postscript</code>	191
<code>\mergeDifferentlyDottedOff</code>	126	<code>\postscript</code>	499
<code>\mergeDifferentlyDottedOn</code>	126	<code>\pp</code>	93
<code>\mergeDifferentlyHeadedOff</code>	126	<code>\ppp</code>	93
<code>\mergeDifferentlyHeadedOn</code>	126	<code>\pppp</code>	93
<code>\mf</code>	93	<code>\ppppp</code>	93
<code>\midi</code>	330	<code>\prall</code>	90
<code>\minor</code>	16	<code>\pralldown</code>	90
<code>\mixolydian</code>	16	<code>\prallmordent</code>	90
<code>\mm</code>	421	<code>\prallprall</code>	90
<code>\mordent</code>	90	<code>\prallup</code>	90
<code>\mp</code>	93	<code>\predefinedFretboardsOff</code>	258
<code>\musicglyph</code>	83, 501	<code>\predefinedFretboardsOn</code>	258
<code>\name</code>	406	<code>\property in \lyricmode</code>	199
<code>\natural</code>	501	<code>\pt</code>	421
<code>\new</code>	400	<code>\put-adjacent</code>	492
<code>\noBeam</code>	69	<code>\quilisma</code>	312, 318
<code>\noBreak</code>	366	<code>\quoteDuring</code>	157, 161
<code>\noPageBreak</code>	368	<code>\raise</code>	186, 492
<code>\noPageTurn</code>	369	<code>\relative</code>	2, 4, 13, 218
<code>\normal-size-sub</code>	477	<code>\RemoveEmptyRhythmicStaffContext</code>	149
<code>\normal-size-super</code>	477	<code>\RemoveEmptyStaffContext</code>	148, 149

<code>\removeWithTag</code>	341	<code>\staccato</code>	90
<code>\repeat</code>	110	<code>\startGroup</code>	174
<code>\repeat percent</code>	117	<code>\startStaff</code>	143, 145
<code>\repeat tremolo</code>	119	<code>\startTrillSpan</code>	108
<code>\repeatTie</code>	43, 112	<code>\stemDown</code>	170
<code>\rest</code>	45	<code>\stemNeutral</code>	170
<code>\reverseturn</code>	90	<code>\stemUp</code>	170
<code>\revert</code>	415	<code>\stencil</code>	511
<code>\rfz</code>	93	<code>\stopGroup</code>	174
<code>\rheel</code>	90	<code>\stopped</code>	90
<code>\right-align</code>	186, 492	<code>\stopStaff</code>	143, 145
<code>\right-brace</code>	511	<code>\stopStaff</code>	148
<code>\right-column</code>	492	<code>\stopTrillSpan</code>	108
<code>\rightHandFinger</code>	260	<code>\storePredefinedDiagram</code>	252
<code>\roman</code>	478	<code>\stropho</code>	312, 318
<code>\rotate</code>	493	<code>\strut</code>	511
<code>\rounded-box</code>	189, 500	<code>\sub</code>	185, 480
<code>\rtoe</code>	90	<code>\super</code>	185, 480
<code>\sacredHarpHeads</code>	32	<code>\sustainOff</code>	222
<code>\sacredHarpHeadsMinor</code>	32	<code>\sustainOn</code>	222
<code>\sans</code>	479	<code>\tabChordRepetition</code>	235
<code>\scaleDurations</code>	41, 57	<code>\tabFullNotation</code>	233
<code>\score</code>	328, 330, 502	<code>\table-of-contents</code>	338, 513
<code>\segno</code>	90	<code>\tag</code>	341
<code>\semiflat</code>	503	<code>\taor</code>	276
<code>\semiGermanChords</code>	287	<code>\teeny</code>	164, 186, 481
<code>\semisharp</code>	503	<code>\tempo</code>	151
<code>\sesquiflat</code>	504	<code>\tenuto</code>	90
<code>\sesquisharp</code>	504	<code>\text</code>	481
<code>\set</code>	64, 412	<code>\textLengthOff</code>	50, 176
<code>\sf</code>	93	<code>\textLengthOn</code>	50, 176
<code>\sff</code>	93	<code>\textSpannerDown</code>	177
<code>\sfz</code>	93	<code>\textSpannerNeutral</code>	177
<code>\sharp</code>	504	<code>\textSpannerUp</code>	177
<code>\shiftOff</code>	126	<code>\thumb</code>	90, 165
<code>\shiftOn</code>	126	<code>\tied-lyric</code>	504
<code>\shiftOnn</code>	126	<code>\tieDashed</code>	43
<code>\shiftOnnn</code>	126	<code>\tieDotted</code>	43
<code>\shortfermata</code>	90	<code>\tieDown</code>	43
<code>\showKeySignature</code>	276	<code>\tieNeutral</code>	43
<code>\showStaffSwitch</code>	219	<code>\tieSolid</code>	43
<code>\signumcongruentiae</code>	90	<code>\tieUp</code>	43
<code>\simple</code>	479	<code>\time</code>	52, 64
<code>\skip</code>	47	<code>\times</code>	37, 57
<code>\slashed-digit</code>	511	<code>\tiny</code>	164, 186, 481
<code>\slurDashed</code>	99	<code>\tocItem</code>	338
<code>\slurDashPattern</code>	99	<code>\translate</code>	187, 493
<code>\slurDotted</code>	99	<code>\translate-scaled</code>	187, 493
<code>\slurDown</code>	99	<code>\transparent</code>	511
<code>\slurHalfDashed</code>	99	<code>\transpose</code>	4, 10, 13
<code>\slurHalfSolid</code>	99	<code>\transposedCueDuring</code>	162
<code>\slurNeutral</code>	99	<code>\transposition</code>	19, 157
<code>\slurSolid</code>	99	<code>\treCorde</code>	222
<code>\slurUp</code>	100	<code>\triangle</code>	191
<code>\small</code>	164, 186, 479	<code>\triangle</code>	500
<code>\smallCaps</code>	479	<code>\trill</code>	90, 108
<code>\smaller</code>	184, 186, 480	<code>\tupletDown</code>	37
<code>\snappizzicato</code>	90	<code>\tupletNeutral</code>	37
<code>\sostenutoOff</code>	222	<code>\tupletUp</code>	37
<code>\sostenutoOn</code>	222	<code>\turn</code>	90
<code>\southernHarmonyHeads</code>	32	<code>\tweak</code>	416
<code>\southernHarmonyHeadsMinor</code>	32	<code>\type</code>	406
<code>\sp</code>	93	<code>\typewriter</code>	481
<code>\spp</code>	93	<code>\unaCorda</code>	222
<code>\staccatissimo</code>	90	<code>\underline</code>	184, 482

<code>\unfoldRepeats</code>	351
<code>\unHideNotes</code>	168
<code>\unset</code>	413
<code>\upbow</code>	90
<code>\upbow</code>	228
<code>\upmordent</code>	90
<code>\upprall</code>	90
<code>\upright</code>	482
<code>\varcoda</code>	90
<code>\vcenter</code>	494
<code>\verbatim-file</code>	511
<code>\verylongfermata</code>	90
<code>\virga</code>	312, 318
<code>\virgula</code>	310
<code>\voiceFourStyle</code>	126
<code>\voiceNeutralStyle</code>	126
<code>\voiceOne</code>	123
<code>\voiceOne ... \voiceFour</code>	123
<code>\voiceOneStyle</code>	126
<code>\voiceThreeStyle</code>	126
<code>\voiceTwoStyle</code>	126
<code>\vspace</code>	494
<code>\whiteout</code>	512
<code>\with</code>	404
<code>\with-color</code>	168, 512
<code>\with-dimensions</code>	512
<code>\with-url</code>	500
<code>\woodwind-diagram</code>	507
<code>\wordwrap</code>	188, 495
<code>\wordwrap-field</code>	494
<code>\wordwrap-internal</code>	513
<code>\wordwrap-lines</code>	194
<code>\wordwrap-lines</code>	513
<code>\wordwrap-string</code>	495
<code>\wordwrap-string-internal</code>	513

|

.....	81
-------	----

~

~	42
---------	----

A

<code>accepts</code>	406
<code>add ChordShape</code>	252
<code>addInstrumentDefinition</code>	156
<code>addQuote</code>	157
<code>aeolian</code>	16
<code>after-title-spacing</code>	358
<code>afterGrace</code>	84
<code>aikenHeads</code>	32
<code>aikenHeadsMinor</code>	32
<code>alias</code>	406
<code>alignAboveContext</code>	408
<code>alignBelowContext</code>	408
<code>AncientRemoveEmptyStaffContext</code>	149
<code>annotate-spacing</code>	394
<code>arpeggio</code>	105
<code>arpeggioArrowDown</code>	105
<code>arpeggioArrowUp</code>	105
<code>arpeggioBracket</code>	106

<code>arpeggioNormal</code>	105
<code>arpeggioParenthesis</code>	106
<code>arpeggioParenthesisDashed</code>	106
<code>arranger</code>	332
<code>arrow-head</code>	191
<code>aug</code>	280
<code>auto-first-page-number</code>	362
<code>autoBeaming</code>	64
<code>autoBeamOff</code>	63
<code>autoBeamOn</code>	63
<code>autochange</code>	218

B

<code>Balloon_engraver</code>	171
<code>balloonGrobText</code>	171
<code>balloonLengthOff</code>	171
<code>balloonLengthOn</code>	171
<code>balloonText</code>	171
<code>banjo-c-tuning</code>	264
<code>banjo-modal-tuning</code>	264
<code>banjo-open-d-tuning</code>	264
<code>banjo-open-dm-tuning</code>	264
<code>bar</code>	72, 77
<code>barCheckSynchronize</code>	81
<code>BarNumber</code>	77
<code>barNumberCheck</code>	81
<code>barNumberVisibility</code>	77
<code>bartype</code>	77
<code>base-shortest-duration</code>	384
<code>beamSettings</code>	64
<code>beatLength</code>	64
<code>before-title-spacing</code>	358
<code>bendAfter</code>	104
<code>between-scores-system-spacing</code>	358
<code>between-system-spacing</code>	358
<code>between-title-spacing</code>	358
<code>binding-offset</code>	360
<code>blank-last-page-force</code>	362
<code>blank-page-force</code>	362
<code>bold</code>	184
<code>bookTitleMarkup</code>	335
<code>bottom-margin</code>	358
<code>bottom-system-spacing</code>	359
<code>box</code>	189
<code>bracket</code>	97, 189, 222
<code>breakable</code>	63
<code>breakbefore</code>	332
<code>breathe</code>	102
<code>breve</code>	35, 45

C

<code>cadenzaOff</code>	55
<code>cadenzaOn</code>	55
<code>center-align</code>	186
<code>center-column</code>	188
<code>change</code>	217
<code>check-consistency</code>	362
<code>chordChanges</code>	285
<code>chordmode</code>	4, 13, 250
<code>chordNameExceptions</code>	287
<code>chordNameLowercaseMinor</code>	287
<code>ChordNames</code>	250

chordNameSeparator	287
chordNoteNamer	287
chordPrefixSpacer	287
chordRootNamer	287
circle	189
clef	13
color	168
column	188
combine	191
common-shortest-duration	384
Completion_heads_engraver	59
composer	332
compressFullBarRests	49, 50
consists	406
controlpitch	9
copyright	332
cr	93
crescHairpin	94
crescTextCresc	94
cross	29
cross-staff	220
cueDuring	161
currentBarNumber	77, 89

D

decr	93
dedication	332
default	20, 22
defaultBarType	77
defaultTimeSignature	53
denies	406
dim	280
dimHairpin	94
dimTextDecr	94
dimTextDecresc	94
dimTextDim	94
dodecaphonic	25
dorian	16
dotsDown	36
dotsNeutral	36
dotsUp	36
draw-circle	191
draw-line	191
drummode	136
DrumStaff	136
dynamic	97
dynamicDown	94
DynamicLineSpanner	94
dynamicNeutral	94
dynamicUp	94

E

easyHeadsOff	30
easyHeadsOn	30
epsfile	191
espressivo	93
evenFooterMarkup	336
evenHeaderMarkup	335
expandFullBarRests	49, 50

F

f	93
featherDurations	72
fermataMarkup	49, 50
ff	93
fff	93
ffff	93
fffff	93
fill-line	188
filled-box	191
finger	165
first-page-number	362
flag-style	220
followVoice	219
font-interface	165, 195
font-size	164, 165
fontSize	184
fontSize	164
forget	26
four-string-banjo	264
fp	93
fret-diagram	242
fret-diagram-interface	247
fret-diagram-terse	244
fret-diagram-verbose	245
FretBoards	249

G

general-align	187
glissando	104
grace	84
GregorianTranscriptionStaff	136
Grid_line_span_engraver	172
Grid_point_engraver	172
gridInterval	172
grow-direction	72

H

halign	186
hbracket	189
hideKeySignature	276
hideNotes	168
hideStaffSwitch	219
horizontal-shift	360
Horizontal_bracket_engraver	174
huge	164, 186

I

improvisationOff	34
improvisationOff	60
improvisationOn	34
improvisationOn	60
indent	155, 360, 387
inner-margin	360
instrument	332
instrumentSwitch	156
ionian	16
italic	184

J

justified-lines	194
justify	188

K

key	16, 32
killCues	162

L

laissezVibrer	43
large	164, 186
larger	184, 186
layout file	365
left-align	186
left-margin	360
length	220
line-width	360, 387
locrian	16
longa	35, 45
lower	186
ly:minimal-breaking	370
ly:optimal-breaking	368
ly:page-turn-breaking	369
lydian	16

M

m	280
magnify	184
magstep	164, 421
maj	280
major	16
major seven symbols	287
majorSevenSymbol	287
make-dynamic-script	97
make-pango-font-tree	197
makeClusters	123
mark	81, 177
markup	177
markup	181
markup	182
markuplines	181, 194
markuplines	195
maxima	35, 45
measureLength	64, 89
measurePosition	55, 89
MensuralStaff	136
mergeDifferentlyDottedOff	126
mergeDifferentlyDottedOn	126
mergeDifferentlyHeadedOff	126
mergeDifferentlyHeadedOn	126
meter	332
mf	93
minimumFret	234
minimumPageTurnLength	369
minimumRepeatLengthForPageTurn	369
minor	16
mixed	222
mixolydian	16
modern	22
modern-cautionary	23
modern-voice	23

modern-voice-cautionary	23
mp	93
MultiMeasureRestText	49
musicglyph	83

N

name	406
neo-modern	24
neo-modern-cautionary	24
neo-modern-voice	24
neo-modern-voice-cautionary	25
no-reset	25
noBeam	69
normalsize	164, 186
Note_heads_engraver	59
null	186
numericTimeSignature	53

O

octaveCheck	9
oddFooterMarkup	335
oddHeaderMarkup	335
oneVoice	123
opus	332
ottava	18
outer-margin	360
outside-staff-horizontal-padding	382
outside-staff-padding	382
outside-staff-priority	382

P

p	93
pad-around	190
pad-markup	190
pad-to-box	190
pad-x	190
page-breaking-between-system-spacing	362
page-count	362
page-limit-inter-system-space	362
page-limit-inter-system-space-factor	362
page-spacing-weight	362
paper-height	359
paper-width	360
parallelMusic	133
parenthesize	170
partcombine	130
partial	55
pedalSustainStyle	222
percent	117
phrasingSlurDashed	101
phrasingSlurDashPattern	102
phrasingSlurDotted	101
phrasingSlurDown	101
phrasingSlurHalfDashed	102
phrasingSlurHalfSolid	102
phrasingSlurNeutral	101
phrasingSlurSolid	101
phrasingSlurUp	101
phrygian	16
piano	23
piano-cautionary	24

PianoStaff	216, 218
piece	332
pipeSymbol	81
Pitch_squash_engraver	60
pitchedTrill	109
poet	332
postscript	191
pp	93
ppp	93
pppp	93
ppppp	93
predefinedFretboardsOff	258
predefinedFretboardsOn	258
print-all-headers	335, 362
print-first-page-number	363
print-page-number	363

Q

quotedCueEventTypes	159
quotedEventTypes	159
quoteDuring	157, 161

R

r	45
R	48
ragged-bottom	363
ragged-last	363, 387
ragged-last-bottom	363
ragged-right	363, 387
raise	186
relative	2, 4, 13, 218
RemoveEmptyRhythmicStaffContext	149
RemoveEmptyStaffContext	148, 149
repeatCommands	114
repeatTie	43
rest	45
revertBeamSettings	66
rfz	93
rgb-color	169
RhythmicStaff	136
right-align	186
right-margin	360
rightHandFinger	260
rounded-box	189

S

s	47
sacredHarpHeads	32
sacredHarpHeadsMinor	32
scaleDurations	41, 57
scoreTitleMarkup	335
set	64
set-accidental-style	20
set-octavation	18
sf	93
sff	93
sfz	93
shiftOff	126
shiftOn	126
shiftOnn	126
shiftOnnn	126

short-indent	155, 361
show-available-fonts	197
showFirstLength	346
showKeySignature	276
showLastLength	346
showStaffSwitch	219
skip	47
skipTypesetting	346
slurDashed	99
slurDashPattern	99
slurDotted	99
slurDown	99
slurHalfDashed	99
slurHalfSolid	99
slurNeutral	99
slurSolid	99
slurUp	100
small	164, 186
smaller	184, 186
sostenutoOff	222
sostenutoOn	222
southernHarmonyHeads	32
southernHarmonyHeadsMinor	32
sp	93
spacing	384
spp	93
staff-padding	217
Staff_midiInstrument	348
Staff_symbol_engraver	148
start-repeat	114
startGroup	174
startStaff	143, 145
startTrillSpan	108
Stem	220
stem-spacing-correction	384
stemDown	170
stemLeftBeamCount	70
stemNeutral	170
stemRightBeamCount	70
stemUp	170
stopGroup	174
stopStaff	143, 145
stopStaff	148
stopTrillSpan	108
storePredefinedDiagram	252
StringTunings	249
StringTunings	239
sub	185
subsubtitle	332
subtitle	332
suggestAccidentals	306
super	185
sus	283
sustainOff	222
sustainOn	222
system-count	364
system-separator-markup	363
systems-per-page	364

T

TabStaff	136, 233
TabVoice	233
tagline	333

taor.....	276
teaching.....	25
teeny.....	164, 186
tempo.....	151
text.....	222
textLengthOff.....	176
textLengthOn.....	50, 176
textLenthOff.....	50
textSpannerDown.....	177
textSpannerNeutral.....	177
textSpannerUp.....	177
thumb.....	165
tieDashed.....	43
tieDotted.....	43
tieDown.....	43
tieNeutral.....	43
tieSolid.....	43
tieUp.....	43
time.....	52, 64
times.....	37, 57
timeSignatureFraction.....	57
tiny.....	164, 186
title.....	332
top-margin.....	359
top-system-spacing.....	359
top-title-spacing.....	359
translate.....	187
translate-scaled.....	187
transpose.....	4, 10, 13
transposedCueDuring.....	162
transposition.....	19, 157
treCorde.....	222
tremolo.....	119
tremoloFlags.....	119
triangle.....	191

trill.....	108
tupletDown.....	37
tupletNeutral.....	37
TupletNumber.....	38
tupletNumberFormatFunction.....	38
tupletSpannerDuration.....	38
tupletUp.....	37
two-sided.....	364
type.....	406

U

unaCorda.....	222
underline.....	184
unfold.....	116
unHideNotes.....	168

V

VaticanaStaff.....	136
voice.....	20, 22
Voice.....	123
voiceOne.....	123

W

whichBar.....	77
with-color.....	168
wordwrap.....	188
wordwrap-lines.....	194

X

x11-color.....	168, 169
----------------	----------

Appendix F LilyPond index

In addition to all the LilyPond commands and keywords, this index lists musical terms and words which relate to each of them, with links to those sections of the manual which describe or discuss that topic. Each link is in two parts. The first part points to the exact location in the manual where the topic appears; the second part points to the start of the section of the manual where that topic is discussed.

!		^	
!	5	^	282
,		-	
'	1	-	200, 204
,		\	
,	1	\!	93
-		\(.....	101
-	90	\)	101
.		\<	93
.		\>	93
.		\abs-fontsize	474
/		\accent	90
/	283	\accepts	406, 407, 408
/+	283	\addChordShape	252
:		\addInstrumentDefinition	156
:	119	\addlyrics	201
<		\addlyrics	202
<	121	\addQuote	157
<...>	121	\aeolian	16
=		\afterGrace	84
=	9	\aikenHeads	32
>		\aikenHeadsMinor	32
>	121	\alias	406
?		\allowPageTurn	369
?	5	\alternative	110
[\AncientRemoveEmptyStaffContext	149
[.....	69	\arpeggio	105
]		\arpeggioArrowDown	105
]	69	\arpeggioArrowUp	105
		\arpeggioBracket	106
		\arpeggioNormal	105
		\arpeggioParenthesis	106
		\arpeggioParenthesisDashed	106
		\arrow-head	191
		\arrow-head	496
		\ascendens	312, 318
		\auctum	312, 318
		\augmentum	318
		\autoBeamOff	63
		\autoBeamOn	63
		\autochange	218
		\backslashed-digit	508
		\balloonGrobText	171
		\balloonLengthOff	171
		\balloonLengthOn	171
		\balloonText	171
		\bar	72, 77
		\barNumberCheck	81
		\beam	496
		\bendAfter	104

<code>\bold</code>	184, 474	<code>\dynamic</code>	97, 475
<code>\book</code>	329, 330	<code>\dynamicDown</code>	94
<code>\bookpart</code>	330, 368	<code>\dynamicNeutral</code>	94
<code>\box</code>	189, 474	<code>\dynamicUp</code>	94
<code>\bracket</code>	97, 189, 497	<code>\easyHeadsOff</code>	30
<code>\break</code>	366	<code>\easyHeadsOn</code>	30
<code>\breathe</code>	102	<code>\epsfile</code>	191
<code>\breve</code>	35, 45	<code>\epsfile</code>	498
<code>\cadenzaOff</code>	55	<code>\espressivo</code>	90, 93
<code>\cadenzaOn</code>	55	<code>\expandFullBarRests</code>	49, 50
<code>\caesura</code>	310	<code>\eyeglasses</code>	508
<code>\caps</code>	474	<code>\f</code>	93
<code>\cavum</code>	312, 318	<code>\featherDurations</code>	72
<code>\center-align</code>	186, 482	<code>\fermata</code>	90
<code>\center-column</code>	188, 482	<code>\fermataMarkup</code>	49, 50
<code>\change</code>	217	<code>\ff</code>	93
<code>\char</code>	508	<code>\fff</code>	93
<code>\chordmode</code>	4, 13, 250	<code>\ffff</code>	93
<code>\chords</code>	285	<code>\fffff</code>	93
<code>\circle</code>	189, 497	<code>\fill-line</code>	188, 484
<code>\clef</code>	13	<code>\filled-box</code>	191
<code>\cm</code>	421	<code>\filled-box</code>	498
<code>\coda</code>	90	<code>\finalis</code>	310
<code>\column</code>	188, 483	<code>\finger</code>	165, 475
<code>\column-lines</code>	512	<code>\flageolet</code>	90
<code>\combine</code>	191	<code>\flat</code>	501
<code>\combine</code>	483	<code>\flexa</code>	318
<code>\compressFullBarRests</code>	49, 50	<code>\fontCaps</code>	475
<code>\concat</code>	483	<code>\fontsize</code>	184, 475
<code>\consists</code>	406	<code>\fp</code>	93
<code>\context</code>	400	<code>\fraction</code>	508
<code>\cr</code>	93	<code>\frenchChords</code>	287
<code>\crescHairpin</code>	94	<code>\fret-diagram</code>	242, 504
<code>\crescTextCresc</code>	94	<code>\fret-diagram-terse</code>	244, 505
<code>\cueDuring</code>	161	<code>\fret-diagram-verbose</code>	245, 505
<code>\customTabClef</code>	501	<code>\fromproperty</code>	508
<code>\decr</code>	93	<code>\general-align</code>	187, 485
<code>\defaultTimeSignature</code>	53	<code>\germanChords</code>	287
<code>\deminutum</code>	312, 318	<code>\glissando</code>	104
<code>\denies</code>	406, 407, 408	<code>\grace</code>	84
<code>\descendens</code>	312, 318	<code>\halfopen</code>	90
<code>\dimHairpin</code>	94	<code>\halign</code>	186, 486
<code>\dimTextDecr</code>	94	<code>\harmonic</code>	229
<code>\dimTextDecresc</code>	94	<code>\harmonicsOff</code>	229
<code>\dimTextDim</code>	94	<code>\harmonicsOn</code>	229
<code>\dir-column</code>	484	<code>\harp-pedal</code>	506
<code>\displayLilyMusic</code>	345	<code>\hbracket</code>	189, 498
<code>\divisioMaior</code>	310	<code>\hcenter-in</code>	487
<code>\divisioMaxima</code>	310	<code>\header</code>	330
<code>\divisioMinima</code>	310	<code>\hideKeySignature</code>	276
<code>\dorian</code>	16	<code>\hideNotes</code>	168
<code>\dotsDown</code>	36	<code>\hideSplitTiedTabNotes</code>	236
<code>\dotsNeutral</code>	36	<code>\hideStaffSwitch</code>	219
<code>\dotsUp</code>	36	<code>\hspace</code>	487
<code>\doubleflat</code>	501	<code>\huge</code>	164, 186, 475
<code>\doublesharp</code>	501	<code>\improvisationOff</code>	34
<code>\downbow</code>	90	<code>\improvisationOff</code>	60
<code>\downbow</code>	228	<code>\improvisationOn</code>	34
<code>\downmordent</code>	90	<code>\improvisationOn</code>	60
<code>\downprall</code>	90	<code>\in</code>	421
<code>\draw-circle</code>	191	<code>\inclinatum</code>	312, 318
<code>\draw-circle</code>	497	<code>\include</code>	339
<code>\draw-line</code>	191	<code>\instrumentSwitch</code>	156
<code>\draw-line</code>	497	<code>\ionian</code>	16
<code>\drummode</code>	136	<code>\italianChords</code>	287

<code>\italic</code>	184, 476	<code>\normal-size-sub</code>	477
<code>\justified-lines</code>	194	<code>\normal-size-super</code>	477
<code>\justified-lines</code>	513	<code>\normal-text</code>	478
<code>\justify</code>	188, 488	<code>\normalsize</code>	164, 186, 478
<code>\justify-field</code>	488	<code>\note</code>	502
<code>\justify-string</code>	489	<code>\note-by-number</code>	502
<code>\keepWithTag</code>	341	<code>\null</code>	186, 510
<code>\key</code>	16, 32	<code>\number</code>	478
<code>\killCues</code>	162	<code>\numericTimeSignature</code>	53
<code>\label</code>	337	<code>\octaveCheck</code>	9
<code>\laissezVibrer</code>	43	<code>\on-the-fly</code>	510
<code>\large</code>	164, 186, 476	<code>\once</code>	414
<code>\larger</code>	184, 186, 476	<code>\once</code>	416
<code>\layout</code>	330, 365	<code>\oneVoice</code>	123
<code>\left-align</code>	186, 489	<code>\open</code>	90
<code>\left-brace</code>	509	<code>\open</code>	228
<code>\left-column</code>	490	<code>\oriscus</code>	312, 318
<code>\lheel</code>	90	<code>\ottava</code>	18
<code>\line</code>	490	<code>\override</code>	414, 510
<code>\linea</code>	312, 318	<code>\override-lines</code>	513
<code>\lineprall</code>	90	<code>\p</code>	93
<code>\locrian</code>	16	<code>\pad-around</code>	190, 490
<code>\longa</code>	35, 45	<code>\pad-markup</code>	190, 491
<code>\longfermata</code>	90	<code>\pad-to-box</code>	190, 491
<code>\lookup</code>	509	<code>\pad-x</code>	190, 491
<code>\lower</code>	186, 490	<code>\page-ref</code>	337, 510
<code>\ltoe</code>	90	<code>\pageBreak</code>	368
<code>\lydian</code>	16	<code>\pageTurn</code>	369
<code>\lyricmode</code>	199, 202	<code>\paper</code>	330
<code>\lyricsto</code>	202	<code>\paper</code>	336
<code>\magnify</code>	184, 476	<code>\paper</code>	358
<code>\major</code>	16	<code>\parallelMusic</code>	133
<code>\makeClusters</code>	123	<code>\parenthesize</code>	170, 499
<code>\marcato</code>	90	<code>\partcombine</code>	130
<code>\mark</code>	81, 177	<code>\partial</code>	55, 110, 111
<code>\markalphabet</code>	509	<code>\pes</code>	318
<code>\markletter</code>	510	<code>\phrasingSlurDashed</code>	101
<code>\markup</code>	177	<code>\phrasingSlurDashPattern</code>	102
<code>\markup</code>	181	<code>\phrasingSlurDotted</code>	101
<code>\markup</code>	182	<code>\phrasingSlurDown</code>	101
<code>\markuplines</code>	181, 194	<code>\phrasingSlurHalfDashed</code>	102
<code>\markuplines</code>	195	<code>\phrasingSlurHalfSolid</code>	102
<code>\maxima</code>	35, 45	<code>\phrasingSlurNeutral</code>	101
<code>\medium</code>	477	<code>\phrasingSlurSolid</code>	101
<code>\melisma</code>	206	<code>\phrasingSlurUp</code>	101
<code>\melismaEnd</code>	206	<code>\phrygian</code>	16
<code>\mergeDifferentlyDottedOff</code>	126	<code>\pitchedTrill</code>	109
<code>\mergeDifferentlyDottedOn</code>	126	<code>\portato</code>	90
<code>\mergeDifferentlyHeadedOff</code>	126	<code>\postscript</code>	191
<code>\mergeDifferentlyHeadedOn</code>	126	<code>\postscript</code>	499
<code>\mf</code>	93	<code>\pp</code>	93
<code>\midi</code>	330	<code>\ppp</code>	93
<code>\minor</code>	16	<code>\pppp</code>	93
<code>\mixolydian</code>	16	<code>\ppppp</code>	93
<code>\mm</code>	421	<code>\prall</code>	90
<code>\mordent</code>	90	<code>\pralldown</code>	90
<code>\mp</code>	93	<code>\prallmordent</code>	90
<code>\musicglyph</code>	83, 501	<code>\prallprall</code>	90
<code>\name</code>	406	<code>\prallup</code>	90
<code>\natural</code>	501	<code>\predefinedFretboardsOff</code>	258
<code>\new</code>	400	<code>\predefinedFretboardsOn</code>	258
<code>\noBeam</code>	69	<code>\property in \lyricmode</code>	199
<code>\noBreak</code>	366	<code>\pt</code>	421
<code>\noPageBreak</code>	368	<code>\put-adjacent</code>	492
<code>\noPageTurn</code>	369	<code>\quilisma</code>	312, 318

<code>\quoteDuring</code>	157, 161	<code>\southernHarmonyHeads</code>	32
<code>\raise</code>	186, 492	<code>\southernHarmonyHeadsMinor</code>	32
<code>\relative</code>	2, 4, 13, 218	<code>\sp</code>	93
<code>\RemoveEmptyRhythmicStaffContext</code>	149	<code>\spp</code>	93
<code>\RemoveEmptyStaffContext</code>	148, 149	<code>\staccatissimo</code>	90
<code>\removeWithTag</code>	341	<code>\staccato</code>	90
<code>\repeat</code>	110	<code>\startGroup</code>	174
<code>\repeat percent</code>	117	<code>\startStaff</code>	143, 145
<code>\repeat tremolo</code>	119	<code>\startTrillSpan</code>	108
<code>\repeatTie</code>	43, 112	<code>\stemDown</code>	170
<code>\rest</code>	45	<code>\stemNeutral</code>	170
<code>\reverseturn</code>	90	<code>\stemUp</code>	170
<code>\revert</code>	415	<code>\stencil</code>	511
<code>\rfz</code>	93	<code>\stopGroup</code>	174
<code>\rheel</code>	90	<code>\stopped</code>	90
<code>\right-align</code>	186, 492	<code>\stopStaff</code>	143, 145
<code>\right-brace</code>	511	<code>\stopStaff</code>	148
<code>\right-column</code>	492	<code>\stopTrillSpan</code>	108
<code>\rightHandFinger</code>	260	<code>\storePredefinedDiagram</code>	252
<code>\roman</code>	478	<code>\strophia</code>	312, 318
<code>\rotate</code>	493	<code>\strut</code>	511
<code>\rounded-box</code>	189, 500	<code>\sub</code>	185, 480
<code>\rtoe</code>	90	<code>\super</code>	185, 480
<code>\sacredHarpHeads</code>	32	<code>\sustainOff</code>	222
<code>\sacredHarpHeadsMinor</code>	32	<code>\sustainOn</code>	222
<code>\sans</code>	479	<code>\tabChordRepetition</code>	235
<code>\scaleDurations</code>	41, 57	<code>\tabFullNotation</code>	233
<code>\score</code>	328, 330, 502	<code>\table-of-contents</code>	338, 513
<code>\segno</code>	90	<code>\tag</code>	341
<code>\semiflat</code>	503	<code>\taor</code>	276
<code>\semiGermanChords</code>	287	<code>\teeny</code>	164, 186, 481
<code>\semisharp</code>	503	<code>\tempo</code>	151
<code>\sesquiflat</code>	504	<code>\tenuto</code>	90
<code>\sesquisharp</code>	504	<code>\text</code>	481
<code>\set</code>	64, 412	<code>\textLengthOff</code>	50, 176
<code>\sf</code>	93	<code>\textLengthOn</code>	50, 176
<code>\sff</code>	93	<code>\textSpannerDown</code>	177
<code>\sfz</code>	93	<code>\textSpannerNeutral</code>	177
<code>\sharp</code>	504	<code>\textSpannerUp</code>	177
<code>\shiftOff</code>	126	<code>\thumb</code>	90, 165
<code>\shiftOn</code>	126	<code>\tied-lyric</code>	504
<code>\shiftOnn</code>	126	<code>\tieDashed</code>	43
<code>\shiftOnnn</code>	126	<code>\tieDotted</code>	43
<code>\shortfermata</code>	90	<code>\tieDown</code>	43
<code>\showKeySignature</code>	276	<code>\tieNeutral</code>	43
<code>\showStaffSwitch</code>	219	<code>\tieSolid</code>	43
<code>\signumcongruentiae</code>	90	<code>\tieUp</code>	43
<code>\simple</code>	479	<code>\time</code>	52, 64
<code>\skip</code>	47	<code>\times</code>	37, 57
<code>\slashed-digit</code>	511	<code>\tiny</code>	164, 186, 481
<code>\slurDashed</code>	99	<code>\tocItem</code>	338
<code>\slurDashPattern</code>	99	<code>\translate</code>	187, 493
<code>\slurDotted</code>	99	<code>\translate-scaled</code>	187, 493
<code>\slurDown</code>	99	<code>\transparent</code>	511
<code>\slurHalfDashed</code>	99	<code>\transpose</code>	4, 10, 13
<code>\slurHalfSolid</code>	99	<code>\transposedCueDuring</code>	162
<code>\slurNeutral</code>	99	<code>\transposition</code>	19, 157
<code>\slurSolid</code>	99	<code>\treCorde</code>	222
<code>\slurUp</code>	100	<code>\triangle</code>	191
<code>\small</code>	164, 186, 479	<code>\triangle</code>	500
<code>\smallCaps</code>	479	<code>\trill</code>	90, 108
<code>\smaller</code>	184, 186, 480	<code>\tupletDown</code>	37
<code>\snappizzicato</code>	90	<code>\tupletNeutral</code>	37
<code>\sostenutoOff</code>	222	<code>\tupletUp</code>	37
<code>\sostenutoOn</code>	222	<code>\turn</code>	90

<code>\tweak</code>	416	absolute octave specification.....	1
<code>\type</code>	406	accent.....	91, 513
<code>\typewriter</code>	481	accentus.....	513
<code>\unaCorda</code>	222	accepts	406
<code>\underline</code>	184, 482	acciaccatura.....	84
<code>\unfoldRepeats</code>	351	acciaccatura	544
<code>\unHideNotes</code>	168	acciaccatura, multi-note.....	88
<code>\unset</code>	413	accidental.....	4
<code>\upbow</code>	90	accidental on tied note.....	5
<code>\upbow</code>	228	accidental style.....	20
<code>\upmordent</code>	90	accidental style, cautionary, modern voice.....	23
<code>\upprall</code>	90	accidental style, default.....	20, 22
<code>\upright</code>	482	accidental style, forget.....	26
<code>\varcoda</code>	90	accidental style, modern.....	22, 23
<code>\vcenter</code>	494	accidental style, modern voice cautionary.....	23
<code>\verbatim-file</code>	511	accidental style, modern-cautionary.....	22
<code>\verylongfermata</code>	90	accidental style, neo-modern.....	24
<code>\virga</code>	312, 318	accidental style, neo-modern-cautionary.....	24
<code>\virgula</code>	310	accidental style, neo-modern-voice.....	24
<code>\voiceFourStyle</code>	126	accidental style, neo-modern-voice-cautionary.....	25
<code>\voiceNeutralStyle</code>	126	accidental style, no reset.....	25
<code>\voiceOne</code>	123	accidental style, piano.....	23
<code>\voiceOne ... \voiceFour</code>	123	accidental style, piano cautionary.....	24
<code>\voiceOneStyle</code>	126	accidental style, teaching.....	25
<code>\voiceThreeStyle</code>	126	accidental style, voice.....	22
<code>\voiceTwoStyle</code>	126	accidental style, voice, modern cautionary.....	23
<code>\vspace</code>	494	accidental, cautionary.....	5
<code>\whiteout</code>	512	accidental, forced for pitched trill.....	109
<code>\with</code>	404	Accidental, musica ficta.....	306
<code>\with-color</code>	168, 512	accidental, parenthesized.....	5
<code>\with-dimensions</code>	512	accidental, quarter-tone.....	6
<code>\with-url</code>	500	accidental, reminder.....	5
<code>\woodwind-diagram</code>	507	accidentals.....	20, 306, 309
<code>\wordwrap</code>	188, 495	accidentals and simultaneous notes.....	26
<code>\wordwrap-field</code>	494	accidentals in chords.....	26
<code>\wordwrap-internal</code>	513	accidentals, automatic.....	20
<code>\wordwrap-lines</code>	194	accidentals, modern.....	23
<code>\wordwrap-lines</code>	513	accidentals, modern cautionary style.....	23
<code>\wordwrap-string</code>	495	accidentals, modern style.....	22
<code>\wordwrap-string-internal</code>	513	accidentals, multivoice.....	23
		accidentals, piano.....	23
		accidentals, piano cautionary.....	24
		accordion.....	222
.....	81	accordion discant symbols.....	223
~		accordion shift symbols.....	223
~.....	42	accordion shifts.....	223
1		add ChordShape	252
15ma.....	18	addChordShape	544
8		adding a white background to text.....	512
8va.....	18	adding custom fret diagrams.....	251
8ve.....	18	addInstrumentDefinition	156
A		addInstrumentDefinition	544
a due part.....	130	additions, in chords.....	282
absolute.....	1	addQuote	157
absolute dynamics.....	93	addQuote	544
absolute octave entry.....	1	adjusting staff symbol.....	142, 421
		aeolian	16
		after-title-spacing	358
		afterGrace	84
		afterGrace	544
		afterGraceFraction	518
		Aiken shape note heads.....	32
		aikenHeads	32
		aikenHeadsMinor	32
		al niente.....	95

alias	406
align to objects	435
alignAboveContext	408
alignBelowContext	408
aligning markup text	186
aligning markups	186
aligning text	186
aligning to cadenza	88
alignment, text, commands	189
alist	516
allowPageTurn	544
altered chords	281
alternate ending in written-out repeats	116
alternate endings	110
alternative endings with ties	112
alto clef	13
Amazing Grace bagpipe example	277
ambitus	27
amount of staff lines	142
anacrusis in a repeat	111
anacrusis	55
analysis, musicological	174
ancient clef	13
AncientRemoveEmptyStaffContext	149
angle brackets	121
angled hairpins	432
annotate-spacing	394
applyContext	544
applyMusic	544
applyOutput	544
appoggiatura	84
appoggiatura	544
Arabic key signatures	323
Arabic music	322
Arabic music example	326
Arabic music template	326
Arabic note names	323
Arabic semi-flat symbol	323
Arabic time signatures	325
arpeggio	105
arpeggio symbols, special	106
arpeggio, cross-staff parenthesis-style	108
arpeggio, parenthesis-style, cross-staff	108
arpeggioArrowDown	105
arpeggioArrowUp	105
arpeggioBracket	106
arpeggioNormal	105
arpeggioParenthesis	106
arpeggioParenthesisDashed	106
arranger	332
arrow-head	191
articulation-event	159
articulations	90, 310
artificial harmonics	229
assertBeamQuant	544
assertBeamSlope	544
association list	516
aug	280
auto-first-page-number	362
autobeam	64
autoBeaming	64
autoBeamOff	63
autoBeamOn	63
autochange	218
autochange	544

autochange and relative music	218
automatic accidentals	20
automatic beam generation	64
automatic beams, tuning	64
automatic chord diagrams	258
automatic fret diagrams	258
automatic part combining	130
automatic staff changes	218
automatic syllable durations	202
automaticBars	431
available fonts, listing	197

B

backslashed digits	508
bagpipe	276
bagpipe example	277
balloon	171
balloon help	171
Balloon_engraver	171
balloonGrobText	171
balloonGrobText	545
balloonLengthOff	171
balloonLengthOn	171
balloonText	171
balloonText	545
banjo tablature	230
banjo tablatures	264
banjo tunings	264
banjo-c-tuning	264
banjo-modal-tuning	264
banjo-open-d-tuning	264
banjo-open-dm-tuning	264
bar	72, 77
bar	545
bar check	81
bar lines	72
bar lines, closing	72
bar lines, default, changing	77
bar lines, double	72
bar lines, invisible	73
bar lines, manual	73
bar lines, suppressing	431
bar lines, symbols on	177
bar lines, turning off	55
bar number	89
bar number alignment	79
bar number check	81
bar number collision	80
bar number, format	79
bar numbering, turning off	55
bar numbers	77
bar numbers, regular spacing	77
barCheckSynchronize	81
baritone clef	13
BarNumber	77
barNumberCheck	81
barNumberCheck	545
barNumberVisibility	77
barre indications	241
Bartók pizzicato	230
bartype	77
base-shortest-duration	384
bass clef	13

bass note, for chords	283	breve	35, 45
Bass, figured	291	breve rest	45
Bass, thorough	291	broken chord	105
Basso continuo	291		
beam, last in polyphonic voice	68	C	
beam, last in score	68	C clef	13
beaming in polymetric meter	57	cadenza	55, 88
beaming in polymetric music	57	cadenza line breaks	56
beams and line breaks	63	cadenza page breaks	56
beams and melismata	63	cadenza, aligning to	88
beams, cross-staff	217	cadenzaOff	55
beams, custom rules for	63	cadenzaOn	55
beams, feathered	72	caesura	103
beams, manual	63, 69	callback	516
beams, setting rules for	63	capo	245
beams, subdividing	67	cautionary accidental	5
beamSettings	64	cautionary accidental style, piano	24
beatLength	64	cautionary accidentals, piano	24
beats per minute	151	center-align	186
beats, grouping	67	center-column	188
before-title-spacing	358	centered dynamics in piano music	217
beginners' music	30	centering a column of text	482
bendAfter	104	centering text on the page	188
bendAfter	545	change	217
between-scores-system-spacing	358	changing direction of text columns	484
between-system-spacing	358	changing fonts	184
between-title-spacing	358	changing instrument names	155
Bézier control points	438	changing properties	412
Bézier curves	438	changing staff automatically	218
binding gutter	364	changing staff manually	217
binding-offset	360	check-consistency	362
bisbiglando	226	choir staff	137
Bison	518	choral score	206
blank-last-page-force	362	choral tenor clef	14
blank-page-force	362	chord chords	278
BNF	518	chord diagrams	241, 249
bold	184	chord diagrams, automatic	258
bookOutputName	545	chord inversions	283
bookOutputSuffix	545	chord mode	279
bookTitleMarkup	335	chord names	279, 284
bottom-margin	358	Chord names in MIDI	350
bottom-system-spacing	359	chord names with fret diagrams	250
bowing indications	228	chord quality	280
box	189	chord shapes for fretted instruments	252
brace, vertical	137	chord steps, altering	282
braces, nesting of	141	chord, broken	105
braces, various sizes	195	chord, modifying one note in	416
bracket	97, 189, 222	chordChanges	285
bracket, horizontal	174	chordmode	4, 13, 250
bracket, phrasing	174	chordNameExceptions	287
bracket, vertical	137	chordNameLowercaseMinor	287
bracket, volta	114	ChordNames	250
brackets	174	chordNameSeparator	287
brackets, angle	121	chordNoteNamer	287
brackets, nesting of	141	chordPrefixSpacer	287
break-align-symbols	435	chordRootNamer	287
break-visibility	428	chords	121, 284
breakable	63	chords and relative octave entry	4
breakbefore	332	chords and ties	42
breaking lines	366	chords, accidentals in	26
breaking pages	387	chords, cross-staff	220
breaks in unmeasured music	56	chords, fingering	166
breath marks	102	chords, jazz	286
breathe	102		
breathe	545		

chords, splitting across staves with \autochange	219
chords, suppressing repeated	285
church modes	16
church rest	50
circle	189
circling text	497
circulus	513
clef	4
clef	13
clef	545
clef, alto	13
clef, ancient	13
clef, baritone	13
clef, bass	13
clef, C	13
clef, F	13
clef, french	13
clef, G	13
clef, mezzosoprano	13
clef, moderntab	240
clef, percussion	265
clef, soprano	13
clef, subbass	13
clef, tab	240
clef, tenor	13
clef, transposing	14
clef, treble	13
clef, varbaritone	13
clef, violin	13
clef, visibility following explicit change	429
clefts	302, 308
clefts, visibility of octavation	431
closing bar lines	72
closure	516
cluster	123
coda	83, 90, 513
coda on bar line	177
collision, bar number	80
collisions	126
color	168
color in chords	169
color, rgb	169
colored notes	168
colored notes in chords	169
colored objects	168
coloring notes	168
coloring objects	168, 427
coloring text	512
coloring voices	126
colors	168
Colors, list of	456
column	188
columns, text	188
combine	191
combining parts	130
comma intervals	327
common-shortest-duration	384
Completion_heads_engraver	59
composer	332
compound time signatures	53
compressFullBarRests	49, 50
compressing music	41
concatenating text	483
condensing rests	52

consists	406
Context, creating	400
contexts, defining new	406
contexts, keeping alive	401
contexts, layout order	420
contexts, lifetime	401
contexts, nested	408
control pitch	9
control points, tweaking	418
controlling general text alignment	485
controlpitch	9
copyright	332
copyright	335
copyright sign	345
cr	93
creating contexts	401
creating empty text objects	510
creating horizontal spaces in text	487
creating text fractions	508
creating vertical spaces in text	494, 511
crescendo	93
crescHairpin	94
crescTextCresc	94
cross	29
cross note heads	29
cross staff chords	220
cross staff line	219
cross staff notes	220
cross staff stems	220
cross-staff	219
cross-staff	220
cross-staff beams	217
cross-staff chords	220
cross-staff line	219
cross-staff notes	217, 220
cross-staff parenthesis-style arpeggio	108
cross-staff stems	220
cross-staff tremolo	120
cue notes	157, 161
cue notes, formatting	161
cue notes, removing	162
cueDuring	161
cueDuring	545
cues	157, 161
cues, formatting	161
currentBarNumber	77, 89
custodes	300
custom fret diagrams	241
custom fret diagrams, adding	251
custom rehearsal mark	82
customized fret diagram	247
customizing chord names	286
custos	300

D

D.S. al Fine.....	83
dampened notes on fretted instruments.....	262
dashed phrasing slur.....	101
dashed slur.....	99
dashed ties.....	43
deadNote	545
decorating text.....	189
decr	93
decrescendo.....	93

dedication	332
default	20, 22
default accidental style	20, 22
default bar lines, changing	77
default note duration	36
default note names	4
defaultBarType	77
defaultNoteHeads	545
defaultTimeSignature	53
denies	406
Devnull context	208
diagram, fret, customized	247
diagrams, chord for fretted instruments	241
diagrams, fret	241
diagrams, fret, transposing	250
diamond note heads	29
diamond-shaped note heads	229
dim	280
dimHairpin	94
diminuendo	93
dimTextDecr	94
dimTextDecresc	94
dimTextDim	94
discant symbols, accordion	223
displayLilyMusic	545
displayMusic	545
distance between staves	373
distances, absolute	421
distances, scaled	421
divisio	310
divisiones	310
dodecaphonic	25
dodecaphonic accidental style	25
dodecaphonic style, neo-modern	25
doits	104
dorian	16
dotsDown	36
dotsNeutral	36
dotsUp	36
dotted notes	36
dotted phrasing slurs	101
dotted slur	99
dotted ties	43
double bar lines	72
double flat	4
double sharp	4
double time signatures	57
double-dotted notes	36
down bow indication	228
downbow	90, 513
downmordent	513
downprall	513
draw-circle	191
draw-line	191
drawing beams within text	496
drawing boxes with rounded corners	498
drawing boxes with rounded corners around text	500
drawing circles within text	497
drawing graphic objects	189
drawing lines within text	497
drawing solid boxes within text	498
drawing staff symbol	142, 421
drawing triangles within text	500
drum staff	136

drummode	136
drums	265, 267
DrumStaff	136
duration, default	36
durations, of notes	35
durations, scaling	41
dynamic	97
dynamic marks, multiple on one note	93
dynamic marks, new	97
dynamic-event	159
dynamicDown	94
DynamicLineSpanner	94
dynamicNeutral	94
dynamics	93
dynamics, absolute	93
dynamics, centered in keyboard music	217
dynamics, editorial	97
dynamics, parenthesis	97
dynamics, vertical positioning	94
dynamicUp	94

E

easy notation	30
easy play note heads	30
easyHeadsOff	30
easyHeadsOn	30
editorial dynamics	97
embedded graphics	191
embedding graphic objects	189
empty staves	148
encapsulated postscript output	346
enclosing text in a box with rounded corners	500
enclosing text within a box	474
end repeat	114
endSpanners	545
engravers, including in contexts	406
EPS output	346
epsfile	191
espressivo	90
espressivo	93
espressivo	513
espressivo articulation	93
evenFooterMarkup	336
evenHeaderMarkup	335
exceptions, chord names	288
expandFullBarRests	49, 50
expanding music	41
explicitClefVisibility	429
explicitKeySignatureVisibility	429
expressions, markup	182
extended chords	281
extender	206

F

f	93
F clef	13
falls	104
featherDurations	72
featherDurations	545
feathered beams	72
fermata	83, 90, 513
fermata on bar line	177

fermata on multi-measure rest	49
fermataMarkup	49, 50
Feta font	457
ff	93
fff	93
ffff	93
fffff	93
Figured bass	291
figured bass alignment	295
figured bass extender lines	293
fill-line	188
filled-box	191
finalis	310
finding available fonts	197
finger	165
finger change	165
fingering	165
fingering chords	166
fingering instructions for chords	166
fingering vs. string numbers	231
fingerings and multi-measure rests	52
fingerings, adding to fret diagrams	259
fingerings, right hand for fretted instruments	260
first-page-number	362
flag-style	220
flageolet	90, 513
flags	304
flat	4
flat, double	4
Flex	517
follow voice	219
followVoice	219
font	516
font families	185
font families, setting	197
font size	184
font size (notation)	164
font size (notation) scaling	164
font size (notation), standard	165
font size, setting	365
font switching	184
Font, Feta	457
font-interface	165, 195
font-size	164, 165
fonts, changing for entire document	197
fonts, explained	195
fonts, finding available	197
fonts, non-text in markup	195
fontsize	184
fontSize	164
foot marks	90
footer	336
forget	26
forget accidental style	26
format, rehearsal mark	82
formatting text spanners	177
four bar music	366
four-string-banjo	264
fp	93
fragments	157, 161
framing text	189
french clef	13
Frenched score	148
Frenched staff	148
Frenched staves	145
fret	234
fret diagram, customized	247
fret diagrams	241, 249
fret diagrams with chord names	250
fret diagrams, adding custom	251
fret diagrams, adding fingerings	259
fret diagrams, automatic	258
fret diagrams, custom	241
fret diagrams, transposing	250
fret diagrams, ukulele	249
fret-diagram	242
fret-diagram markup	242
fret-diagram-interface	247
fret-diagram-terse	244
fret-diagram-terse markup	244
fret-diagram-verbose	245
fret-diagram-verbose markup	245
FretBoards	249
fretted instruments, chord shapes	252
fretted instruments, dampened notes	262
fretted instruments, harmonics	262
fretted instruments, indicating position and barring	262
fretted instruments, predefined string tunings	239
fretted instruments, right hand fingerings	260
full-measure rests	48
G	
G clef	13
general-align	187
ghost notes	170
glissando	104
global variable	518
glyph	516
glyphs, music	83
grace	84
grace	545
grace notes	84, 276
grace notes within tuplet brackets	41
grace notes, changing layout settings	85
grace notes, following	84
grace notes, tweaking	85
grace-note synchronization	88
grammar for LilyPond	518
grand staff	137
graphic notation	191
graphic objects, drawing	189
graphic objects, embedding	189
graphical object interfaces	517
graphical objects	517
graphics, embedding	189, 191
Gregorian square neumes ligatures	311
Gregorian transcription staff	136
GregorianTranscriptionStaff	136
grid lines	172
Grid_line_span_engraver	172
Grid_point_engraver	172
gridInterval	172
grob	410, 517
grob properties	414
grob-interface	517
grobs, overwriting	427
grobs, visibility of	426
grouping beats	67

grow-direction	72
guitar chord charts	60
guitar note heads	29
guitar strumming rhythms, showing	60
guitar tablature	230
gutter	364

H

hairpin	93
hairpins, angled	432
Hal Leonard	30
halfopen	513
halign	186
harmonic indications in tablature notation	237
harmonic note heads	29
harmonicNote	545
harmonics on fretted instruments	262
harmonics, artificial	229
harmonics, natural	229
harmonicsOn	545
harp pedal diagrams	227
harp pedals	227
harps	226
hbracket	189
header	336
help, balloon	171
hidden notes	168
hideKeySignature	276
hideNotes	168
hideStaffSwitch	219
hiding ancient staves	149
hiding of staves	148
hiding rhythmic staves	149
horizontal bracket	174
horizontal spacing	383
horizontal text alignment	186
horizontal-shift	360
Horizontal_bracket_engraver	174
horizontally centering text	482
hufnagel	298, 299
huge	164, 186
hyphens	206

I

ictus	513
images, embedding	191
immutable objects	517
immutable properties	517
importing stencils into text	511
improvisation	34
improvisationOff	34
improvisationOff	60
improvisationOn	34
improvisationOn	60
including files	339
indent	155, 360, 387
indicating No Chord in ChordNames	284
indicating position and barring for fretted instruments	262
inlining an Encapsulated PostScript image	498
inner-margin	360
inserting music into text	502

inserting PostScript directly into text	499
inserting URL links into text	500
instrument	332
instrument names	154, 348
instrument names, adding to other contexts	155
instrument names, centering	154
instrument names, changing	155
instrument names, short	154
instrument switch	156
instruments, transposing	10
instrumentSwitch	156
instrumentSwitch	545
interface	517
interface, layout	410
interleaved music	133
Internals Reference	398
invisible notes	168
invisible rest	47
invisible stem	170
ionian	16
italic	184

J

jazz chords	286
justified text	188
justified-lines	194
justify	188
justifying lines of text	513
justifying text	488

K

keep tagged music	341
keepWithTag	545
key	16, 32
key signature	4, 16, 306, 309
key signature, visibility following explicit change	429
keyboard instrument staves	216
keyboard music, centering dynamics	217
keyed instrument staves	216
killCues	162
killCues	546
kirchenpausen	50

L

label	546
laissez vibrer	43
laissezVibrer	43
landscape	357
language, note names in other	7
language, pitch names in other	7
large	164, 186
larger	184, 186
layers	427
layout file	365
layout interface	410
layout objects	517
ledger lines, setting	142
left aligning text	489
left-align	186
left-margin	360

length	220
length of notes.....	35
lexer.....	517
lheel.....	513
Ligatures.....	300
ligatures in text.....	483
line breaks.....	73, 366
line breaks and beams.....	63
line breaks in cadenzas.....	56
line breaks in unmetered music.....	56
line, cross-staff.....	219
line, staff-change.....	219
line, staff-change follower.....	219
line-width	360, 387
lineprall.....	513
lines, grid.....	172
lines, vertical between staves.....	172
List of colors.....	456
listing available fonts.....	197
locrian	16
longa	35, 45
longa rest.....	45
longfermata.....	513
lower	186
lowering text.....	490
ltoe.....	513
ly:minimal-breaking	370
ly:optimal-breaking	368
ly:page-turn-breaking	369
lydian	16
lyrics.....	199
lyrics and beaming.....	64
lyrics and melodies.....	202
lyrics assigned to one voice.....	123
lyrics, aligning with sporadic melody.....	402
Lyrics, increasing space between.....	208
lyrics, keeping inside margin.....	176
lyrics, skip.....	47
lyrics, variables.....	201

M

m	280
magnify	184
magnifying text.....	476
magstep	164, 421
maj	280
major	16
major seven symbols	287
majorSevenSymbol	287
makam	327
makamlar	327
make-dynamic-script	97
make-pango-font-tree	197
makeClusters	123
makeClusters	546
manual bar lines.....	73
manual beams.....	63, 69
manual beams, direction shorthand for.....	69
manual beams, grace notes.....	69
manual measure lines.....	73
manual rehearsal mark.....	82
manual repeat mark.....	114
manual staff changes.....	217
Manuals	1
maqam	322
maqams	322
marcato	91, 513
margin, text running over.....	176
mark	81, 177
mark, phrasing.....	101
mark, rehearsal.....	81
mark, rehearsal, format.....	82
mark, rehearsal, manual.....	82
mark, rehearsal, style.....	82
marks, text.....	177
markup	177
markup	181
markup	182
markup expressions.....	182
markup mode, quoted text.....	183
markup mode, special characters.....	183
markup on multi-measure rest.....	49
markup syntax.....	182
markup text.....	182
markup text alignment commands.....	189
markup text padding.....	190
markup text, aligning.....	186
markup text, decorating.....	189
markup text, framing.....	189
markup text, justified.....	188
markup text, multi-page.....	194
markup text, wordwrapped.....	188
markup, centering on the page.....	188
markup, multi-line.....	188
markup, multi-page.....	194
markup, music notation inside.....	192
markup, score inside.....	193
markuplines	181, 194
markuplines	195
markups, aligning.....	186
maxima	35, 45
maxima rest.....	45
measure check.....	81
measure groupings.....	67
measure lines.....	72
measure lines, invisible.....	73
measure lines, manual.....	73
measure number.....	89
measure number and repeats.....	114
measure number check.....	81
measure number, format.....	79
measure numbers.....	77
measure repeats.....	117
measure sub-grouping.....	67
measure, change length.....	55
measure, partial.....	55
measureLength	64, 89
measurePosition	55, 89
Medicaea, Editio.....	298, 299
medium intervals.....	322
melisma	204, 206
melismata	204
melismata, beams.....	63
melody rhythms, showing.....	60
mensural	298, 299
Mensural ligatures.....	307
mensural music, transcription of.....	140
MensuralStaff	136

MensuralStaffContext	301
MensuralVoiceContext	301
mensuration sign	303
mensurstriche layout	140
mergeDifferentlyDottedOff	126
mergeDifferentlyDottedOn	126
mergeDifferentlyHeadedOff	126
mergeDifferentlyHeadedOn	126
merging notes	126
merging text	483
meter	52
meter	332
meter style	53
meter, polymetric	57
metronome mark	151
metronome marking with text	151
mezzosoprano clef	13
mf	93
microtones	7
Microtones in MIDI	350
MIDI	19, 347
MIDI block	349
MIDI context definitions	350
MIDI transposition	19
MIDI, chord names	350
MIDI, microtones	350
MIDI, Pitches	350
MIDI, quarter tones	350
MIDI, Rhythms	350
minimumFret	234
minimumPageTurnLength	369
minimumRepeatLengthForPageTurn	369
minor	16
mixed	222
mixolydian	16
mode	518
modern	22
modern accidental style	22, 23
modern accidentals	23
modern cautionary accidental style	23
modern style accidentals	22, 23
modern style cautionary accidentals	23
modern-cautionary	23
modern-cautionary accidental style	22
modern-voice	23
modern-voice-cautionary	23
moderntab clef	240
modes	16
modifiers, in chords	280
mordent	90, 513
mordent, down	90
mordent, up	90
movements, multiple	329
mp	93
multi-line markup	188
multi-line text	188
multi-measure rest with markup	49
multi-measure rest, attaching fermata	49
multi-measure rest, attaching text	49
multi-measure rest, contracting	49
multi-measure rest, expanding	49
multi-measure rest, script	49
multi-measure rests	48
multi-measure rests and fingerings	52
multi-measure rests, positioning	50

multi-note acciaccatura	88
multi-page markup	194
MultiMeasureRestText	49
multiple dynamic marks on one note	93
multiple phrasing slurs	101
multiple slurs	99
multiple voices	126
multivoice accidentals	23
music glyphs	83
music inside markup	192
music, beginners'	30
music, unmetered	89
Musica ficta	306
musicglyph	83
musicMap	546
musicological analysis	174
musicQuotes	518
mutable objects	517
mutable properties	517

N

N.C. symbol	284
name	406
name of singer	210
natural harmonics	229
natural pitch	4
natural sign	4
neo-modern	24
neo-modern accidental style	24
neo-modern-cautionary	24
neo-modern-cautionary accidental style	24
neo-modern-voice	24
neo-modern-voice accidental style	24
neo-modern-voice-cautionary	25
neo-modern-voice-cautionary accidental style	25
neomensural	299
nested contexts	408
nested repeat	114
nested staff brackets	141
nesting of staves	141
new contexts	400
new dynamic marks	97
new staff	136
niente, al	95
no chord symbol	284
no reset accidental style	25
no-reset	25
noBeam	69
non-ASCII characters	344
non-empty texts	176
non-musical symbols	191
non-text fonts in markup	195
noPageBreak	546
noPageTurn	546
normal repeat	110
normalsize	164, 186
notation font size	164
notation inside markup	192
notation, explaining	171
notation, graphic	191
note cluster	123
note collisions	126
note duration, default	36

note durations	35
note grouping bracket	174
note head styles	29, 473
note heads	164
note heads, Aiken	32
note heads, ancient	304
note heads, cross	29
note heads, diamond	29
note heads, diamond-shaped	229
note heads, easy notation	30
note heads, easy play	30
note heads, guitar	29
note heads, harmonic	29
note heads, improvisation	34
note heads, parlato	29
note heads, practice	30
note heads, sacred harp	32
note heads, shape	32
note heads, slashed	34
note heads, Southern Harmony	32
note heads, special	29
note lengths	35
note names, default	4
note names, Dutch	4
note names, other languages	7
note-event	159
Note_heads_engraver	59
notes within text by log and dot-count	502
notes within text by string	502
notes, colored	168
notes, colored in chords	169
notes, cross-staff	217, 220
notes, dotted	36
notes, double-dotted	36
notes, ghost	170
notes, hidden	168
notes, invisible	168
notes, parenthesized	170
notes, smaller	163
notes, splitting	59
notes, transparent	168
notes, transposition of	10
null	186
number of staff lines	142
numbers, bar	77
numbers, measure	77
numericTimeSignature	53

O

objects, colored	168
objects, coloring	427
objects, overwriting	427
objects, rotating	432
objects, visibility of	426
octave specification, relative	2
octavated clefs, visibility of	431
octavation	18
octave changing mark	1
octave check	9
octave correction	9
octave entry, absolute	1
octave entry, relative	2
octave specification, absolute	1

octave transposition	14
octaveCheck	9
octaveCheck	546
oddFooterMarkup	335
oddHeaderMarkup	335
oneVoice	123
open	90, 513
open string indication	228
opus	332
orchestral strings	228
organ pedal marks	90
orientation	357
ornamentation	90
ornaments	84, 90
ossia	145, 149, 408
ottava	18
ottava	546
Ottoman music	327
outer-margin	360
output-count	518
output-def	518
output-suffix	518
outside-staff-horizontal-padding	382
outside-staff-padding	382
outside-staff-priority	382
overrideBeamSettings	546
overrideProperty	546
overrides, reverting	415
overriding for only one moment	416
overriding properties within text markup	510
overwriting objects	427

P

p	93
pad-around	190
pad-markup	190
pad-to-box	190
pad-x	190
padding	411
padding around text	190
padding text	491
padding text horizontally	491
page breaks	387
page breaks in cadenzas	56
page breaks in unmetred music	56
page breaks, forcing	332
page layout	336, 387
page size	357
page-breaking-between-system-spacing	362
page-count	362
page-limit-inter-system-space	362
page-limit-inter-system-space-factor	362
page-spacing-weight	362
pageBreak	546
pageTurn	546
palmMute	546
palmMuteOn	546
Pango	195
paper size	357
paper-height	359
paper-width	360
parallel music	133
parallelMusic	133

<code>parallelMusic</code>	546	<code>piano staves</code>	216
<code>parentheses</code>	170	<code>piano-cautionary</code>	24
<code>parenthesize</code>	170	<code>PianoStaff</code>	216, 218
<code>parenthesize</code>	547	<code>pickup in a repeat</code>	111
<code>parenthesized accidental</code>	5	<code>pickup measure</code>	55
<code>parlato</code>	199	<code>piece</code>	332
<code>parlato note heads</code>	29	<code>pipeSymbol</code>	81
<code>parser</code>	518	<code>pitch names</code>	1
<code>parser variable</code>	518	<code>pitch names, other languages</code>	7
<code>parseStringResult</code>	518	<code>pitch range</code>	27
<code>part combiner</code>	130	<code>Pitch_squash_engraver</code>	60
<code>partcombine</code>	130	<code>pitched trill with forced accidental</code>	109
<code>partcombine</code>	547	<code>pitched trills</code>	109
<code>partCombineListener</code>	518	<code>pitchedTrill</code>	109
<code>partial</code>	55	<code>pitchedTrill</code>	547
<code>partial measure</code>	55	<code>pitches</code>	1
<code>pause mark</code>	102	<code>Pitches in MIDI</code>	350
<code>pedal diagrams, harp</code>	227	<code>pitches, transposition of</code>	10
<code>pedal indication styles</code>	222	<code>pitchnames</code>	518
<code>pedal indication, bracket</code>	222	<code>pizzicato, Bartók</code>	230
<code>pedal indication, mixed</code>	222	<code>pizzicato, snap</code>	230
<code>pedal indication, text</code>	222	<code>placing horizontal brackets around text</code>	498
<code>pedal marks, organ</code>	90	<code>placing parentheses around text</code>	499
<code>pedal sustain style</code>	222	<code>placing vertical brackets around text</code>	497
<code>pedal, sostenuto</code>	222	<code>poet</code>	332
<code>pedal, sustain</code>	222	<code>pointAndClickOff</code>	547
<code>pedals, harp</code>	227	<code>pointAndClickOn</code>	547
<code>pedals, piano</code>	222	<code>polymetric scores</code>	404
<code>pedalSustainStyle</code>	222	<code>polymetric signatures</code>	57
<code>percent</code>	117	<code>polyphonic music</code>	126
<code>percent repeats</code>	117	<code>polyphony, single-staff</code>	123
<code>percussion</code>	265, 267	<code>portato</code>	91, 513
<code>percussion clef</code>	265	<code>positioning multi-measure rests</code>	50
<code>percussion staff</code>	136	<code>postscript</code>	191
<code>Petrucci</code>	298, 299	<code>pp</code>	93
<code>phrasing bracket</code>	174	<code>ppp</code>	93
<code>phrasing marks</code>	101	<code>pppp</code>	93
<code>phrasing slur</code>	99	<code>ppppp</code>	93
<code>phrasing slur, dashed</code>	101	<code>practice note heads</code>	30
<code>phrasing slur, defining dash patterns</code>	102	<code>prall</code>	90, 513
<code>phrasing slur, dotted</code>	101	<code>prall, down</code>	90
<code>phrasing slur, half solid and half dashed</code>	102	<code>prall, up</code>	90
<code>phrasing slur, multiple</code>	101	<code>pralldown</code>	513
<code>phrasing slur, simultaneous</code>	101	<code>prallmordent</code>	90, 513
<code>phrasing slurs</code>	101	<code>prallprall</code>	90, 513
<code>phrasing, in lyrics</code>	204	<code>prallup</code>	513
<code>phrasingSlurDashed</code>	101	<code>predefined string tunings for fretted instruments</code>	239
<code>phrasingSlurDashPattern</code>	102	<code>predefinedFretboardsOff</code>	258
<code>phrasingSlurDashPattern</code>	547	<code>predefinedFretboardsOn</code>	258
<code>phrasingSlurDotted</code>	101	<code>prima volta</code>	110
<code>phrasingSlurDown</code>	101	<code>print-all-headers</code>	335, 362
<code>phrasingSlurHalfDashed</code>	102	<code>print-first-page-number</code>	363
<code>phrasingSlurHalfSolid</code>	102	<code>print-page-number</code>	363
<code>phrasingSlurNeutral</code>	101	<code>printing chord names</code>	284
<code>phrasingSlurSolid</code>	101	<code>printing order</code>	427
<code>phrasingSlurUp</code>	101	<code>printing reserved characters</code>	183
<code>phrygian</code>	16	<code>printing special characters</code>	183
<code>piano</code>	23	<code>prob</code>	518
<code>piano accidental style</code>	23	<code>properties</code>	412
<code>piano accidentals</code>	23	<code>properties, grob</code>	414
<code>piano cautionary accidental style</code>	24	<code>property object</code>	518
<code>piano cautionary accidentals</code>	24	<code>punctuation</code>	199
<code>piano music, centering dynamics</code>	217	<code>putting space around text</code>	491
<code>piano pedals</code>	222		
<code>piano staff</code>	137		

Q

quarter tones.....	5
Quarter tones in MIDI.....	350
quarter-tone accidental.....	6
quoted text.....	176
quoted text in markup mode.....	183
<code>quotedCueEventTypes</code>	159
<code>quotedEventTypes</code>	159
<code>quoteDuring</code>	157, 161
<code>quoteDuring</code>	547
quotes, in lyrics.....	199, 200
quoting other voices.....	157, 161

R

<code>r</code>	45
<code>R</code>	48
<code>ragged-bottom</code>	363
<code>ragged-last</code>	363, 387
<code>ragged-last-bottom</code>	363
<code>ragged-right</code>	363, 387
railroad tracks.....	103
<code>raise</code>	186
raising text.....	492
range of pitches.....	27
Ratisbona, Editio.....	299
referencing page numbers in text.....	510
regular line breaks.....	366
rehearsal mark format.....	82
rehearsal mark style.....	82
rehearsal mark, manual.....	82
rehearsal marks.....	81
<code>relative</code>	2, 4, 13, 218
relative music and autochange.....	218
relative octave entry.....	2
relative octave entry and chords.....	4
relative octave entry and transposition.....	4
relative octave specification.....	2
relative pitch in chords.....	121
reminder accidental.....	5
removals, in chords.....	282
remove tagged music.....	341
<code>RemoveEmptyRhythmicStaffContext</code>	149
<code>RemoveEmptyStaffContext</code>	148, 149
<code>removeWithTag</code>	547
removing cue notes.....	162
removing cues.....	162
renaissance music.....	140
repeat and measure number.....	114
repeat and slur.....	114
repeat bars.....	72
repeat number, changing.....	114
repeat timing information.....	114
repeat volta, changing.....	114
repeat with alternate endings.....	110
repeat with anacrusis.....	111
repeat with pickup.....	111
repeat with upbeat.....	111
repeat, ambiguous.....	114
repeat, end.....	114
repeat, manual.....	114
repeat, measure.....	117
repeat, nested.....	114
repeat, normal.....	110

repeat, percent.....	117
repeat, short.....	117
repeat, start.....	114
repeat, tremolo.....	119
repeat, unfold.....	116
<code>repeatCommands</code>	114
repeating ties.....	43
repeats.....	75
repeats in MIDI.....	351
repeats with ties.....	112
repeats, written-out.....	116
<code>repeatTie</code>	43
repetitious music.....	116
reserved characters, printing.....	183
<code>resetRelativeOctave</code>	547
resizing of staves.....	145
<code>rest</code>	45
rest, church.....	50
rest, collisions of.....	52
rest, condensing ordinary.....	52
rest, entering durations.....	45
rest, full-measure.....	48
rest, invisible.....	47
rest, multi-measure.....	45, 48
rest, specifying vertical position.....	45
rest, whole for a full measure.....	48
rest, whole-measure.....	45
rest-event.....	159
rests, ancient.....	305
<code>reverseturn</code>	90, 513
<code>revertBeamSettings</code>	66, 547
reverting overrides.....	415
<code>rfz</code>	93
rgb color.....	169
<code>rgb-color</code>	169
rheel.....	513
rhythmic staff.....	136
<code>RhythmicStaff</code>	136
Rhythms in MIDI.....	350
rhythms, showing melody.....	60
right aligning text.....	492
right hand fingerings for fretted instruments.....	260
<code>right-align</code>	186
<code>right-margin</code>	360
<code>rightHandFinger</code>	260
<code>rightHandFinger</code>	547
root of chord.....	280
rotating objects.....	432
rotating text.....	493
<code>rounded-box</code>	189
<code>rtoe</code>	513

S

<code>s</code>	47
sacred harp note heads.....	32
<code>sacredHarpHeads</code>	32
<code>sacredHarpHeadsMinor</code>	32
SATB.....	206
scalable vector graphics output.....	346
<code>scaleDurations</code>	41, 57
<code>scaleDurations</code>	547
scaling durations.....	41
scaling text.....	493
Scheme object.....	519

Scheme variable.....	518	skipTypesetting	346
score inside markup.....	193	slashed digits.....	511
scoreTitleMarkup	335	slashed note heads.....	34
Scottish highland bagpipe.....	276	slides in tablature notation.....	237
script on multi-measure rest.....	49	slur and repeat.....	114
scripts.....	90	slur style.....	99
seconda volta.....	110	slur, dashed.....	99
segno.....	74, 83, 90, 513	slur, dashed phrasing.....	101
segno on bar line.....	177	slur, definind dash patterns for phrasing.....	102
selecting font size (notation).....	164	slur, defining dash patterns.....	99
Semai form.....	325	slur, dotted.....	99
semi-flat.....	7	slur, dotted phrasing.....	101
Semi-flat symbol appearance.....	323	slur, half dashed and half solid.....	99
semi-flats.....	5	slur, half solid and half dashed phrasing.....	102
semi-sharp.....	7	slur, multiple phrasing.....	101
semi-sharps.....	5	slur, phrasing.....	99, 101
semicirculus.....	513	slur, phrasing, defining dash patterns.....	102
separate text.....	181	slur, simultaneous phrasing.....	101
sesqui-flat.....	7	slur, solid.....	99
sesqui-sharp.....	7	slurDashed	99
set	64	slurDashPattern	99
set-accidental-style	20	slurDashPattern	547
set-octavation	18	slurDotted	99
setBeatGrouping	547	slurDown	99
setting extent of text objects.....	512	slurHalfDashed	99
setting horizontal text alignment.....	486	slurHalfSolid	99
setting of ledger lines.....	142	slurNeutral	99
setting subscript in standard font size.....	477	slurs.....	99
setting superscript in standard font size.....	477	slurs, above notes.....	99
seventh chords.....	280	slurs, below notes.....	99
sf	93	slurs, manual placement.....	99
sff	93	slurs, modifying.....	438
sfz	93	slurs, multiple.....	99
shape notes.....	32	slurs, simultaneous.....	99
shared properties.....	517	slurSolid	99
sharp.....	4	slurUp	100
sharp, double.....	4	small	164, 186
shift note.....	126	smaller	184, 186
shift rest, automatic.....	126	smaller notes.....	163
shiftDurations	547	smob.....	519
shifting voices.....	126	snap pizzicato.....	230
shiftOff	126	snappizzicato.....	513
shiftOn	126	Solesmes.....	299
shiftOnn	126	solid slur.....	99
shiftOnnn	126	solo part.....	130
short-indent	155, 361	soprano clef.....	13
shortfermata.....	513	sos.....	222
show-available-fonts	197	sostenuto pedal.....	222
showFirstLength	346	sostenutoOff	222
showFirstLength	518	sostenutoOn	222
showKeySignature	276	Sound.....	347
showLastLength	346	Southern Harmony note heads.....	32
showLastLength	518	southernHarmonyHeads	32
showStaffSwitch	219	southernHarmonyHeadsMinor	32
signatures, polymetric.....	57	sp	93
signumcongruentiae.....	513	space between staves.....	373
simple closure.....	516	space inside systems.....	373
simple text strings.....	479	spacer note.....	47
simple text strings with tie characters.....	504	spacer rest.....	47
simultaneous notes and accidentals.....	26	spaces, in lyrics.....	199, 200
simultaneous phrasing slurs.....	101	spacing	384
simultaneous slurs.....	99	Spacing lyrics.....	208
singer name.....	210	spacing of ledger lines.....	142
single-staff polyphony.....	123	spacing, display of layout.....	394
skip	47	spacing, horizontal.....	383

spacing, vertical	373	stem, invisible	170
spacingTweaks	547	stem, neutral	170
special arpeggio symbols	106	stem, up	170
special characters in markup mode	183	stem, with slash	86
special note heads	29	stem-spacing-correction	384
splitting notes	59	stemDown	170
spp	93	stemLeftBeamCount	70
Sprechgesang	199	stemNeutral	170
Square neumes ligatures	311	stemRightBeamCount	70
staccatissimo	91, 513	stems, cross-staff	220
staccato	91, 513	stemUp	170
stacking text in a column	483	stencil	519
staff change line	219	stencil, removing	427
staff changes, automatic	218	stop staff lines	142
staff changes, manual	217	stopGroup	174
staff distance	373	stopped	90, 513
staff group	137	stopping a staff	143
staff initiation	136	stopStaff	143, 145
staff instantiation	136	stopStaff	148
staff line, thickness of	142	stopTrillSpan	108
staff lines, amount of	142	storePredefinedDiagram	252
staff lines, number of	142	storePredefinedDiagram	548
staff size, setting	365	string numbers	231
staff switching	219	string vs. fingering numbers	231
staff symbol, drawing	142	string, indicating open	228
staff symbol, setting of	142, 421	strings, orchestral	228
staff, choir	137	strings, writing for	228
staff, drum	136	stringTunings	249
staff, empty	148	StringTunings	239
staff, Frenched	145	strumming rhythms, showing	60
staff, grand	137	style, rehearsal mark	82
staff, hiding	148	style, slur	99
staff, multiple	137	styledNoteHeads	548
staff, nested	141	styles, note heads	29
staff, new	136	styles, voice	126
staff, percussion	136	sub	185
staff, piano	137	subbass clef	13
staff, resizing of	145	subscript	185
staff, single	136	subscript text	480
staff, starting	143	subsubtitle	332
staff, stopping	143	subtitle	332
staff-change line	219	suggestAccidentals	306
staff-padding	217	super	185
Staff.midiInstrument	348	superscript	185
Staff_symbol_engraver	148	superscript text	480
standalone text	181	sus	283
standard font size (notation)	165	sustain pedal	222
stanza number	210	sustain pedal style	222
start of system	137	sustainOff	222
start repeat	114	sustainOn	222
start staff lines	142	SVG output	346
start-repeat	114	switching fonts	184
startGroup	174	switching instruments	156
starting a staff	143	symbols, non-musical	191
startStaff	143, 145	syntax, markup	182
startTrillSpan	108	system	137
staves, keyboard instruments	216	system start delimiters	137
staves, keyed instruments	216	system start delimiters, nested	141
staves, multiple	137	system-count	364
staves, nested	141	system-separator-markup	363
staves, piano	216	systems-per-page	364
stem	170		
Stem	220		
stem, direction	170		
stem, down	170		

T

tab clef	240
tabChordRepetition	548
tablature	136, 230
tablature and harmonic indications	237
tablature and slides	237
tablature, banjo	230, 239, 264
tablature, bass guitar	239
tablature, guitar	230, 239
tablature, mandolin	239
tablature, predefined string tunings	239
tablature, ukulele	239
tablatures, basic	233
tablatures, custom	239
tablatures, default	233
tabstaff	136
TabStaff	136, 233
TabVoice	233
tag	341
tag	548
tagline	333
tagline	335
taor	276
taqasim	325
teaching	25
teaching accidental style	25
teeny	164, 186
Template Arabic music	326
tempo	151
tempo marks within tuplet brackets	41
tenor clef	13
tenor clef, choral	14
tenuto	91, 513
text	222
text alignment commands	189
text columns, left-aligned	490
text columns, right-aligned	492
text in columns	188
text in volta bracket	115
text items, non-empty	176
text marks	177
text markup	182
text on bar line	177
text on multi-measure rest	49
text outside margin	176
text padding	190
Text scripts	176
text size	184
text spanners	177
text spanners, formatting	177
text spread over multiple pages	194
text, aligning	186
text, centering on the page	188
text, decorating	189
text, framing	189
text, horizontal alignment	186
text, justified	188
text, keeping inside margin	176
text, multi-line	188
Text, other languages	175
text, separate	181
text, standalone	181
text, top-level	181
text, vertical alignment	186
text, wordwrapped	188
textLengthOff	176
textLengthOn	50, 176
textLenthOff	50
textSpannerDown	177
textSpannerNeutral	177
textSpannerUp	177
thickness of staff lines	142
Thorough bass	291
thumb	165
thumb	513
thumb marking	90
thumb-script	165
tie	42
tied note, accidental	5
tieDashed	43
tieDashPattern	548
tieDotted	43
tieDown	43
tieNeutral	43
ties and chords	42
ties and volta brackets	43
ties in alternative endings	112
ties in repeats	112
ties, appearance	43
ties, dashed	43
ties, dotted	43
ties, in lyrics	200, 204
ties, laissez vibrer	43
ties, modifying	438
ties, placement	43
ties, repeating	43
tieSolid	43
tieUp	43
time	52, 64
time administration	89
time signature	52
time signature style	53
time signature, compound	53
time signature, visibility of	52
time signatures	303
time signatures, double	57
Time signatures, multiple	404
time signatures, polymetric	57
times	37, 57
timeSignatureFraction	57
timing (within the score)	89
timing information and repeats	114
tiny	164, 186
title	332
titles	336
tocItem	548
Top	1
top-level text	181
top-margin	359
top-system-spacing	359
top-title-spacing	359
toplevel-bookparts	518
toplevel-scores	518
transcription of mensural music	140
translate	187
translate-scaled	187
translating text	493
transparent notes	168
transparent, making objects	427

transpose 4, 10, 13
transposedCueDuring 162
transposedCueDuring 548
 transposing 10
 transposing clef 14
 transposing fret diagrams 250
 transposing instrument 19
 transposing instruments 10
 transposition 10
transposition 19, 157
transposition 548
 transposition and relative octave entry 4
 transposition of notes 10
 transposition of pitches 10
 transposition, instrument 19
 transposition, MIDI 19
 tre corde 222
 treble clef 13
treCorde 222
tremolo 119
 tremolo beams 119
 tremolo marks 119
 tremolo, cross-staff 120
tremoloFlags 119
 triads 280
triangle 191
 trill 90
trill 108
 trill 513
 trill, pitched with forced accidental 109
 trills 108
 trills, pitched 109
 triplet formatting 38
 triplets 37
 tuning automatic beaming 64
 tunings, banjo 264
 tuplet bracket placement 37
 tuplet formatting 38
 Tuplet number changes 38
tupletDown 37
tupletNeutral 37
TupletNumber 38
tupletNumberFormatFunction 38
 tuplets 37
tupletSpannerDuration 38
tupletUp 37
 Turkish music 327
 Turkish note names 327
 turn 90, 513
tweak 548
 tweaking 416
 tweaking control points 418
 tweaking grace notes 85
 tweaks in a variable 418
 tweaks in lyrics 418
two-sided 364
type 406
 typeface 516
 typeset text 182

U

U.C. 222
 ukulele 242

una corda 222
unaCorda 222
underline 184
 underlining text 482
unfold 116
 unfold music 116
 unfold music with alternate endings 116
 unfold repeat 116
 unfold repeat with alternate endings 116
unfoldRepeats 548
unHideNotes 168
 Unicode 344
 unmetered music 55, 89
 unmetered music, line breaks 56
 unmetered music, page breaks 56
 up bow indication 228
 upbeat 55
 upbeat in a repeat 111
 upbow 90, 513
 upmordent 513
 upprall 513
 UTF-8 344

V

varbaritone clef 13
 varcoda 90, 513
 variables 331
 variables, use of 340
 Vaticana, Editio 298, 299
VaticanaStaff 136
 VaticanaStaffContext 308
 VaticanaVoiceContext 308
 vertical lines between staves 172
 vertical positioning of dynamics 94
 vertical spacing 373, 387
 vertical text alignment 186
 vertically centering text 494
 verylongfermata 513
 violin clef 13
 visibility of objects 426
 visibility of octavated clefs 431
voice 20, 22
 voice 123
Voice 123
 voice accidental style 22
 voice styles 126
 voice, following 219
voiceOne 123
 voices, multiple 126
 volta 110
 volta bracket 114
 volta bracket with text 115
 volta brackets and ties 43
 volta, prima 110
 volta, seconda 110

W

whichBar 77
 White mensural ligatures 307
 whitespace 332
 whole rest for a full measure 48
 wind instruments 273

`with-color` 168
`withMusicProperty` 548
`wordwrap` 188
`wordwrap-lines` 194
wordwrapped text 188
writing music in parallel 133
written-out repeats 116

X

`x11 color` 168, 169
`x11-color` 168, 169
`xNote` 548
`xNotesOn` 548