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 2.27.5. It assumes that the reader is familiar with the material in the *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 `https://lilypond.org/`.

Copyright © 1998–2026 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 2.27.5

Table of Contents

Musical notation

1 Pitches ... 3

1.1 Writing pitches ... 3
1.1.1 Absolute octave entry ... 3
1.1.2 Relative octave entry ... 4
1.1.3 Accidentals ... 8
1.1.4 Approximate pitches ... 10
1.1.5 Note names in other languages ... 10
1.2 Changing multiple pitches ... 12
1.2.1 Octave checks ... 12
1.2.2 Transpose ... 13
1.2.3 Inversion ... 16
1.2.4 Retrograde ... 17
1.2.5 Modal transformations ... 17
1.3 Displaying pitches ... 19
1.3.1 Clef ... 19
1.3.2 Key signature ... 24
1.3.3 Ottava brackets ... 26
1.3.4 Instrument transpositions ... 30
1.3.5 Automatic accidentals ... 31
1.3.6 Alternate accidental glyphs ... 39
1.3.7 Ambitus ... 40
1.4 Note heads ... 43
1.4.1 Special note heads ... 43
1.4.2 Easy notation note heads ... 45
1.4.3 Shape note heads ... 46
1.4.4 Improvisation ... 50

2 Rhythms ... 52

2.1 Writing rhythms ... 52
2.1.1 Durations ... 52
2.1.2 Tuplets ... 55
2.1.3 Scaling durations ... 61
2.1.4 Ties ... 63
2.2 Writing rests ... 67
2.2.1 Rests ... 67
2.2.2 Invisible rests ... 69
2.2.3 Full measure rests ... 71
2.2.4 Caesuras ... 76
2.3 Displaying rhythms ... 78
2.3.1 Measures ... 78
2.3.2 Time signature ... 80
2.3.3 Metronome marks ... 87
2.3.4 Upbeats ... 92
2.3.5 Unmetered music ... 93
2.3.6 Polymetric notation ... 95
2.3.7 Automatic note splitting ... 98

2.3.8 Showing melody rhythms ... 100
2.4 Beams ... 102
2.4.1 Automatic beams ... 102
2.4.2 Setting automatic beam behavior ... 106
2.4.3 Manual beams ... 115
2.4.4 Feathered beams ... 119
2.4.5 Slashed beams ... 120
2.5 Bars ... 121
2.5.1 Bar lines ... 121
2.5.2 Automatic bar lines ... 131
2.5.3 Bar numbers ... 133
2.5.4 Bar and bar number checks ... 141
2.5.5 Rehearsal marks ... 142
2.5.6 Measure counts ... 144
2.5.7 Section divisions ... 147
2.6 Special rhythmic concerns ... 148
2.6.1 Grace notes ... 148
2.6.2 Aligning to cadenzas ... 153

3 Expressive marks ... 156
3.1 Expressive marks attached to notes ... 156
3.1.1 Articulations and ornamentations ... 156
3.1.2 Dynamics ... 160
3.1.3 New dynamic marks ... 168
3.2 Expressive marks as curves ... 171
3.2.1 Slurs ... 171
3.2.2 Phrasing slurs ... 174
3.2.3 Breath marks ... 176
3.2.4 Falls and doits ... 177
3.3 Expressive marks as lines ... 177
3.3.1 Glissando ... 177
3.3.2 Arpeggio ... 182
3.3.3 Trills ... 186

4 Repeats ... 189
4.1 Long repeats ... 189
4.1.1 Written-out repeats ... 189
4.1.2 Simple repeats ... 190
4.1.3 Alternative endings ... 191
4.1.4 Other variation in repeated sections ... 193
4.1.5 Al-fine repeats ... 195
4.1.6 Segno repeat structure ... 196
4.1.7 Segno repeat appearance ... 199
4.1.8 Manual repeat marks ... 205
4.2 Short repeats ... 209
4.2.1 Percent repeats ... 209
4.2.2 Tremolo repeats ... 212

5 Simultaneous notes ... 216

5.1 Single voice ... 216
5.1.1 Chorded notes ... 216
5.1.2 Chord repetition ... 218
5.1.3 Simultaneous expressions ... 220
5.1.4 Clusters ... 221
5.2 Multiple voices ... 222
5.2.1 Single-staff polyphony ... 222
5.2.2 Voice styles ... 227
5.2.3 Collision resolution ... 227
5.2.4 Merging rests ... 232
5.2.5 Automatic part combining ... 233
5.2.6 Writing music in parallel ... 238

6 Staff notation ... 242

6.1 Displaying staves ... 242
6.1.1 Instantiating new staves ... 242
6.1.2 Grouping staves ... 244
6.1.3 Nested staff groups ... 248
6.1.4 Separating systems ... 249
6.2 Modifying single staves ... 250
6.2.1 Staff symbol ... 250
6.2.2 Ossia staves ... 254
6.2.3 Hiding staves ... 257
6.3 Writing parts ... 262
6.3.1 Instrument names ... 262
6.3.2 Quoting other voices ... 266
6.3.3 Formatting cue notes ... 269
6.3.4 Compressing empty measures ... 274

7 Editorial annotations ... 278

7.1 Inside the staff ... 278
7.1.1 Selecting notation font size ... 278
7.1.2 Fingering instructions ... 282
7.1.3 Gliding fingers ... 285
7.1.4 Hidden notes ... 289
7.1.5 Coloring objects ... 289
7.1.6 Staff highlights ... 292
7.1.7 Brackets for optional material ... 295
7.1.8 Parentheses ... 296
7.1.9 Stems ... 297
7.2 Outside the staff ... 298
7.2.1 Note names ... 298
7.2.2 Balloon help ... 300
7.2.3 Grid lines ... 302
7.2.4 Analysis brackets ... 304

8 Text ... 308

8.1 Writing text ... 308
8.1.1 Text objects overview ... 308

8.1.2 Text scripts ... 310
8.1.3 Text spanners ... 312
8.1.4 Section labels ... 313
8.1.5 Text marks ... 314
8.1.6 Separate text ... 318
8.2 Formatting text ... 320
8.2.1 Text markup introduction ... 320
8.2.2 Selecting font and font size ... 324
8.2.3 Text alignment ... 327
8.2.4 Graphic notation inside markup ... 332
8.2.5 Music notation inside markup ... 335
8.3 Fonts ... 337
8.3.1 Unsupported font formats ... 337
8.3.2 Finding fonts ... 338
8.3.3 Font families ... 338
8.3.4 Font features ... 340
8.3.5 Changing fonts ... 341

Specialist notation

9 Vocal music ... 345
9.1 Common notation for vocal music ... 345
9.1.1 References for vocal music ... 345
9.1.2 Entering lyrics ... 346
9.1.3 Aligning lyrics to a melody ... 347
9.1.4 Automatic syllable durations ... 349
9.1.5 Manual syllable durations ... 352
9.1.6 Multiple syllables to one note ... 353
9.1.7 Multiple notes to one syllable ... 354
9.1.8 Extenders and hyphens ... 357
9.1.9 Gradual changes of vowel ... 357
9.2 Techniques specific to lyrics ... 358
9.2.1 Working with lyrics and variables ... 358
9.2.2 Placing lyrics vertically ... 360
9.2.3 Placing syllables horizontally ... 363
9.2.4 Lyrics and repeats ... 366
9.2.5 Divisi lyrics ... 374
9.2.6 Polyphony with shared lyrics ... 376
9.3 Stanzas ... 377
9.3.1 Adding stanza numbers ... 377
9.3.2 Stanza reminders ... 378
9.3.3 Adding dynamics marks to stanzas ... 381
9.3.4 Adding singers' names to stanzas ... 382
9.3.5 Stanzas with different rhythms ... 382
9.3.6 Printing stanzas at the end ... 385
9.3.7 Printing stanzas at the end in multiple columns ... 386
9.4 Songs ... 388
9.4.1 References for songs ... 388
9.4.2 Lead sheets ... 389
9.5 Choral ... 389

9.5.1 References for choral ... 389
9.5.2 Score layouts for choral ... 390
9.6 Opera and stage musicals ... 392
9.6.1 References for opera and stage musicals ... 393
9.6.2 Character names ... 393
9.6.3 Musical cues ... 395
9.6.4 Spoken music ... 399
9.6.5 Dialogue over music ... 399
9.7 Chants psalms and hymns ... 400
9.7.1 References for chants and psalms ... 400
9.7.2 Setting a chant ... 400
9.7.3 Pointing a psalm ... 406
9.7.4 Phrase bar lines in hymn tunes ... 409
9.7.5 Partial measures in hymn tunes ... 410
9.8 Ancient vocal music ... 412

10 Keyboard and other multi-staff instruments ... 413
10.1 Common notation for keyboards ... 413
10.1.1 References for keyboards ... 414
10.1.2 Changing staff manually ... 414
10.1.3 Changing staff automatically ... 416
10.1.4 Staff-change lines ... 418
10.2 Piano ... 421
10.2.1 Piano pedals ... 421
10.3 Organ ... 423
10.3.1 Organ pedal marks ... 423
10.4 Accordion ... 424
10.4.1 Discant symbols ... 424
10.5 Harp ... 425
10.5.1 References for harps ... 425
10.5.2 Harp pedals ... 425

11 Unfretted string instruments ... 427
11.1 Common notation for unfretted strings ... 427
11.1.1 References for unfretted strings ... 427
11.1.2 Bowing indications ... 428
11.1.3 Harmonics ... 428
11.1.4 Snap (Bartók) pizzicato ... 429

12 Fretted string instruments ... 431
12.1 Common notation for fretted strings ... 431
12.1.1 References for fretted strings ... 431
12.1.2 String number indications ... 432
12.1.3 Default tablatures ... 433
12.1.4 Custom tablatures ... 452
12.1.5 Fret diagram markups ... 455
12.1.6 Predefined fret diagrams ... 466
12.1.7 Automatic fret diagrams ... 477
12.1.8 Right-hand fingerings ... 479
12.2 Guitar ... 481

12.2.1 Indicating position and barring 481
12.2.2 Indicating harmonics and dampened notes 482
12.2.3 Indicating power chords 483
12.3 Banjo 484
12.3.1 Banjo tablatures 484
12.4 Lute 485
12.4.1 Lute tablatures 485

13 Percussion 487
13.1 Common notation for percussion 487
13.1.1 References for percussion 487
13.1.2 Basic percussion notation 487
13.1.3 Drum rolls 488
13.1.4 Pitched percussion 488
13.1.5 Percussion staves 489
13.1.6 Custom percussion staves 491
13.1.7 Ghost notes 494

14 Wind instruments 495
14.1 Common notation for wind instruments 495
14.1.1 References for wind instruments 495
14.1.2 Fingerings 496
14.2 Bagpipes 498
14.2.1 Bagpipe definitions 498
14.2.2 Bagpipe example 498
14.3 Woodwinds 500
14.3.1 Woodwind diagrams 500

15 Chord notation 508
15.1 Chord mode 508
15.1.1 Chord mode overview 508
15.1.2 Common chords 509
15.1.3 Extended and altered chords 510
15.1.4 Chord inversions and specific voicings 513
15.2 Displaying chords 513
15.2.1 Printing chord names 513
15.2.2 Customizing chord names 516
15.2.3 Chord grids 522
15.3 Figured bass 526
15.3.1 Introduction to figured bass 526
15.3.2 Entering figured bass 527
15.3.3 Displaying figured bass 531

16 Contemporary music 535
16.1 Pitch and harmony in contemporary music 535
16.1.1 References for pitch and harmony in contemporary music 535
16.1.2 Microtonal notation 535
16.1.3 Contemporary key signatures and harmony 535
16.2 Contemporary approaches to rhythm 535
16.2.1 References for contemporary approaches to rhythm 535

16.2.2 Tuplets in contemporary music 535
16.2.3 Contemporary time signatures 535
16.2.4 Extended polymetric notation 535
16.2.5 Beams in contemporary music 535
16.2.6 Bar lines in contemporary music 535
16.3 Graphical notation 535
16.4 Contemporary scoring techniques 537
16.5 New instrumental techniques 537
16.6 Further reading and scores of interest 537
16.6.1 Books and articles on contemporary musical notation 537
16.6.2 Scores and musical examples 537

17 Ancient notation 538
17.1 Overview of the supported styles 539
17.2 Ancient notation – common features 540
17.2.1 Predefined contexts 540
17.2.2 Ligatures 540
17.2.3 Custodes 541
17.3 Typesetting mensural music 542
17.3.1 Mensural contexts 542
17.3.2 Mensural clefs 543
17.3.3 Mensural time signatures 544
17.3.4 Mensural note heads 544
17.3.5 Mensural flags 545
17.3.6 Mensural rests 546
17.3.7 Mensural accidentals and key signatures 546
17.3.8 Annotational accidentals (*musica ficta*) 547
17.3.9 Historical overview of ligatures 547
17.3.10 White mensural ligatures 549
17.3.11 Reading white mensural ligatures 551
17.4 Typesetting Gregorian chant 554
17.4.1 Gregorian chant contexts 555
17.4.2 Gregorian clefs 555
17.4.3 Gregorian accidentals and key signatures 556
17.4.4 Divisiones 556
17.4.5 Gregorian articulation signs 557
17.4.6 Augmentum dots (*morae*) 558
17.4.7 Gregorian square neume ligatures 558
17.5 Typesetting Kievan square notation 564
17.5.1 Kievan contexts 564
17.5.2 Kievan clefs 565
17.5.3 Kievan notes 565
17.5.4 Kievan accidentals 565
17.5.5 Kievan bar lines 566
17.5.6 Kievan melismata 566
17.6 Working with ancient music – scenarios and solutions 567
17.6.1 Incipits 567
17.6.2 Mensurstriche layout 568
17.6.3 Transcribing Gregorian chant 569
17.6.4 Ancient and modern from one source 572

18 World music ... 575
18.1 Common notation for non-Western music ... 575
18.1.1 Extending notation and tuning systems ... 575
18.2 Arabic music ... 575
18.2.1 References for Arabic music ... 575
18.2.2 Arabic note names ... 576
18.2.3 Arabic key signatures ... 577
18.2.4 Arabic time signatures ... 579
18.2.5 Arabic music example ... 580
18.2.6 Further reading for Arabic music ... 581
18.3 Turkish classical music ... 581
18.3.1 References for Turkish classical music ... 581
18.3.2 Turkish note names ... 581
18.3.3 Turkish key signatures ... 582
18.3.4 Further reading for Turkish music ... 583
18.4 Persian classical music ... 583
18.4.1 Persian music notation ... 583
18.4.2 Persian tunings ... 584
18.4.3 Persian key signatures ... 584
18.4.4 Further reading on Persian music ... 585

General input and output

19 Input modes ... 589

20 Input structure ... 592
20.1 Structure of a score ... 592
20.2 Multiple scores in a book ... 593
20.3 Multiple output files from one input file ... 595
20.4 Output file names ... 595
20.5 File structure ... 596

21 Titles and headers ... 600
21.1 Creating titles, headers, and footers ... 600
21.1.1 Titles explained ... 600
21.1.2 Default layout of bookpart and score titles ... 603
21.1.3 Default layout of headers and footers ... 607
21.2 Custom titles, headers, and footers ... 608
21.2.1 Custom text formatting for titles ... 608
21.2.2 Custom layout for titles ... 608
21.2.3 Custom layout for headers and footers ... 611
21.3 Creating output file metadata ... 612
21.4 Creating footnotes ... 613
21.4.1 Footnotes in music expressions ... 613
21.4.2 Footnotes in stand-alone text ... 619
21.5 Creating in-notes ... 622
21.6 Reference to page numbers ... 623
21.7 Table of contents ... 625

22 Working with input files ... 628
22.1 Including LilyPond files ... 628
22.2 Different editions from one source ... 629
22.2.1 Using variables ... 629
22.2.2 Using tags ... 631
22.2.3 Using global settings ... 641
22.3 Using music functions ... 641
22.3.1 Substitution function syntax ... 642
22.3.2 Substitution function examples ... 642
22.3.3 How to prevent sharing of music expressions ... 644
22.3.4 Substitution functions and relative octave entry ... 646
22.4 Special characters ... 647
22.4.1 Text encoding ... 647
22.4.2 Unicode ... 648
22.4.3 ASCII aliases ... 648

23 Controlling output ... 650
23.1 Extracting fragments of music ... 650
23.2 Skipping corrected music ... 650
23.3 Alternative output formats ... 651
23.3.1 SVG Output ... 651
23.4 Embedding files in PDF output ... 652
23.5 Replacing the notation font ... 652

24 Creating MIDI output ... 654
24.1 Supported notation for MIDI ... 654
24.2 Unsupported notation for MIDI ... 655
24.3 The MIDI block ... 655
24.4 Controlling MIDI dynamics ... 656
24.4.1 Dynamic marks in MIDI ... 656
24.4.2 Setting MIDI volume ... 657
24.4.3 Setting MIDI block properties ... 659
24.5 Using MIDI instruments ... 660
24.6 Using repeats with MIDI ... 661
24.7 MIDI channel mapping ... 661
24.8 Context properties for MIDI effects ... 664
24.9 Enhancing MIDI output ... 665
24.9.1 The `articulate` script ... 665
24.9.2 The `swing` script ... 665

25 Extracting musical information ... 667
25.1 Displaying LilyPond notation ... 667
25.2 Displaying Scheme music expressions ... 667
25.3 Saving music events to a file ... 667

Spacing issues

26 Page layout ... 671

26.1 The \paper block ... 671
26.2 Paper size and automatic scaling ... 672
26.2.1 Setting the paper size ... 672
26.2.2 Automatic scaling to paper size ... 673
26.3 Fixed vertical spacing \paper variables ... 673
26.4 Flexible vertical spacing \paper variables ... 674
26.4.1 Structure of flexible vertical spacing alists ... 674
26.4.2 List of flexible vertical spacing \paper variables ... 675
26.5 Horizontal spacing \paper variables ... 676
26.5.1 \paper variables for widths and margins ... 676
26.5.2 \paper variables for two-sided mode ... 678
26.5.3 \paper variables for shifts and indents ... 678
26.6 Other \paper variables ... 679
26.6.1 \paper variables for line breaking ... 679
26.6.2 \paper variables for page breaking ... 679
26.6.3 \paper variables for page numbering ... 680
26.6.4 \paper variables concerning headers and markups ... 681
26.6.5 \paper variables for debugging ... 682

27 Score layout ... 683

27.1 The \layout block ... 683
27.2 Setting the staff size ... 685

28 Breaks ... 689

28.1 Line breaking ... 689
28.2 Page breaking ... 693
28.2.1 Manual page breaking ... 693
28.2.2 Optimal page breaking ... 694
28.2.3 Minimal page breaking ... 694
28.2.4 One-page page breaking ... 694
28.2.5 One-line page breaking ... 695
28.2.6 One-line-auto-height page breaking ... 695
28.2.7 Optimal page turning ... 695

29 Vertical spacing ... 697

29.1 Flexible vertical spacing within systems ... 697
29.1.1 Within-system spacing properties ... 697
29.1.2 Spacing of ungrouped staves ... 700
29.1.3 Spacing of grouped staves ... 701
29.1.4 Spacing of non-staff lines ... 702
29.2 Explicit staff and system positioning ... 704
29.3 Vertical collision avoidance ... 710

30 Horizontal spacing 713
30.1 Horizontal spacing overview 713
30.2 New spacing section 714
30.3 Changing horizontal spacing globally 715
30.3.1 Uniform stretching of tuplets 716
30.3.2 Strict note spacing 717
30.4 Adjusting horizontal spacing for specific layout objects 717
30.4.1 Overview of object-specific horizontal spacing tweaks 717
30.4.2 Spacing between adjacent non-musical items 718
30.4.3 Spacing between adjacent columns 720
30.5 Line width 722
30.6 Proportional notation 722

31 Fitting music onto fewer pages 729
31.1 Displaying spacing 729
31.2 Changing spacing 730

Changing defaults

32 Tuning output 735

33 Interpretation contexts 736
33.1 Contexts explained 736
33.1.1 Output definitions – blueprints for contexts 736
33.1.2 `Score` – the master of all contexts 736
33.1.3 Top-level contexts – staff containers 737
33.1.4 Intermediate-level contexts – staves 737
33.1.5 Bottom-level contexts – voices 738
33.2 Creating and referencing contexts 739
33.3 Keeping contexts alive 742
33.4 Modifying context plug-ins 745
33.5 Changing context default settings 746
33.5.1 Changing all contexts of the same type 746
33.5.2 Changing just one specific context 749
33.5.3 Order of precedence 751
33.6 Defining new contexts 751
33.7 Context layout order 755

34 Explaining the Internals Reference 757
34.1 Navigating the program reference 757
34.2 Layout interfaces 757
34.3 Determining the grob property 759

35 Modifying properties ... 760
35.1 Overview of modifying properties ... 760
35.2 \set and \unset ... 760
35.3 \override and \revert ... 762
35.4 The \once command ... 763
35.5 \temporary and \undo ... 764
35.6 \set versus \override ... 764
35.7 \tweak and \single ... 764
35.8 The \offset command ... 766
35.9 Modifying alists ... 771

36 Useful concepts and properties ... 774
36.1 Direction and placement ... 774
36.1.1 Articulation direction indicators ... 774
36.1.2 The direction property ... 775
36.2 Distances and measurements ... 775
36.3 Dimensions ... 776
36.4 Spanners ... 776
36.4.1 Modifying broken spanners ... 776
36.4.2 Setting minimum lengths for spanners ... 778
36.4.3 Controlling spanner end points ... 781
36.5 Line styles ... 782
36.6 Line spanners ... 782
36.7 Visibility of objects ... 784
36.7.1 Removing the stencil ... 785
36.7.2 Making objects transparent ... 785
36.7.3 Painting objects white ... 785
36.7.4 Using break-visibility ... 786
36.7.5 Special considerations ... 788
36.8 Rotating objects ... 791
36.8.1 Rotating layout objects ... 791
36.8.2 Rotating markup ... 791
36.9 Aligning objects ... 791
36.9.1 Setting X-offset and Y-offset directly ... 793
36.9.2 Using the side-position-interface ... 793
36.9.3 Using the self-alignment-interface ... 794
36.9.4 Using the break-alignable-interface ... 797
36.10 Modifying stencils ... 800
36.11 Modifying shapes ... 801
36.11.1 Modifying ties and slurs ... 801

Appendices

A Markup commands 809

A.1 Text markup commands 809
A.1.1 Font markup 809
A.1.2 Markup for text alignment 822
A.1.3 Graphical markup 840
A.1.4 Markup for music and musical symbols 851
A.1.5 Conditional markup 864
A.1.6 Instrument-specific markup 864
A.1.7 Accordion registers 869
A.1.8 Other markup commands 874
A.2 Text markup list commands 886

B Notation manual tables 890

B.1 Chord name chart 890
B.2 Common chord modifiers 890
B.3 Predefined string tunings 893
B.4 Predefined fretboard diagrams 894
B.4.1 Diagrams for Guitar 894
B.4.2 Diagrams for Ukulele 896
B.4.3 Diagrams for Mandolin 897
B.5 Predefined paper sizes 899
B.6 MIDI instruments 901
B.7 List of colors 902
B.8 The Emmentaler font 907
B.8.1 Modern glyph charts 907
B.8.2 Ancient glyph charts 916
B.9 Note head styles 922
B.10 Accidental glyph sets 922
B.11 Clef styles 923
B.11.1 Standard clefs 923
B.11.2 Percussion staff clef 924
B.11.3 Tab staff clefs 924
B.11.4 Ancient music clefs 924
B.12 List of special characters 927
B.13 List of articulations 928
B.13.1 Articulation scripts 928
B.13.2 Ornament scripts 928
B.13.3 Fermata scripts 929
B.13.4 Instrument-specific scripts 930
B.13.5 Repeat sign scripts 930
B.13.6 Ancient scripts 930
B.14 List of breath marks 931
B.15 Percussion notes 931
B.16 List of bar lines 933
B.17 Default values for `outside-staff-priority` 937
B.18 Default values for `script-priority` 938
B.19 Grobs and their break-align symbols 939
B.20 Technical glossary 940
B.21 Available music functions 943

B.22 Context modification identifiers 964
B.23 Paper variables 964
B.24 Naming conventions 973
B.25 Predefined type predicates 974
B.25.1 R5RS primary predicates 974
B.25.2 R5RS secondary predicates 974
B.25.3 Guile predicates 975
B.25.4 LilyPond scheme predicates 975
B.25.5 LilyPond exported predicates 976

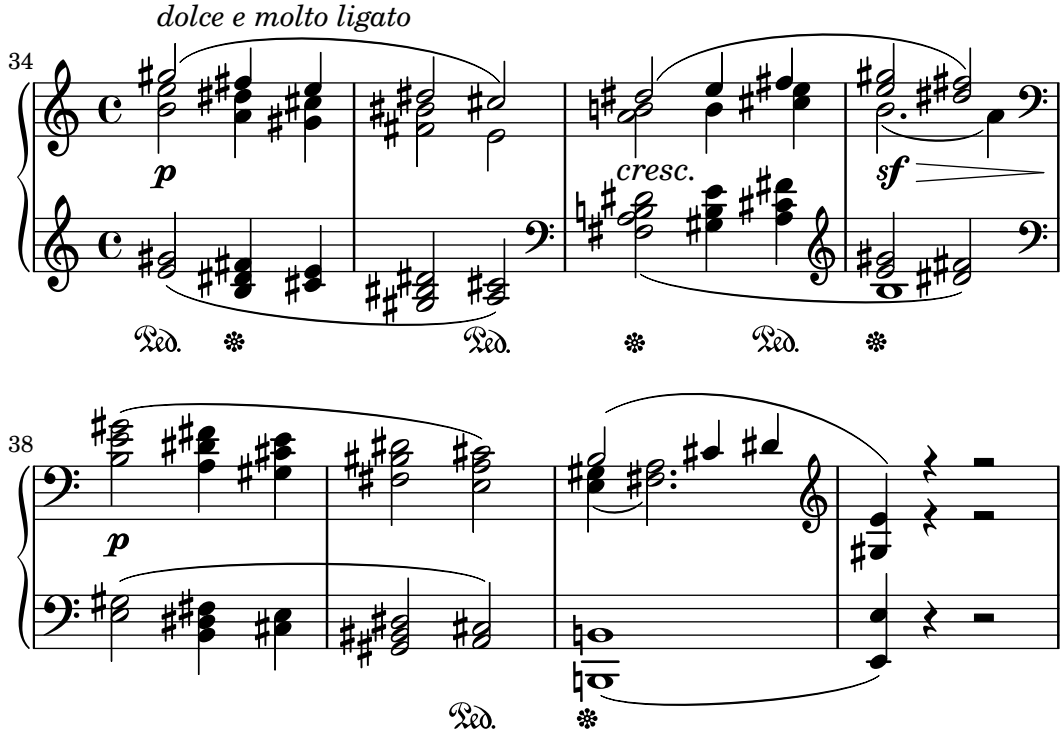
C Cheat sheet 978

D GNU Free Documentation License 981

E Index 988

Musical notation

1 Pitches



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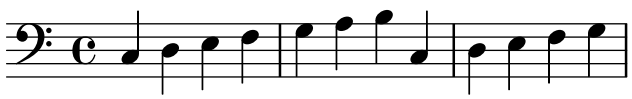
1.1 Writing pitches

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

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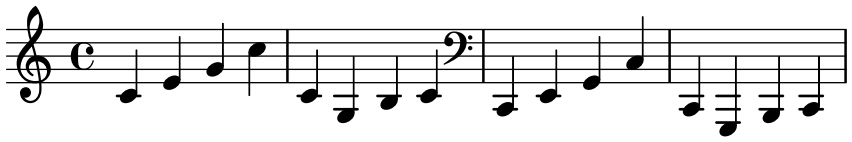
A pitch name is specified using lowercase letters `a` through `g`. The note names `c` to `b` are engraved in the octave below middle C.

```
{
  \clef bass
  c4 d e f
  g4 a b c
  d4 e f g
}
```
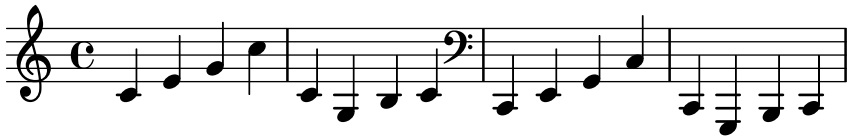


Other octaves may be specified with a single quote (`'`) or comma (`,`) character. Each `'` raises the pitch by one octave; each `,` lowers the pitch by an octave.

```
{
  \clef treble
  c'4 e' g' c''
  c'4 g b c'
  \clef bass
  c,4 e, g, c
  c,4 g,, b,, c,
}
```



Common octave marks can be entered just once on a reference pitch after `\fixed` placed before the music. Pitches inside `\fixed` only need `'` or `,` marks when they are above or below the octave of the reference pitch.

```
{
  \fixed c' {
    \clef treble
    c4 e g c'
    c4 g, b, c
  }
  \clef bass
  \fixed c, {
    c4 e g c'
    c4 g, b, c
  }
}
```



Pitches in the music expression following `\fixed` are unaffected by any enclosing `\relative`, discussed next.

See also

Music Glossary: Section “Pitch names” in *Music Glossary*.

Snippets: Section “Pitches” in *Snippets*.

1.1.2 Relative octave entry

Absolute octave entry requires specifying the octave for every single note. Relative octave entry, in contrast, specifies each octave in relation to the last note: changing one note’s octave will affect all of the following notes.

Relative note mode has to be entered explicitly using the `\relative` command:

`\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. Which choices are meaningful?

  an octave of c
  : Identifying middle C with c' is quite basic, so finding octaves of c tends to be straightforward. If your music starts with gis above c''', you'd write something like \relative c''' { gis' ... }
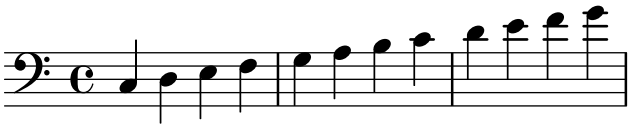
  an octave of the first note inside
  : Writing \relative gis''' { gis ... } makes it easy to determine the absolute pitch of the first note inside.

  no explicit starting pitch
  : The form \relative { gis''' ... } serves as a compact version of the previous option: the first note inside is written in absolute pitch itself. (This happens to be equivalent to choosing f as the reference pitch.)

  The documentation will usually employ the last option.
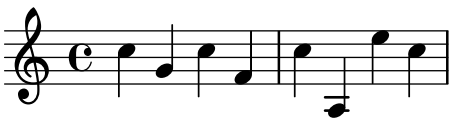
Here is the relative mode shown in action:

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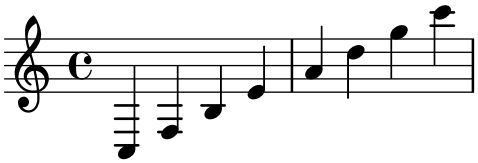


Octave changing marks are used for intervals greater than a fourth:

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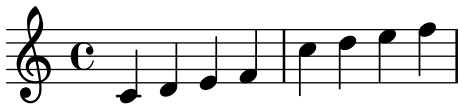


A note sequence without a single octave mark can nevertheless span large intervals:

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



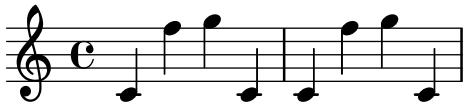
When \relative blocks are nested, the innermost \relative block starts with its own reference pitch independently of the outer \relative.

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



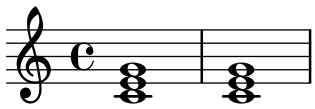
To use absolute mode inside of \relative, put the absolute music inside \fixed c { ... } and the absolute pitches will not affect the octaves of the relative music:

```
\relative {
  c'4 \fixed c { f'' g'' } c |
  c4 \fixed c'' { f g } c
}
```
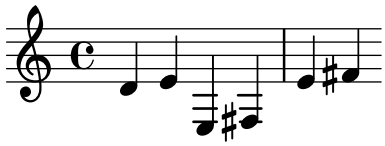


\relative is not allowed inside of \chordmode blocks. \chordmode blocks inside a \relative block remain unchanged.

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



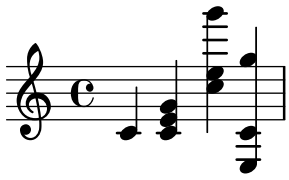
Music inside a \transpose block is absolute unless a \relative is included.

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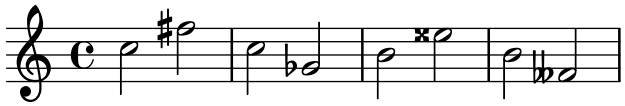


If the preceding item is a chord, the first note of the chord is used as the reference point for the octave placement of a following note or chord. Inside chords, the next note is always relative to the preceding one. Examine the next example carefully, paying attention to the `c` notes.

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



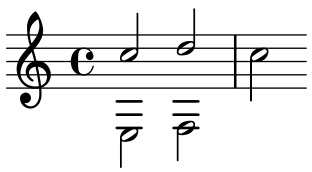
As explained above, the octave of pitches is calculated only with the note names, regardless of any alterations. Therefore, an E-double-sharp following a B will be placed higher, while an F-double-flat will be placed lower. In other words, a double-augmented fourth is considered a smaller interval than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

```
\relative {
  c''2 fis
  c2 ges
  b2 eisis
  b2 feses
}
```



In complex situations, it is sometimes useful to get back to a certain pitch regardless of what happened before. This can be done using `\resetRelativeOctave`:

```
\relative {
  <<
    { c''2 d }
    \\
    { e,,2 f }
  >>
  \resetRelativeOctave c''
  c2
}
```



See also

Music Glossary: Section “fifth” in *Music Glossary*, Section “interval” in *Music Glossary*, Section “Pitch names” in *Music Glossary*.

Notation Reference: Section 1.2.1 [Octave checks], page 12.

Snippets: Section “Pitches” in *Snippets*.
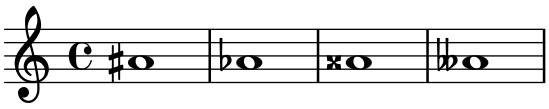
Internals Reference: Section “RelativeOctaveMusic” in *Internals Reference*.

1.1.3 Accidentals

> **Note:** New users are sometimes confused about accidentals and key signatures. In LilyPond, note names specify pitches; key signatures and clefs determine how these pitches are displayed. An unaltered note like `c` means ‘C natural’, regardless of the key signature or clef. For more information, see Section “Pitches and key signatures” in *Learning Manual*.
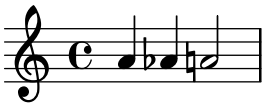
A *sharp* pitch is made by adding `is` to the note name, and a *flat* pitch by adding `es`. As you might expect, a *double sharp* or *double flat* is made by adding `isis` or `eses`. This syntax is derived from Dutch note naming conventions. To use other names for accidentals, see Section 1.1.5 [Note names in other languages], page 10.

```
\relative c'' { ais1 aes aisis aeses }
```
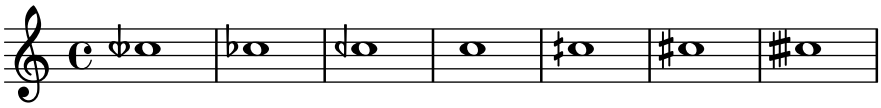


A natural pitch is entered as a simple note name; no suffix is required. A natural sign will be printed when needed to cancel the effect of an earlier accidental or key signature.

```
\relative c'' { a4 aes a2 }
```
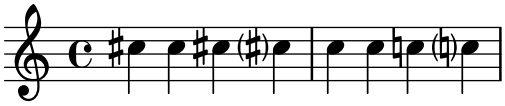


Quarter tones may be added; the following is a series of Cs with increasing pitches:

```
\relative c'' { ceseh1 ces ceh c cih cis cisih }
```



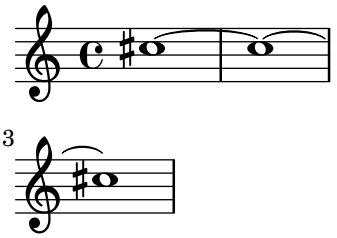
Normally accidentals are printed automatically, but you may also print them manually. A reminder accidental can be forced by adding an exclamation mark `!` after the pitch. A cautionary accidental (i.e., an accidental within parentheses) can be obtained by adding the question mark `?` after the pitch.

```
\relative c'' { cis cis cis! cis? c c c! c? }
```



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

```
\relative c'' {
  cis1~ 1~
  \break
  cis
}
```

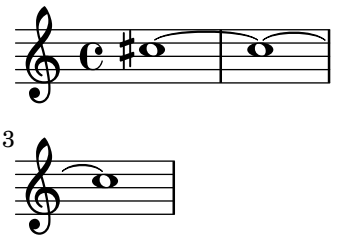


Selected snippets

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

This shows how to hide accidentals on tied notes at the start of a new system.
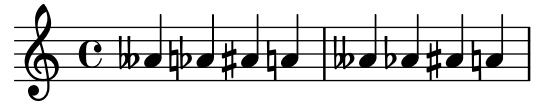
```
\relative c'' {
  \override Accidental.hide-tied-accidental-after-break = ##t
  cis1~ cis~
  \break
  cis
}
```



Preventing extra naturals from being automatically added

In accordance with traditional typesetting rules, a natural sign is printed before a sharp or flat if a previous double sharp or flat on the same note is canceled. To change this behavior to contemporary practice, 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 “Pitches and key signatures” in *Learning Manual*.

Notation Reference: Section 1.3.5 [Automatic accidentals], page 31, Section 17.3.8 [Annotational accidentals (*musica ficta*)], page 547, Section 1.1.5 [Note names in other languages], page 10.

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*.
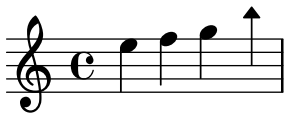
Known issues and warnings

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

1.1.4 Approximate pitches

To write a note when the pitch is not known exactly, such as the highest note a singer can sing, use the `\approximatePitch` command with a pitch that communicates the rough expectation. LilyPond prints the note at the given pitch with a triangular head and no ledger lines.

```
\relative c'' {
  e4 f g \approximatePitch c
}
```



See also
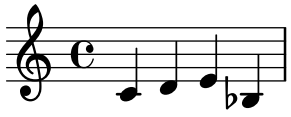
Internals Reference: Section "ApproximatePitchNoteHead" in *Internals Reference*, Section "Note_heads_engraver" in *Internals Reference*.

1.1.5 Note names in other languages

There are predefined sets of note and accidental names for various other languages. Selecting the note name language is usually done at the beginning of the file; the following example is written using Italian note names:

```
\language "italiano"

\relative {
  do' re mi sib
}
```



The available languages and the note names they define are:

Language	Note Names
nederlands	c d e f g a bes b
català or catalan	do re mi fa sol la sib si
deutsch	c d e f g a b h
english	c d e f g a bf/b-flat b
español or espanol	do re mi fa sol la sib si

français	do ré/re mi fa sol la sib si
italiano	do re mi fa sol la sib si
norsk	c d e f g a b h
português or portugues	do re mi fa sol la sib si
suomi	c d e f g a b h
svenska	c d e f g a b h
vlaams	do re mi fa sol la sib si
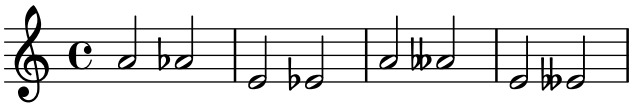
In addition to note names, accidental suffixes may also vary depending on the language:

Language	sharp	flat	double sharp	double flat
nederlands	is	es	isis	eses
català or catalan	d/s	b	dd/ss	bb
deutsch	is	es	isis	eses
english	s/-sharp	f/-flat	ss/x/-sharpsharp	ff/-flatflat
español or espanol	s	b	ss/x	bb
français	d	b	dd/x	bb
italiano	d	b	dd	bb
norsk	iss/is	ess/es	ississ/isis	essess/eses
português or portugues	s	b	ss	bb
suomi	is	es	isis	eses
svenska	iss	ess	ississ	essess
vlaams	k	b	kk	bb

In Dutch, German, Norwegian, and Finnish, `aes` is contracted to `as`; in Dutch and Norwegian, however, both forms are accepted by LilyPond. Exactly the same holds for `es` and `ees`, `aeses` and `ases`, and finally `eeses` and `eses`.

In German and Finnish, LilyPond additionally provides the more frequent form `asas` for `ases`.

```
\relative c'' { a2 as e es a ases e eses }
```



Some music uses microtones whose alterations are fractions of a 'normal' sharp or flat. The following table lists note name suffixes for quarter tone accidentals; here the prefixes *semi-* and *sesqui-* respectively mean 'half' and 'one and a half'.
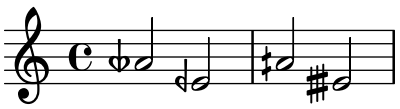
Language	semi-sharp	semi-flat	sesqui-sharp	sesqui-flat
nederlands	ih	eh	isih	eseh
català or catalan	qd/qs	qb	tqd/tqs	tqb
deutsch	ih	eh	isih	eseh
english	qs	qf	tqs	tqf
español or espanol	cs	cb	tcs	tcb
français	sd	sb	dsd	bsb
italiano	sd	sb	dsd	bsb
norsk	ih	eh	issih/isih	esseh/eseh

português or portugues	sqt	bqt	stqt	btqt
suomi	ih	eh	isih	eseh
svenska	ih	eh	issih	esseh
vlaams	hk	hb	khk	bhb

In German, there are similar name contractions for microtones as with normal pitches described above.

```
\language "deutsch"

\relative c'' { asah2 eh aih eisih }
```



Most languages presented here are commonly associated with Western classical music, also referred to as *Common Practice Period*. However, alternate pitches and tuning systems are also supported: see Section 18.1 [Common notation for non-Western music], page 575.

See also

Music Glossary: Section “Pitch names” in *Music Glossary*, Section “Common Practice Period” in *Music Glossary*.

Notation Reference: Section 18.1 [Common notation for non-Western music], page 575.

Installed Files: `scm/define-note-names.scm`.

Snippets: Section “Pitches” in *Snippets*.

1.2 Changing multiple pitches
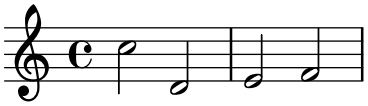
This section discusses how to modify pitches.

1.2.1 Octave checks

In relative mode, it is easy to forget an octave changing mark. Octave checks make such errors easier to find by displaying a warning and correcting the octave if a note is found in an unexpected octave.
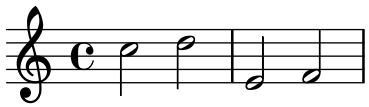
To check the octave of a note, specify the absolute octave after the `=` symbol. This example will generate a warning (and change the pitch) because the second note is the absolute octave `d''` instead of `d'` as indicated by the octave correction.

```
\relative {
  c''2 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. While the previous note itself is not changed, future notes are relative to the corrected value.
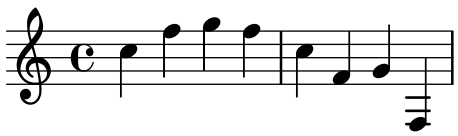
```
\relative {
  c''2 d
  \octaveCheck c'
  e2 f
}
```



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

```
\relative {
  c''4 f g f

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



See also

Snippets: Section "Pitches" in *Snippets*.

Internals Reference: Section "RelativeOctaveCheck" in *Internals Reference*.

1.2.2 Transpose

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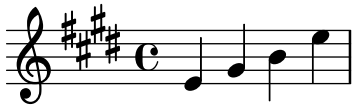
```
\transpose frompitch topitch musicexpr
```

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

> **Note:** Music inside a \transpose block is absolute unless a \relative is included in the block.
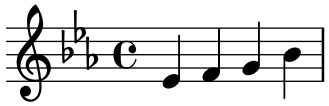
Consider a piece written in the key of D-major. It can be transposed up to E-major; note that the key signature is automatically transposed as well.

```
\transpose d e {
  \relative {
    \key d \major
    d'4 fis a d
  }
}
```



If a part written in C (normal *concert pitch*) is to be played on the A clarinet (for which an A is notated as a C and thus sounds a minor third lower than notated), the appropriate part will be produced with:
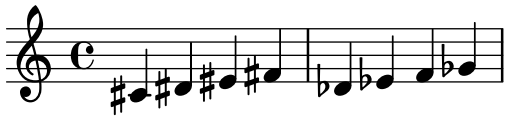
```
\transpose a c' {
  \relative {
    \key c \major
    c'4 d e g
  }
}
```



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' 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 Section 1.3.4 [Instrument transpositions], page 30.

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))
         ;; `ly:pitch-alteration` returns quarter tone steps.
         (a (* 4 (ly:pitch-alteration p)))
         (n (ly:pitch-notename p)))
     (cond
      ((and (> a 1)
            (or (eqv? n 6) (eqv? n 2)))
       (set! a (- a 2))
       (set! n (+ n 1)))
      ((and (< a -1)
            (or (eqv? n 0) (eqv? 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))))
     (when (< n 0)
       (set! o (- o 1))
       (set! n (+ n 7)))
     (when (> n 6)
       (set! o (+ o 1))
       (set! n (- n 7)))
     (ly:make-pitch o n (/ a 4))))
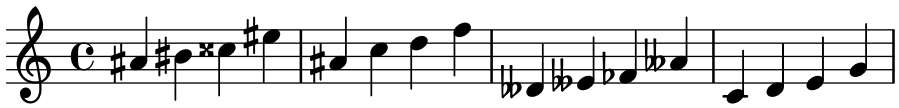
#(define (naturalize music)
   (let ((es (ly:music-property music 'elements))
         (e (ly:music-property music 'element))
         (p (ly:music-property music 'pitch)))
     (when (pair? es)
       (ly:music-set-property! music 'elements
                               (map naturalize es)))
     (when (ly:music? e)
       (ly:music-set-property! music 'element
                               (naturalize e)))
```

```
      (when (ly:pitch? p)
        (set! p (naturalize-pitch p))
        (ly:music-set-property! music 'pitch p))
      music))

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

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

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



See also

Notation Reference: Section 1.3.4 [Instrument transpositions], page 30, Section 1.2.3 [Inversion], page 16, Section 1.2.5 [Modal transformations], page 17, Section 1.1.2 [Relative octave entry], page 4, Section 1.2.4 [Retrograde], page 17.

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`.

Triple accidentals will not be printed if using `\transpose`. An 'enharmonically equivalent' pitch will be used instead (e.g., d-flat rather than e-triple-flat).
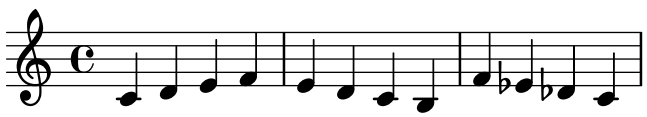
1.2.3 Inversion

A music expression can be inverted and transposed in a single operation with:

`\inversion` *`around-pitch to-pitch musicexpr`*

The *`musicexpr`* is inverted interval by interval around *`around-pitch`*, and then transposed so that *`around-pitch`* is mapped to *`to-pitch`*.

```
music = \relative { c' d e f }
\new Staff {
  \music
  \inversion d' d' \music
  \inversion d' ees' \music
}
```



> **Note:** Motifs to be inverted should be expressed in absolute form or be first converted to absolute form by enclosing them in a `\relative` block.

See also
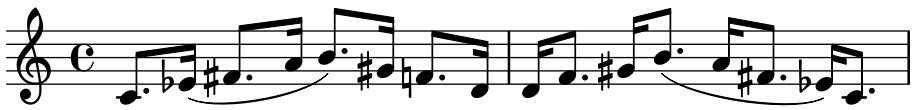
Notation Reference: Section 1.2.5 [Modal transformations], page 17, Section 1.2.4 [Retrograde], page 17, Section 1.2.2 [Transpose], page 13.

1.2.4 Retrograde

A music expression can be reversed to produce its retrograde:

```
music = \relative { c'8. ees16( fis8. a16 b8.) gis16 f8. d16 }

\new Staff {
  \music
  \retrograde \music
}
```



Known issues and warnings

`\retrograde` is a rather simple tool. Since many events are 'mirrored' rather than exchanged, tweaks and directional modifiers for opening spanners need to be added at the matching closing spanners: `^(` needs to be ended by `^)`, every `\<` or `\cresc` needs to be ended by `\!` or `\endcr`, every `\>` or `\decr` needs to be ended by `\enddecr`. Property-changing commands/overrides with a lasting effect will likely cause surprises.

See also

Notation Reference: Section 1.2.3 [Inversion], page 16, Section 1.2.5 [Modal transformations], page 17, Section 1.2.2 [Transpose], page 13.

1.2.5 Modal transformations

In a musical composition that is based on a scale, a motif is frequently transformed in various ways. It may be *transposed* to start at different places in the scale or it may be *inverted* around a pivot point in the scale. It may also be reversed to produce its *retrograde*, see Section 1.2.4 [Retrograde], page 17.

> **Note:** Any note that does not lie within the given scale will be left untransformed.

Modal transposition
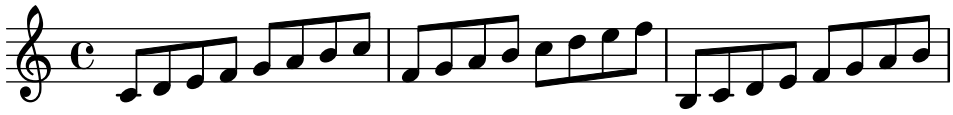
A motif can be transposed within a given scale with:

`\modalTranspose` *from-pitch to-pitch scale motif*

The notes of *motif* are shifted within the *scale* by the number of scale degrees given by the interval between *to-pitch* and *from-pitch*:

```
diatonicScale = \relative { c' d e f g a b }
motif = \relative { c'8 d e f g a b c }
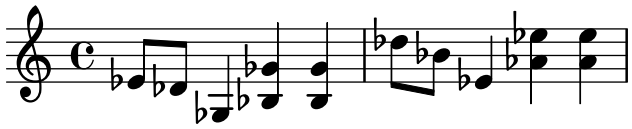
\new Staff {
  \motif
  \modalTranspose c f \diatonicScale \motif
  \modalTranspose c b, \diatonicScale \motif
}
```



An ascending scale of any length and with any intervals may be specified:

```
pentatonicScale = \relative { ges aes bes des ees }
motif = \relative { ees'8 des ges,4 <ges' bes,> <ges bes,> }
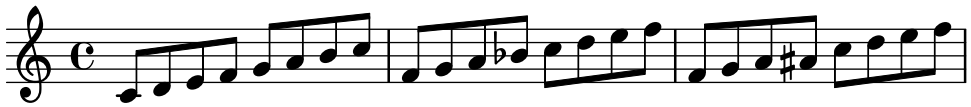
\new Staff {
  \motif
  \modalTranspose ges ees' \pentatonicScale \motif
}
```



When used with a chromatic scale \modalTranspose has a similar effect to \transpose, but with the ability to specify the names of the notes to be used:

```
chromaticScale = \relative { c' cis d dis e f fis g gis a ais b }
motif = \relative { c'8 d e f g a b c }

\new Staff {
  \motif
  \transpose c f \motif
  \modalTranspose c f \chromaticScale \motif
}
```



Modal inversion

A motif can be inverted within a given scale around a given pivot note and transposed in a single operation with:
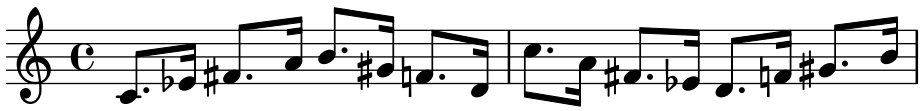
\modalInversion *around-pitch to-pitch scale motif*

The notes of *motif* are placed the same number of scale degrees from the *around-pitch* note within the *scale*, but in the opposite direction, and the result is then shifted within the *scale* by the number of scale degrees given by the interval between *to-pitch* and *around-pitch*.

So to simply invert around a note in the scale use the same value for *around-pitch* and *to-pitch*:

```
octatonicScale = \relative { ees' f fis gis a b c d }
motif = \relative { c'8. ees16 fis8. a16 b8. gis16 f8. d16 }
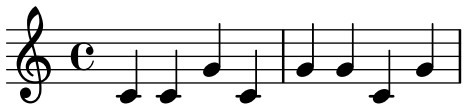
\new Staff {
  \motif
  \modalInversion fis' fis' \octatonicScale \motif
}
```



To invert around a pivot between two notes in the scale, invert around one of the notes and then transpose by one scale degree. The two notes specified can be interpreted as bracketing the pivot point:

```
scale = \relative { c' g' }
motive = \relative { c' c g' c, }
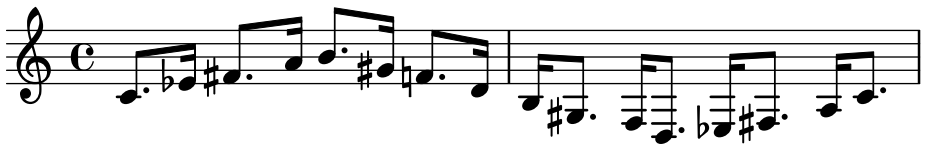
\new Staff {
  \motive
  \modalInversion c' g' \scale \motive
}
```



The combined operation of inversion and retrograde produce the retrograde inversion:

```
octatonicScale = \relative { ees' f fis gis a b c d }
motif = \relative { c'8. ees16 fis8. a16 b8. gis16 f8. d16 }

\new Staff {
  \motif
  \retrograde \modalInversion c' c' \octatonicScale \motif
}
```



See also

Notation Reference: Section 1.2.3 [Inversion], page 16, Section 1.2.4 [Retrograde], page 17, Section 1.2.2 [Transpose], page 13.
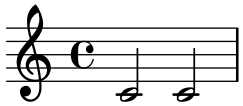
1.3 Displaying pitches

This section discusses how to alter the output of pitches.

1.3.1 Clef
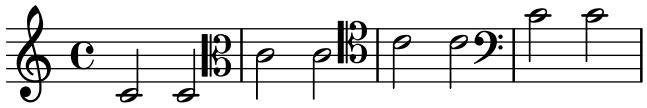
Without any explicit command, the default clef for LilyPond is the treble (or *G*) clef.

```
c'2 c'
```



However, the clef can be changed by using the `\clef` command and an appropriate clef name. *Middle C* is shown in each of the following examples.

```
\clef treble
c'2 c'
\clef alto
c'2 c'
\clef tenor
c'2 c'
\clef bass
c'2 c'
```
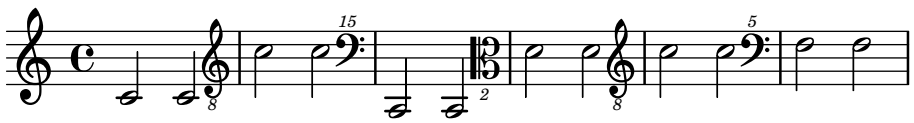


For the full range of possible clef names see Section B.11 [Clef styles], page 923.

Specialized clefs, such as those used in *Ancient* music, are described in Section 17.3.2 [Mensural clefs], page 543, and Section 17.4.2 [Gregorian clefs], page 555. Music that requires tablature clefs is discussed in Section 12.1.3 [Default tablatures], page 433, and Section 12.1.4 [Custom tablatures], page 452.

For mixing clefs when using cue notes, see the `\cueClef` and `\cueDuringWithClef` commands in Section 6.3.3 [Formatting cue notes], page 269.
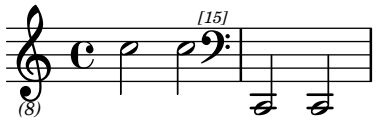
By adding `_8` or `^8` to the clef name, the clef is transposed one octave down or up respectively, and `_15` and `^15` transpose by two octaves. Other integers can be used if required. Clef names containing non-alphabetic characters must be enclosed in quotes

```
\clef treble
c'2 c'
\clef "treble_8"
c'2 c'
\clef "bass^15"
c'2 c'
\clef "alto_2"
c'2 c'
\clef "G_8"
c'2 c'
\clef "F^5"
c'2 c'
```



Optional octavation can be obtained by enclosing the numeric argument in parentheses or brackets:

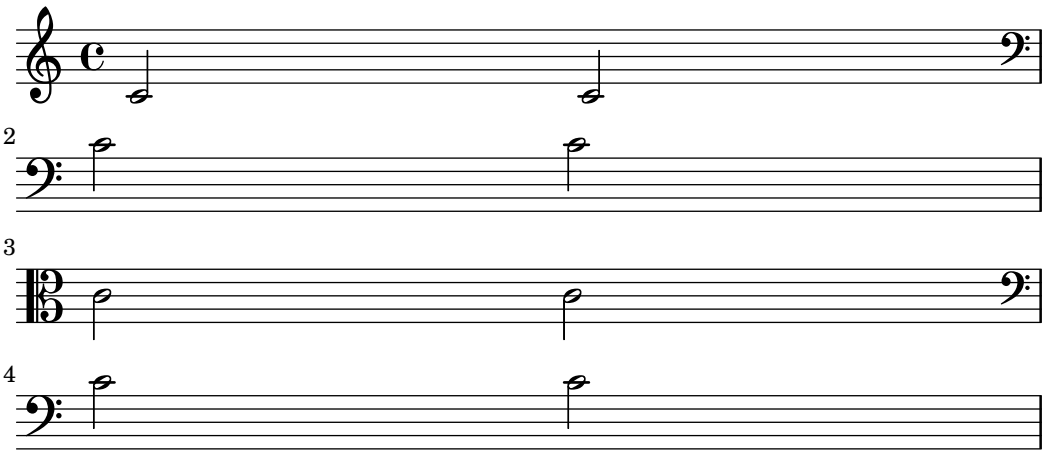
```
\clef "treble_(8)"
c'2 c'
\clef "bass^[15]"
c'2 c'
```



The pitches are displayed as if the numeric argument were given without parentheses/ brackets.
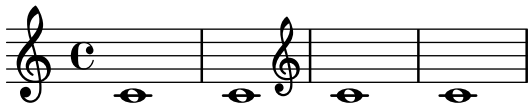
By default, a clef change taking place at a line break causes the new clef symbol to be printed at the end of the previous line, as a 'warning clef', as well as at the beginning of the next. This warning clef can be suppressed.

```
\clef treble c'2 c' \break
\clef bass c'2 c' \break
\set Staff.explicitClefVisibility = #end-of-line-invisible
\clef alto c'2 c' \break
\unset Staff.explicitClefVisibility
\clef bass c'2 c'
```



By default, a clef that has previously been printed will not be reprinted if the same `\clef` command is issued again and will be ignored. The command `\set Staff.forceClef = ##t` changes this behavior.

```
\clef treble
c'1
\clef treble
c'1
\set Staff.forceClef = ##t
c'1
\clef treble
c'1
```

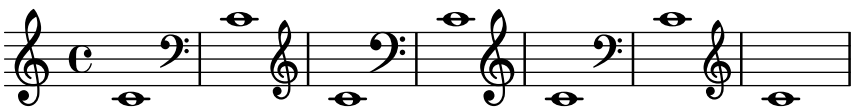


To be more precise, it is not the `\clef` command itself that prints a clef. Instead, it sets or changes a property of the `Clef_engraver`, which then decides by its own whether to display a

clef or not in the current staff. The `forceClef` property overrides this decision locally to reprint a clef once.

When there is a manual clef change, the glyph of the changed clef will be smaller than normal. This behavior can be overridden.

```
\clef "treble"
c'1
\clef "bass"
c'1
\clef "treble"
c'1
\override Staff.Clef.full-size-change = ##t
\clef "bass"
c'1
\clef "treble"
c'1
\revert Staff.Clef.full-size-change
\clef "bass"
c'1
\clef "treble"
c'1
```



Selected snippets

Tweaking clef properties

Changing the clef glyph, its position, or the ottavation does not change the position of subsequent notes on the staff. To get key signatures on their correct staff lines, `middleCClefPosition` must also be specified, with positive or negative values moving "middle C" up or down respectively, relative to the staff's center line.

For example, `\clef "treble_8"` is equivalent to setting the context properties `clefGlyph`, `clefPosition` (the vertical position of the clef itself on the staff), `middleCPosition`, and `clefTransposition`. Note that when any of these properties (except `middleCPosition`) are changed a new clef symbol is printed.

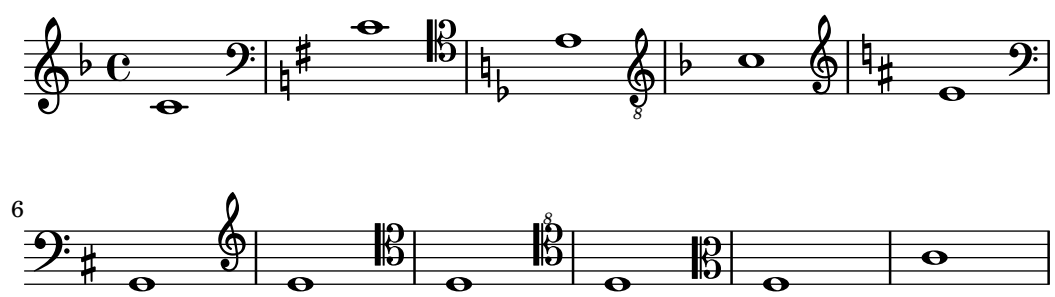
The following examples show the possibilities when setting these properties manually. On the first line, the manual changes preserve the standard relative positioning of clefs and notes, whereas on the second line, they do not.

```
{
  % The default treble clef.
  \key f \major
  c'1
  % The standard bass clef
  \set Staff.clefGlyph = "clefs.F"
  \set Staff.clefPosition = 2
  \set Staff.middleCPosition = 6
  \set Staff.middleCClefPosition = 6
  \key g \major
  c'1
  % The baritone clef.
```

```
  \set Staff.clefGlyph = "clefs.C"
  \set Staff.clefPosition = 4
  \set Staff.middleCPosition = 4
  \set Staff.middleCClefPosition = 4
  \key f \major
  c'1
  % The standard choral tenor clef.
  \set Staff.clefGlyph = "clefs.G"
  \set Staff.clefPosition = -2
  \set Staff.clefTransposition = -7
  \set Staff.middleCPosition = 1
  \set Staff.middleCClefPosition = 1
  \key f \major
  c'1
  % A non-standard clef.
  \set Staff.clefPosition = 0
  \set Staff.clefTransposition = 0
  \set Staff.middleCPosition = -4
  \set Staff.middleCClefPosition = -4
  \key g \major
  c'1 \break

  % The following clef changes do not preserve
  % the normal relationship between notes, key signatures
  % 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.clefTransposition = 7
  c'1
  \set Staff.clefTransposition = 0
  \set Staff.clefPosition = 0
  c'1

  % Return to the normal clef.
  \set Staff.middleCPosition = 0
  c'1
}
```



See also

Notation Reference: Section 17.3.2 [Mensural clefs], page 543, Section 17.4.2 [Gregorian clefs], page 555, Section 12.1.3 [Default tablatures], page 433, Section 12.1.4 [Custom tablatures], page 452, Section 6.3.3 [Formatting cue notes], page 269.

Installed Files: `scm/parser-clef.scm`.

Snippets: Section "Pitches" in *Snippets*.
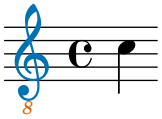
Internals Reference: Section "Clef_engraver" in *Internals Reference*, Section "Clef" in *Internals Reference*, Section "ClefModifier" in *Internals Reference*, Section "clef-interface" in *Internals Reference*.

Known issues and warnings

Ottavation numbers attached to clefs are treated as separate grobs. So any `\override` done to the *Clef* will also need to be applied, as a separate `\override`, to the *ClefModifier* grob.

```
\new Staff \with {
  \override Clef.color = #(universal-color 'blue)
  \override ClefModifier.color = #(universal-color 'vermillion)
}

\clef "treble_8" c'4
```



1.3.2 Key signature
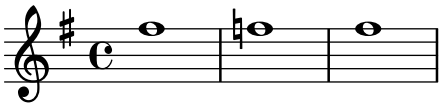
> **Note:** New users are sometimes confused about accidentals and key signatures. In LilyPond, note names are the raw input; key signatures and clefs determine how this raw input is displayed. An unaltered note like `c` means 'C natural', regardless of the key signature or clef. For more information, see Section "Pitches 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`.
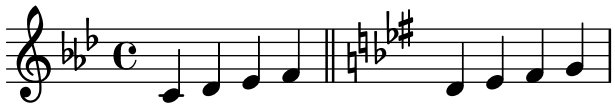
```
\relative {
  \key g \major
  fis''1
  f
  fis
}
```



Additional modes can be defined, by listing the alterations for each scale step when the mode starts on C.

```
freygish = #`((0 . ,NATURAL) (1 . ,FLAT) (2 . ,NATURAL)
    (3 . ,NATURAL) (4 . ,NATURAL) (5 . ,FLAT) (6 . ,FLAT))
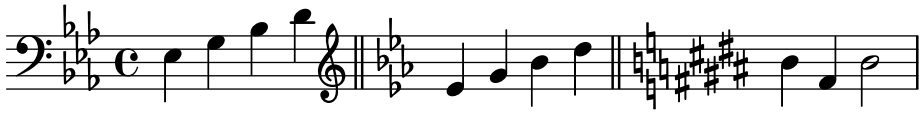
\relative {
  \key c \freygish c'4 des e f
  \bar "||" \key d \freygish d es fis g
}
```



Accidentals in the key signature may be printed in octaves other than their traditional positions, or in multiple octaves, by using the `flat-positions` and `sharp-positions` properties of `KeySignature`. Entries in these properties specify the range of staff positions where accidentals will be printed. If a single position is specified in an entry, the accidentals are placed within the octave ending at that staff position.

```
\override Staff.KeySignature.flat-positions = #'((-5 . 5))
\override Staff.KeyCancellation.flat-positions = #'((-5 . 5))
\clef bass \key es \major es g bes d'
\clef treble \bar "||" \key es \major es' g' bes' d''

\override Staff.KeySignature.sharp-positions = #'(2)
\bar "||" \key b \major b' fis' b'2
```
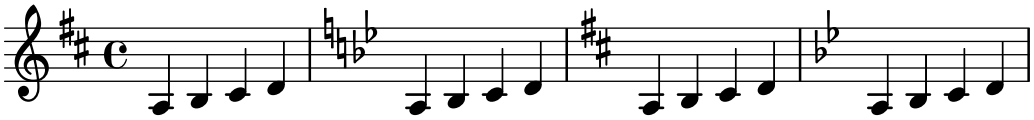


Selected snippets

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 the `printKeyCancellation` property to `#f` 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 `keyAlterations` 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.keyAlterations =
  #`(((octave . step) . alter) ((octave . step) . alter) ...)
```
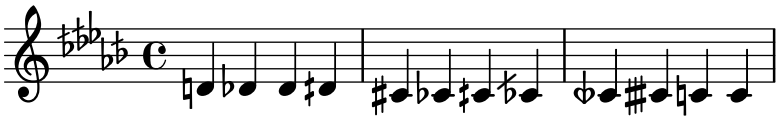
where, for each element in the list, *octave* specifies the octave (0 being the octave from middle C to the B above), *step* specifies the note within the octave (0 means C and 6 means B), and *alter* is one of `SHARP`, `FLAT`, `DOUBLE-SHARP`, etc., preceded by a comma.

Alternatively, you can use the more concise format `(step . alter)` for each item in the list if the same alterations are used in all octaves.

For microtonal scales where a "sharp" is not 100 cents, *alter* refers to the alteration as a proportion of a 200-cent whole tone.

```
\include "arabic.ly"

\relative do' {
  \set Staff.keyAlterations = #`((0 . ,SEMI-FLAT)
                                 (1 . ,SEMI-FLAT)
                                 (2 . ,FLAT)
                                 (5 . ,FLAT)
                                 (6 . ,SEMI-FLAT))
  % \set Staff.extraNatural = ##f
  re reb \dwn reb resd
  dod dob dosd \dwn dob |
  dobsb dodsd do do |
}
```



See also

Music Glossary: Section "church mode" in *Music Glossary*, Section "scordatura" in *Music Glossary*.

Learning Manual: Section "Pitches and key signatures" in *Learning Manual*.

Snippets: Section "Pitches" in *Snippets*.

Internals Reference: Section "KeyChangeEvent" in *Internals Reference*, Section "Key_engraver" in *Internals Reference*, Section "Key_performer" in *Internals Reference*, Section "KeyCancellation" in *Internals Reference*, Section "KeySignature" in *Internals Reference*, Section "key-signature-interface" in *Internals Reference*.
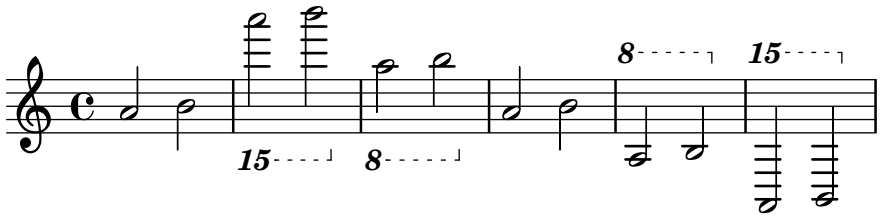
1.3.3 Ottava brackets

Ottava brackets raise or lower displayed notes by one or more octaves, leaving the sounding pitch unchanged. The purpose of the octavation is to reduce the use of ledger lines, especially when wide intervals occur in short passages. The `\ottava` takes a positive integer to indicate

playing higher than printed, or a negative to play lower. The effect of the ottava brackets lasts to the next entry of a different octavation, and can be ended by using `\ottava 0`.

In the following example, all notes sound at the same pitch:

```
\relative c'' {
  a2 b
  \ottava -2
  a2 b
  \ottava -1
  a2 b
  \ottava 0
  a2 b
  \ottava 1
  a2 b
  \ottava 2
  a2 b
}
```
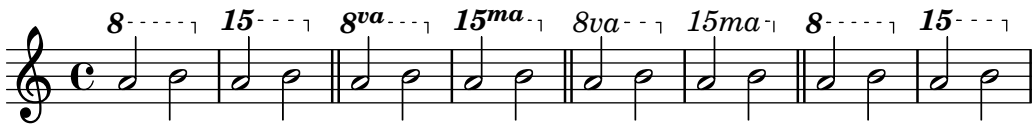


By default, only a number is printed at the start of the bracket. That setting may be changed to include an abbreviated ordinal, either in superscript or in normal letters; the initial bold font weight of these characters may also be altered, as explained in Section 8.2.2 [Selecting font and font size], page 324.

The following example demonstrates various options, as well as how to go back to the current default behavior:

```
\relative c'' {
  \ottava 1
  a'2 b
  \ottava 2
  a'2 b
  \bar "||"
  \set Staff.ottavationMarkups = #ottavation-ordinals
  \ottava 1
  a,2 b
  \ottava 2
  a'2 b
  \bar "||"
  \override Staff.OttavaBracket.font-series = #'normal
  \set Staff.ottavationMarkups = #ottavation-simple-ordinals
  \ottava 1
  a,2 b
  \ottava 2
  a'2 b
  \bar "||"
  \revert Staff.OttavaBracket.font-series
  \set Staff.ottavationMarkups = #ottavation-numbers
  \ottava 1
```

```
  a,2 b
  \ottava 2
  a'2 b
}
```



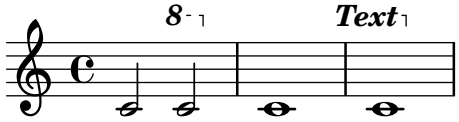
Selected snippets

Changing ottava text

Internally, `\ottava` sets the properties `ottavation` (for example, to `8va` or `8vb`) and `middleCPosition`. To override the text of the bracket, set `ottavation` after invoking `\ottava`.

Short text is especially useful when a brief ottava is used.

```
{
  c'2
  \ottava 1
  \set Staff.ottavation = "8"
  c''2
  \ottava 0
  c'1
  \ottava 1
  \set Staff.ottavation = "Text"
  c''1
}
```



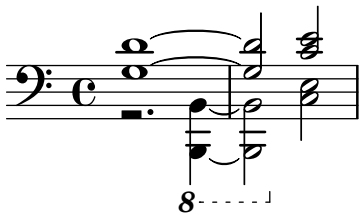
Adding an ottava marking to a single voice

If you have more than one voice on the staff, setting octavation in one voice transposes the position of notes in all voices for the duration of the ottava bracket. If the octavation is only intended to apply to one voice, the `Ottava_spanner_engraver` should be moved to `Voice` context.

```
\layout {
  \context {
    \Staff
    \remove Ottava_spanner_engraver
  }
  \context {
    \Voice
    \consists Ottava_spanner_engraver
  }
}

{
  \clef bass
  << { <g d'>1~ q2 <c' e'> }
```

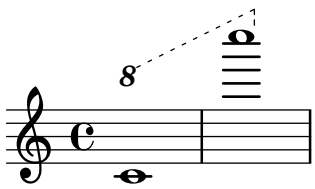
```
  \\
    {
      r2.
      \ottava -1
      <b,,, b,,>4 ~ |
      q2
      \ottava 0
      <c e>2
    }
  >>
}
```



Modifying the ottava spanner slope

It is possible to change the slope of the ottava spanner.

```
\relative c'' {
  \override Staff.OttavaBracket.stencil = #ly:line-spanner::print
  \override Staff.OttavaBracket.bound-details =
    #`((left . ((Y . 0)
                (attach-dir . ,LEFT)
                (padding . 0)
                (stencil-align-dir-y . ,CENTER)))
       (right . ((Y . 5.0) ; Change the number here
                 (padding . 0)
                 (attach-dir . ,RIGHT)
                 (text . ,(make-draw-dashed-line-markup
                           (cons 0 -1.2))))))
  \override Staff.OttavaBracket.left-bound-info =
     #ly:horizontal-line-spanner::calc-left-bound-info-and-text
  \override Staff.OttavaBracket.right-bound-info =
     #ly:horizontal-line-spanner::calc-right-bound-info
  \ottava 1
  c1
  c'''1
}
```



See also

Music Glossary: Section “octavation” in *Music Glossary*.

Notation Reference: Section 8.2.2 [Selecting font and font size], page 324.

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*.

1.3.4 Instrument transpositions
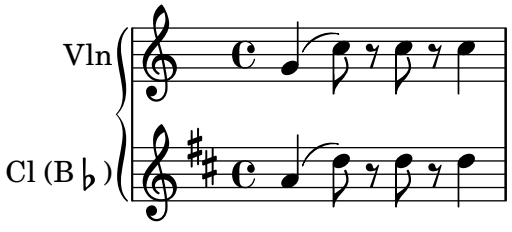
When typesetting scores that involve transposing instruments, some parts can be typeset in a different pitch than the *concert pitch*. In these cases, the key of the *transposing instrument* should be specified; otherwise the MIDI output and cues in other parts will produce incorrect pitches. For more information about quotations, see Section 6.3.2 [Quoting other voices], page 266.

`\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.

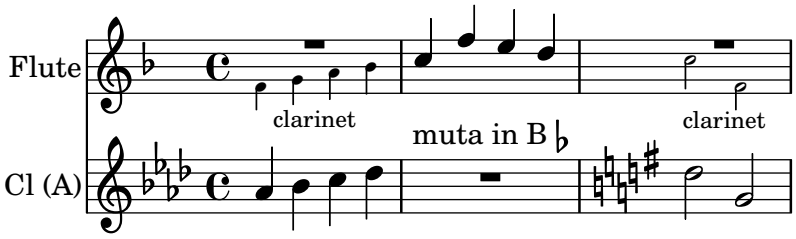
Here are a few notes for violin and B-flat clarinet where the parts have been entered using the notes and key as they appear in each part of the conductor's score. The two instruments are playing in unison.

```
\new GrandStaff <<
  \new Staff = "violin" \with {
    instrumentName = "Vln"
    midiInstrument = "violin"
  }
  \relative c'' {
    % not strictly necessary, but a good reminder
    \transposition c'
    \key c \major
    g4( c8) r c r c4
  }
  \new Staff = "clarinet" \with {
    instrumentName = \markup { Cl (B\flat) }
    midiInstrument = "clarinet"
  }
  \relative c'' {
    \transposition bes
    \key d \major
    a4( d8) r d r d4
  }
>>
```



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

```
flute = \relative c'' {
  \key f \major
  \cueDuring "clarinet" #DOWN {
    R1 _\markup\tiny "clarinet"
    c4 f e d
    R1 _\markup\tiny "clarinet"
  }
}
clarinet = \relative c'' {
  \key aes \major
  \transposition a
  aes4 bes c des
  R1^\markup { muta in B\flat }
  \key g \major
  \transposition bes
  d2 g,
}
\addQuote "clarinet" \clarinet
<<
  \new Staff \with { instrumentName = "Flute" }
    \flute
  \new Staff \with { instrumentName = "Cl (A)" }
    \clarinet
>>
```



See also

Music Glossary: Section "concert pitch" in *Music Glossary*, Section "transposing instrument" in *Music Glossary*.

Notation Reference: Section 6.3.2 [Quoting other voices], page 266, Section 1.2.2 [Transpose], page 13.

Snippets: Section "Pitches" in *Snippets*.

1.3.5 Automatic accidentals

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

```
\new Staff <<
  \accidentalStyle voice
  { ... }
>>
```

The accidental style normally applies to the current `Staff` (with the exception of the styles `choral`, `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:

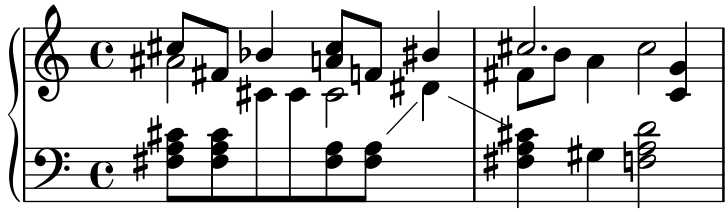
```
\accidentalStyle StaffGroup.voice
```

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

```
musicA = {
  <<
    \relative {
      cis''8 fis, bes4 <a cis>8 f bis4 |
      cis2. <c, g'>4 |
    }
    \\
    \relative {
      ais'2 cis, |
      fis8 b a4 cis2 |
    }
  >>
}

musicB = {
  \clef bass
  \new Voice {
    \voiceTwo \relative {
      <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 {
  <<
    \new Staff = "up" {
      \accidentalStyle default
      \musicA
    }
    \new Staff = "down" {
      \accidentalStyle default
      \musicB
    }
  >>
}
```

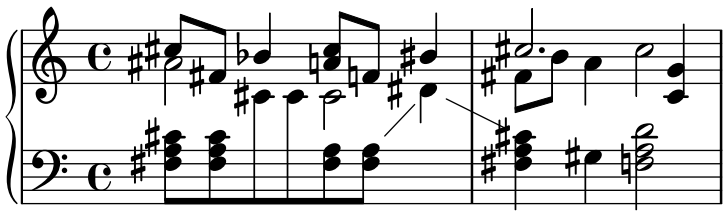


Note that the last lines of this example can be replaced by the following, as long as the same accidental style should be used in both staves.

```
\new PianoStaff {
  <<
    \new Staff = "up" {
      %%% change the next line as desired:
      \accidentalStyle Score.default
      \musicA
    }
    \new Staff = "down" {
      \musicB
    }
  >>
}
```

`default`

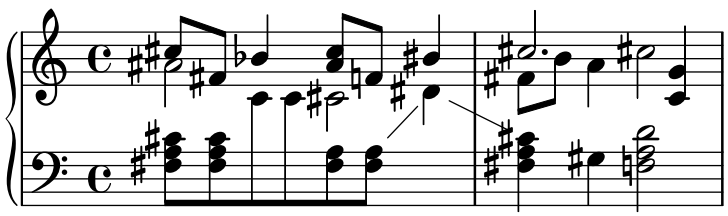
This is the default typesetting behavior. It corresponds to eighteenth-century common practice: accidentals are remembered to the end of the measure in which they occur and only in their own octave. Thus, in the example below, no natural signs are printed before the `b` in the second measure or the last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



`voice`

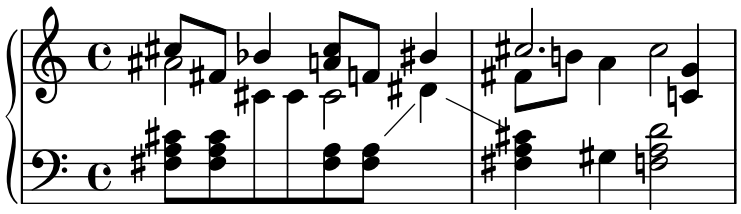
The normal behavior is to remember the accidentals at `Staff`-level. In this style, however, accidentals are typeset individually for each voice. Apart from that, the rule is similar to `default`.

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.



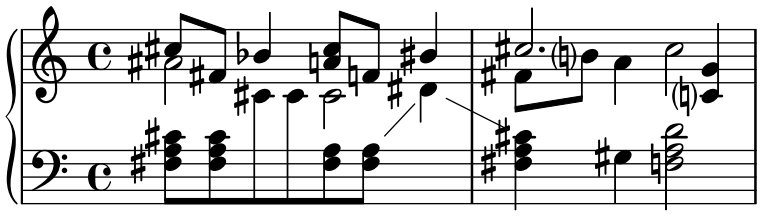
`modern`

This rule corresponds to the common practice in the twentieth century. It omits some extra natural signs, which were traditionally prefixed to a sharp following a double sharp, or a flat following a double flat. The `modern` rule prints the same accidentals as `default`, with two additions 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:



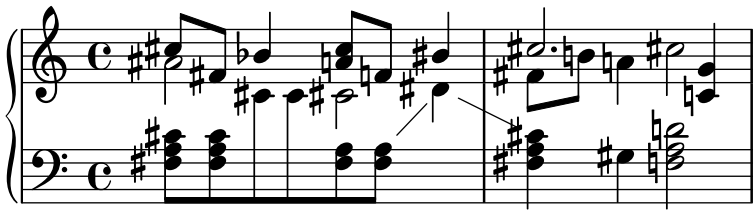
`modern-cautionary`

This rule is similar to `modern`, but the 'extra' accidentals are printed as cautionary accidentals (with parentheses). They can also be printed at a different size by overriding `AccidentalCautionary`'s `font-size` property.



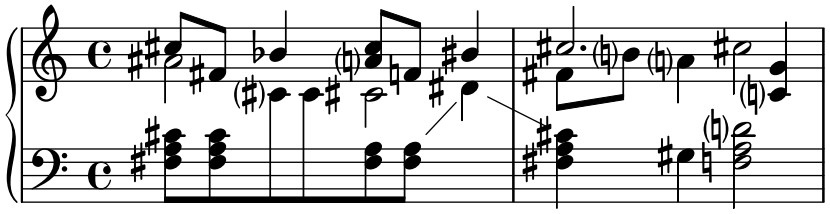
`modern-voice`

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



`modern-voice-cautionary`

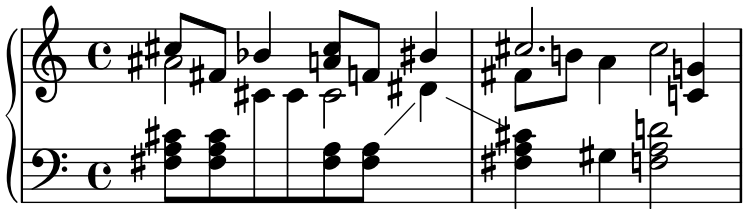
This rule is the same as `modern-voice`, but with the extra accidentals (the ones not typeset by `voice`) typeset as cautionaries. Even though all accidentals typeset by `default` *are* typeset with this rule, some of them are typeset as cautionaries.



`piano`

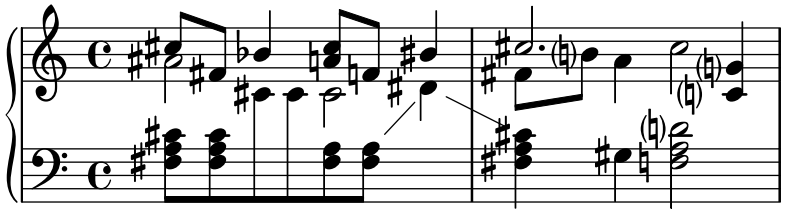
This rule reflects twentieth-century practice for piano notation. Its behavior is very similar to `modern` style, but here accidentals also get canceled across the staves in the same `GrandStaff` or `PianoStaff`, hence all the cancellations of the final notes.

This accidental style applies to the current `GrandStaff` or `PianoStaff` unless qualified with a second argument.



`piano-cautionary`

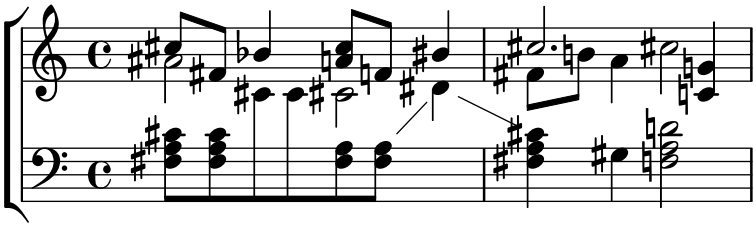
This is the same as `piano` but with the extra accidentals typeset as cautionaries.



`choral`

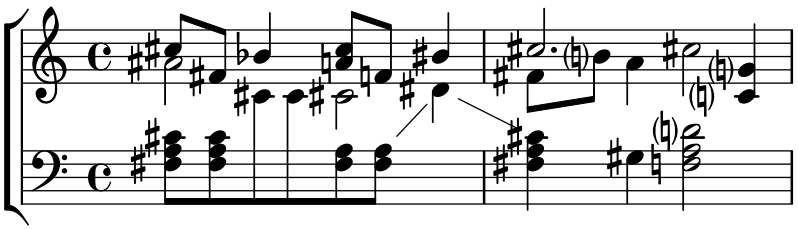
This rule is a combination of the `modern-voice` and the `piano` style. It shows all accidentals required for singers that only follow their own voice, as well as additional accidentals for readers that follow all voices of an entire `ChoirStaff` simultaneously.

This accidental style applies to the current `ChoirStaff` unless qualified with a second argument.



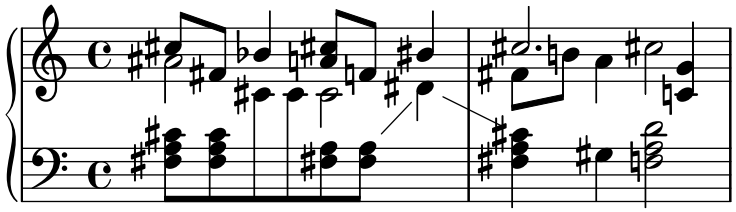
`choral-cautionary`

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



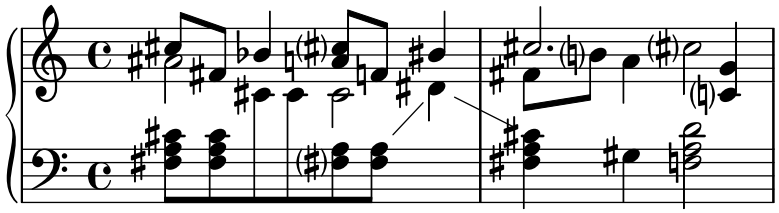
`neo-modern`

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.



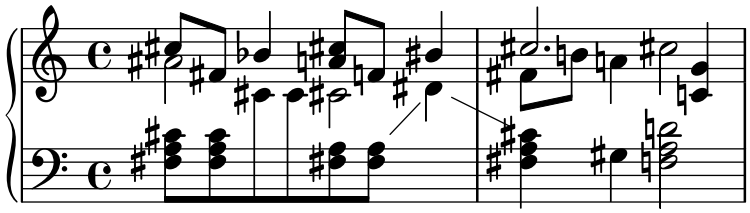
`neo-modern-cautionary`

This rule is similar to `neo-modern`, but the 'extra' accidentals are printed as cautionary accidentals (with parentheses). They can also be printed at a different size by overriding `AccidentalCautionary`'s `font-size` property.



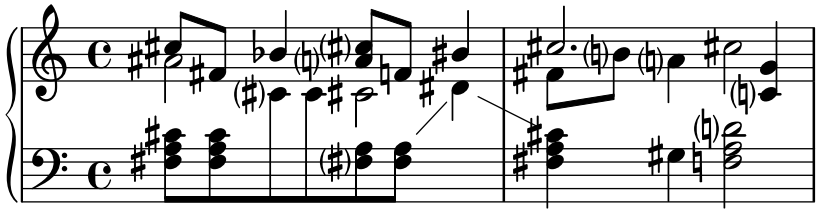
`neo-modern-voice`

This rule is used for multi-voice 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`.



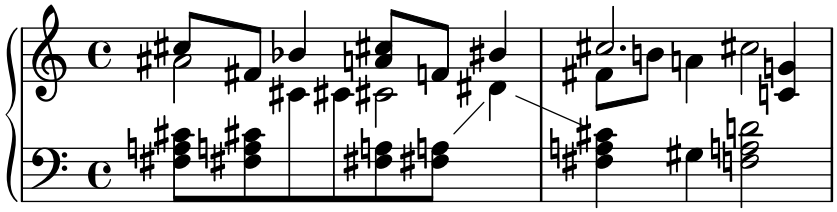
`neo-modern-voice-cautionary`

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



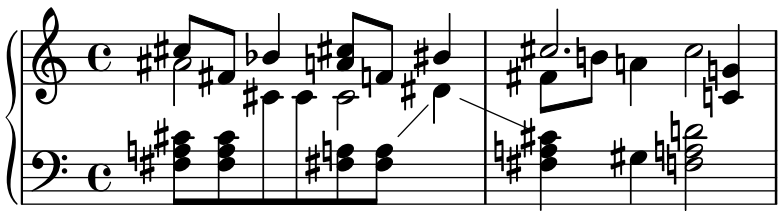
`dodecaphonic`

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.



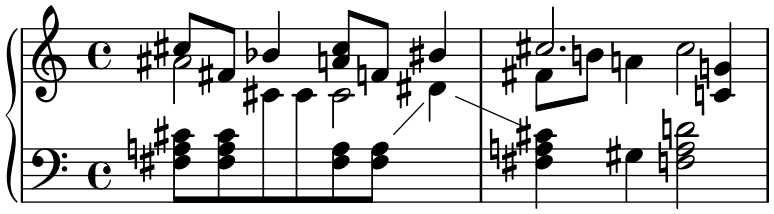
`dodecaphonic-no-repeat`

Like with the dodecaphonic accidental style *every* note gets an accidental sign by default, but accidentals are suppressed for pitches immediately repeated within the same staff.
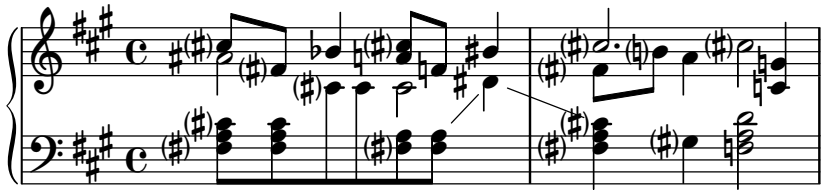


`dodecaphonic-first`

Similar to the dodecaphonic accidental style *every* pitch gets an accidental sign, but only the first time it is encountered in a measure. Accidentals are only remembered for the actual octave but throughout voices.



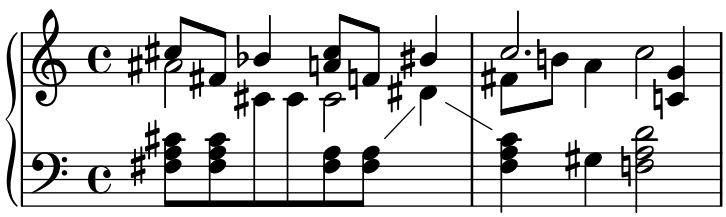
`teaching`

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.



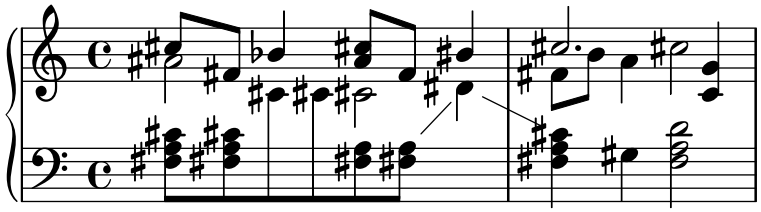
`no-reset`

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.



See also
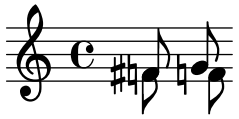
Snippets: Section “Pitches” in *Snippets*.

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

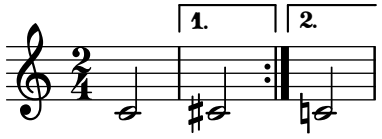
Simultaneous notes are not considered in the automatic determination of accidentals; only previous notes and the key signature are taken into account. Forcing accidentals with ‘`!`’ or ‘`?`’ may be required when the same note name occurs simultaneously with different alterations, as in `<f! fis!>`.

```
\relative c' <<
  { fis8 g } \\
  { f! f }
>>
```



A more sophisticated solution is given in a LilyPond Wiki snippet (`https://wiki.lilypond.community/wiki/Accidental_adjustments_for_single-voice_polyphony`).

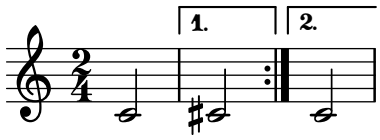
In alternative endings, cautionary cancellation should be based on the previous *played* measure, but it is based on the previous *printed* measure. In the following example, the natural `c` in the second alternative does not need a natural sign:



The following workaround can be used: define a function that locally changes the accidental style to `forget`:

```
forget = #(define-music-function (music) (ly:music?) #{
  \accidentalStyle forget
  #music
  \accidentalStyle modern
#})
{
  \accidentalStyle modern
  \time 2/4
  \repeat volta 2 {
    c'2
```

```
  }
  \alternative {
     \volta 1 { cis' }
     \volta 2 { \forget c' }
  }
}
```
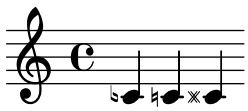


1.3.6 Alternate accidental glyphs

Non-Western and ancient notation systems have their own accidentals. The glyphs are controlled through the `alterationGlyphs` property of the `Staff` context and similar context types. The predefined values for this property are listed in Section B.10 [Accidental glyph sets], page 922.

```
\layout {
  \context {
    \Staff
    alterationGlyphs = #alteration-vaticana-glyph-name-alist
  }
}
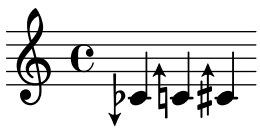
{ ces' c' cis' }
```



The property may also be set to a custom associative list mapping alterations to glyph names. Alterations are given as fractions in tones. Glyphs are listed at [Accidental glyphs], page 909.

```
\layout {
  \context {
    \Staff
    alterationGlyphs =
      #'((-1/2 . "accidentals.flat.arrowdown")
         (0 . "accidentals.natural.arrowup")
         (1/2 . "accidentals.sharp.arrowup"))
  }
}

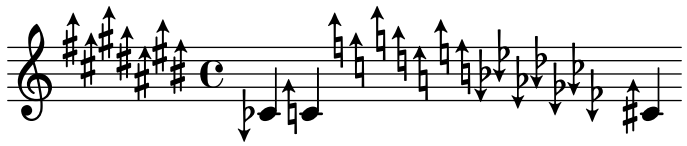
{ ces' c' cis' }
```



The `padding-pairs` property of `KeySignature` and `KeyCancellation` objects is an associative list mapping pairs of glyphs to the padding that should be added between these glyphs in key signatures.

```
\layout {
  \context {
    \Staff
    alterationGlyphs =
      #'((-1/2 . "accidentals.flat.arrowdown")
         (0 . "accidentals.natural.arrowup")
         (1/2 . "accidentals.sharp.arrowup"))
    \override KeySignature.padding-pairs =
      #'((("accidentals.sharp.arrowup" . "accidentals.sharp.arrowup")
            . 0.25)
         (("accidentals.flat.arrowdown" . "accidentals.flat.arrowdown")
            . 0.3))
    \override KeyCancellation.padding-pairs =
      #'((("accidentals.natural.arrowup" . "accidentals.natural.arrowup")
            . 0.7))
  }
}

{
  \key cis \major
  ces' c'
  \key ces \major
  cis'
}
```



See also

Notation Reference: Section B.10 [Accidental glyph sets], page 922, [Accidental glyphs], page 909.

Internals Reference: Section "accidental-switch-interface" in *Internals Reference*, Section "Alteration_glyph_engraver" in *Internals Reference*, Section "key-signature-interface" in *Internals Reference*.
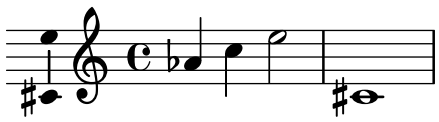
1.3.7 Ambitus

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 {
  aes' c e2
  cis,1
}
```
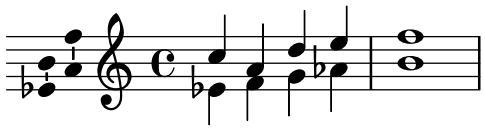


Selected snippets

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
  }
>>
```
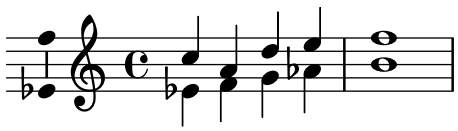


Ambitus with multiple voices

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
  }
>>
```



Changing the ambitus gap

It is possible to change the default gap between the ambitus noteheads and the line joining them.

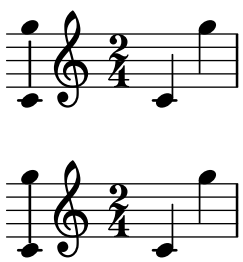
```
\layout {
  \context {
    \Voice
    \consists "Ambitus_engraver"
  }
}

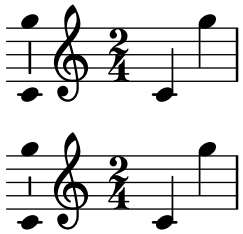
\new Staff {
  \time 2/4
  % Default setting
  c'4 g''
}

\new Staff {
  \time 2/4
  \override AmbitusLine.gap = 0
  c'4 g''
}

\new Staff {
  \time 2/4
  \override AmbitusLine.gap = 1
  c'4 g''
}

\new Staff {
  \time 2/4
  \override AmbitusLine.gap = 1.5
  c'4 g''
}
```
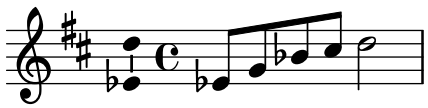




Ambitus after key signature

By default, ambitus are positioned at the left of the clef. The `\ambitusAfter` function allows for changing this placement. Syntax is `\ambitusAfter` *`grob-interface`*; see Graphical Object Interfaces (`https://lilypond.org/doc/v2.24/Documentation/internals/graphical-object-interfaces`) for a list of possible values for *grob-interface*.

A common use case is printing the ambitus between key signature and time signature.

```
\new Staff \with {
  \consists Ambitus_engraver
} \relative {
  \ambitusAfter key-signature
  \key d \major
  es'8 g bes cis d2
}
```



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.

1.4 Note heads
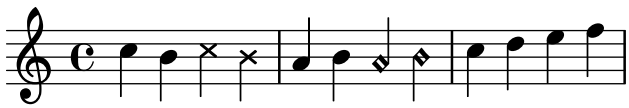
This section suggests ways of altering note heads.

1.4.1 Special note heads

The appearance of note heads may be altered:

```
\relative c'' {
  c4 b
  \override NoteHead.style = #'cross
  c4 b
  \revert NoteHead.style
  a b
  \override NoteHead.style = #'harmonic
```

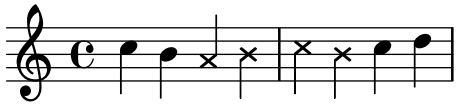
```
  a b
  \revert NoteHead.style
  c4 d e f
}
```



To see all note head styles, see Section B.9 [Note head styles], page 922.

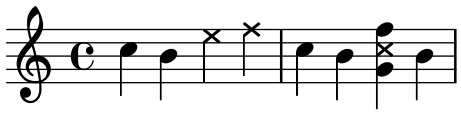
The cross style is used to represent a variety of musical intentions. The following generic predefined commands modify the note head in both staff and tablature contexts and can be used to represent any musical meaning:

```
\relative {
  c''4 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:
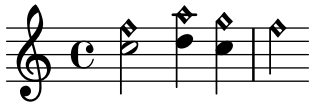
```
\relative {
  c''4 b
  \xNote { e f }
  c b < g \xNote c f > b
}
```



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 similar shorthand for diamond shapes:

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



Predefined commands

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

See also

Snippets: Section “Pitches” in *Snippets*.

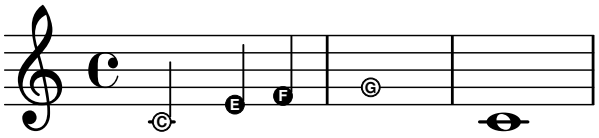
Notation Reference: Section B.9 [Note head styles], page 922, Section 5.1.1 [Chorded notes], page 216, Section 12.2.2 [Indicating harmonics and dampened notes], page 482.

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*.

1.4.2 Easy notation note heads

The ‘easy play’ note head includes a note name inside the head. It is used in music for beginners. To make the letters readable, it should be printed in a large font size. To print with a larger font, see Section 27.2 [Setting the staff size], page 685.

```
#(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
   (make-engraver
    (acknowledgers
     ((note-head-interface 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))))))

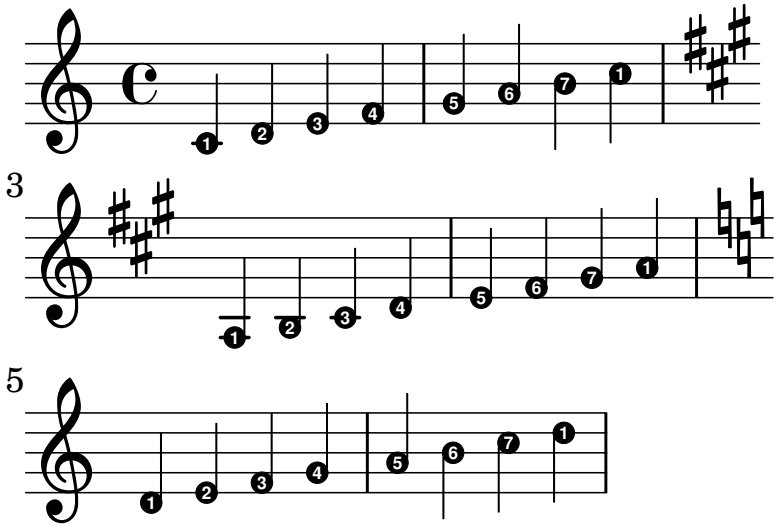
#(set-global-staff-size 30)

\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 27.2 [Setting the staff size], page 685.

Snippets: Section “Pitches” in *Snippets*.
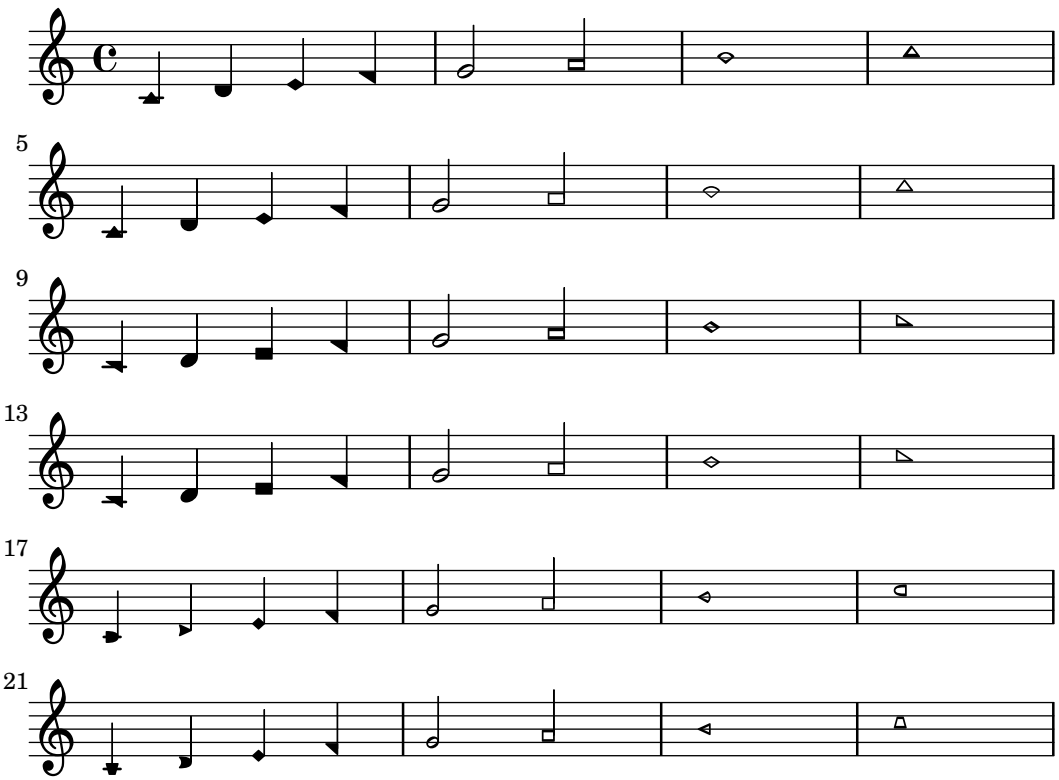
Internals Reference: Section “note-event” in *Internals Reference*, Section “Note_heads_engraver” in *Internals Reference*, Section “NoteHead” in *Internals Reference*, Section “note-head-interface” in *Internals Reference*.

1.4.3 Shape note heads

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

Shape note heads can be produced in Sacred Harp, Southern Harmony, Funk (Harmonia Sacra), Walker, and Aiken (Christian Harmony) styles:
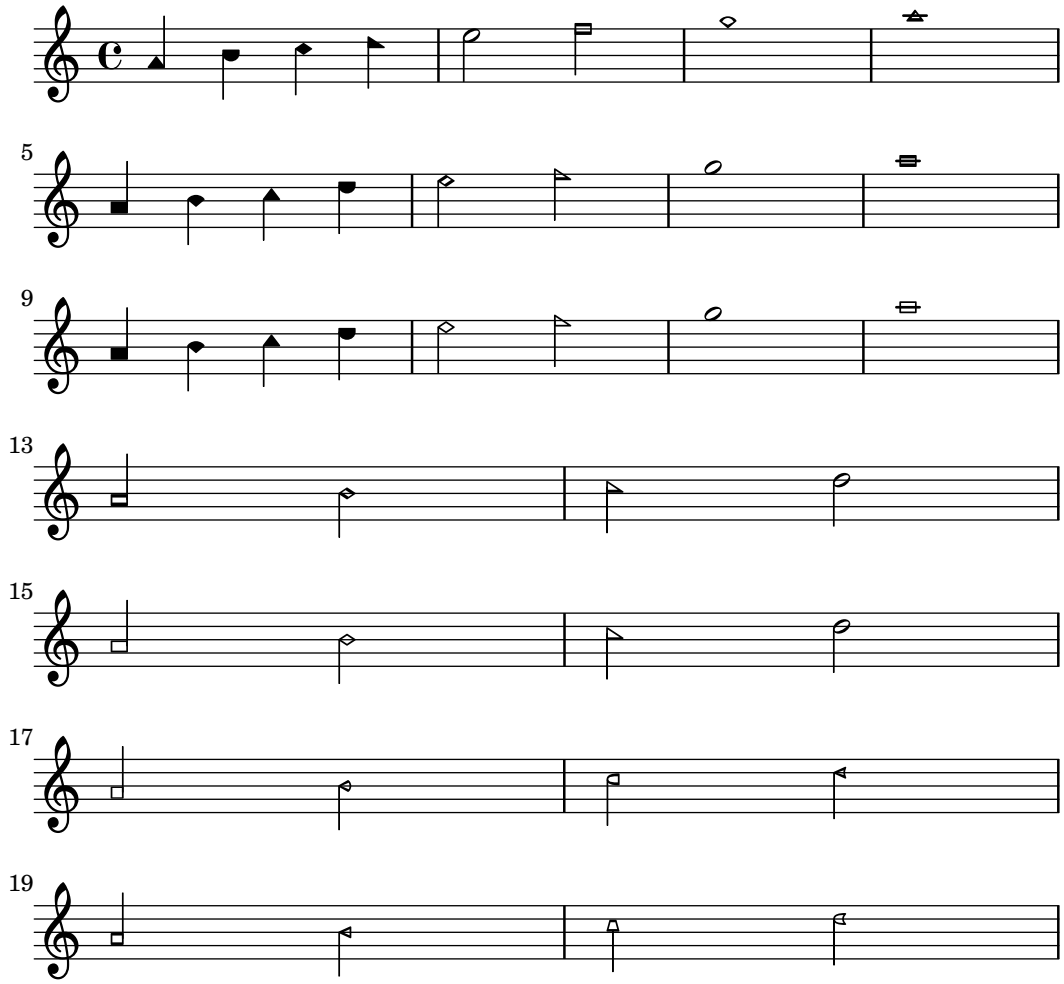
```
\relative c'' {
  \aikenHeads
  c, d e f g2 a b1 c \break
  \aikenThinHeads
  c,4 d e f g2 a b1 c \break
  \sacredHarpHeads
  c,4 d e f g2 a b1 c \break
  \southernHarmonyHeads
  c,4 d e f g2 a b1 c \break
  \funkHeads
  c,4 d e f g2 a b1 c \break
  \walkerHeads
  c,4 d e f g2 a b1 c \break
}
```



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:

```
\relative c'' {
  \key a \minor
  \aikenHeads
  a b c d e2 f g1 a \break
  \aikenHeadsMinor
  a,4 b c d e2 f g1 a \break
  \aikenThinHeadsMinor
  a,4 b c d e2 f g1 a \break
  \sacredHarpHeadsMinor
```

```
  a,2 b c d \break
  \southernHarmonyHeadsMinor
  a2 b c d \break
  \funkHeadsMinor
  a2 b c d \break
  \walkerHeadsMinor
  a2 b c d \break
}
```



Predefined commands

\aikenHeads, \aikenHeadsMinor, \aikenThinHeads, \aikenThinHeadsMinor, \funkHeads, \funkHeadsMinor, \sacredHarpHeads, \sacredHarpHeadsMinor, \southernHarmonyHeads, \southernHarmonyHeadsMinor, \walkerHeads, \walkerHeadsMinor.
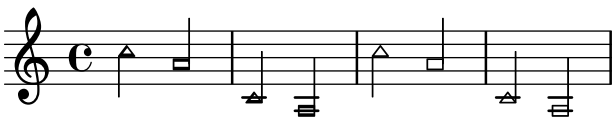
Selected snippets

Aiken head thin variant noteheads

Aiken head white notes get harder to read at smaller staff sizes, especially with ledger lines. Losing interior white space makes them appear as quarter notes.

```
\score {
  {
    \aikenHeads
    c''2 a' c' a

    % Switch to thin-variant noteheads
    \set shapeNoteStyles = ##(doThin reThin miThin
                              faThin sol laThin tiThin)
    c'' a' c' a
  }
}
```
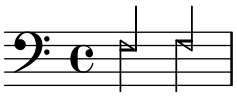


Direction of merged 'fa' shape note heads

Using property `NoteCollision.fa-merge-direction`, the direction of "fa" shape note heads ("fa", "faThin", etc.) can be controlled independently of the stem direction if two voices with the same pitch and different stem directions are merged. If this property is not set, the "down" glyph variant is used.

```
{
  \clef bass

  << { \aikenHeads
       f2
       \override Staff.NoteCollision.fa-merge-direction = #UP
       f2 }
  \\ { \aikenHeads
       f2
       f2 }
  >>
}
```



Applying note head styles depending on the step of the scale

The `shapeNoteStyles` property can be used to define various note head styles for each step of the scale (as set by the key signature or the `tonic` property).

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

That said, to imitate old American song books, there are several predefined note head styles available through shortcut commands such as `\aikenHeads` or `\sacredHarpHeads`.

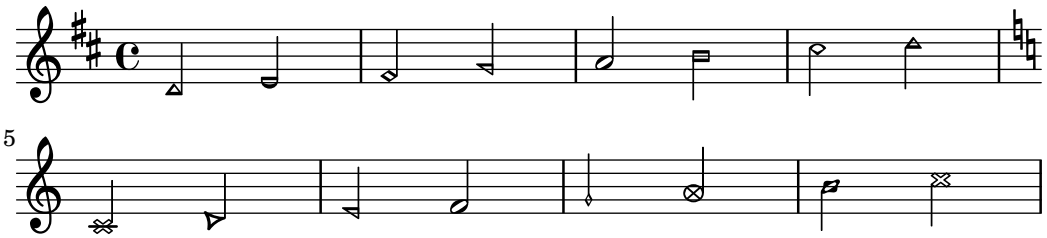
This example shows different ways to obtain shape note heads, and demonstrates the ability to transpose a melody without losing the correspondence between harmonic functions and note head styles.

```
fragment = {
  \key c \major
  c2 d
  e2 f
  g2 a
  b2 c
}

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

  \break

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



To see all note head styles, see Section B.9 [Note head styles], page 922.

See also

Snippets: Section “Pitches” in *Snippets*.

Notation Reference: Section B.9 [Note head styles], page 922.

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*.

1.4.4 Improvisation

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

```
\new Voice \with {
  \consists Pitch_squash_engraver
} \relative {
  e''8 e g a a16( bes) a8 g
  \improvisationOn
  e8 ~
  2 ~ 8 f4 f8 ~
```

```
  2
  \improvisationOff
  a16( bes) a8 g e
}
```



Predefined commands
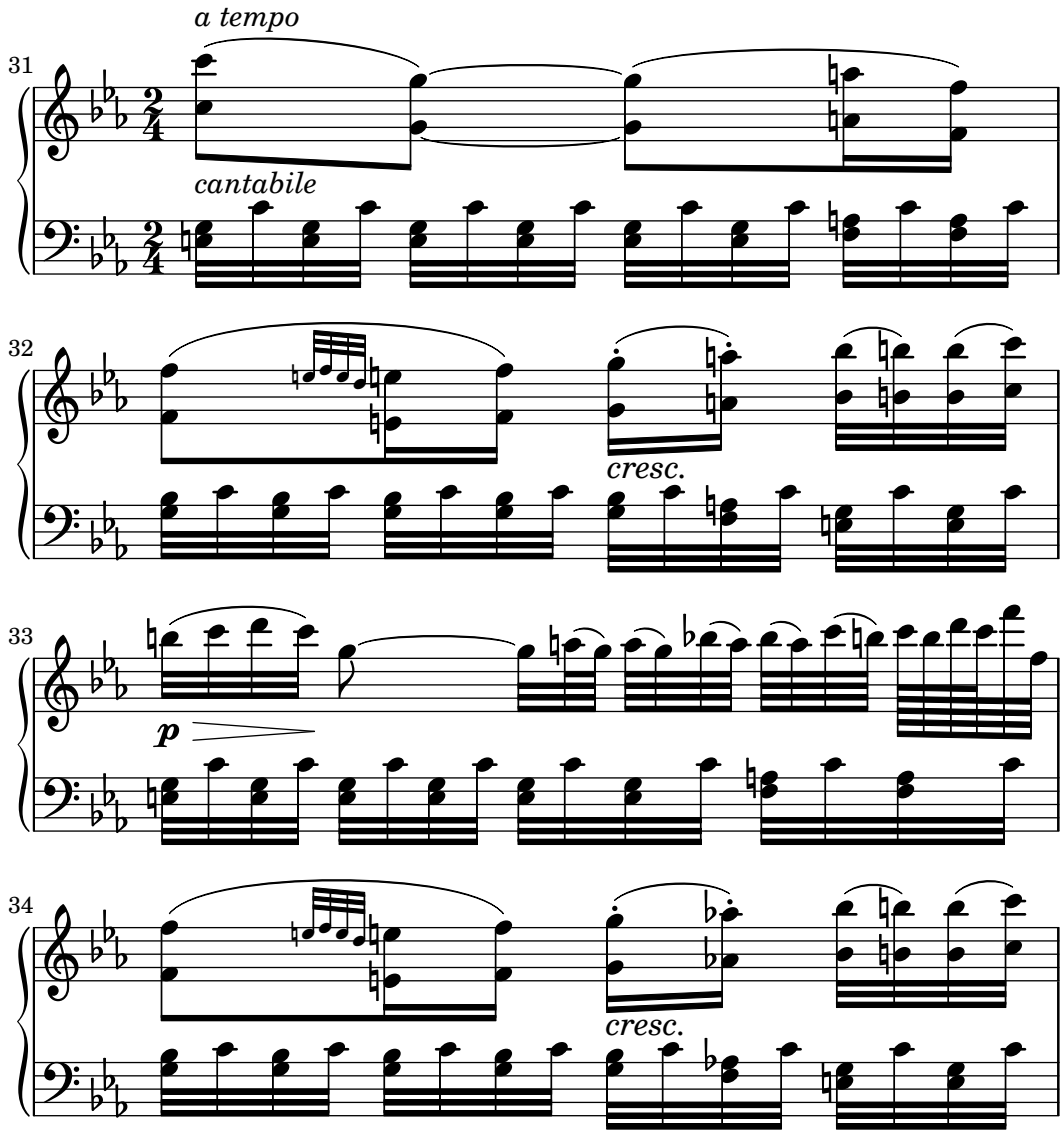
`\improvisationOn`, `\improvisationOff`.

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*.

2 Rhythms



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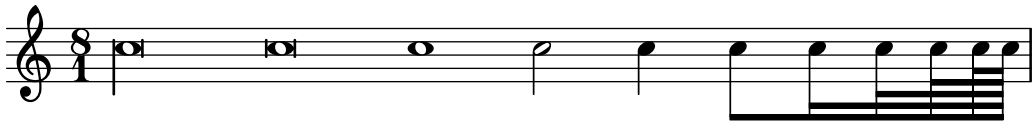
2.1 Writing rhythms

2.1.1 Durations

The durations of notes are entered using numbers and dots. The number entered is based on the reciprocal value of the length of the note. For example, a quarter note is designated using the numerical value of `4` as it is a 1/4 note, a half note using `2`, an eighth using `8` and so on. Durations as short as `1024` notes can be entered but shorter values, while possible, can only be entered as beamed notes. Also see Section 2.4 [Beams], page 102.
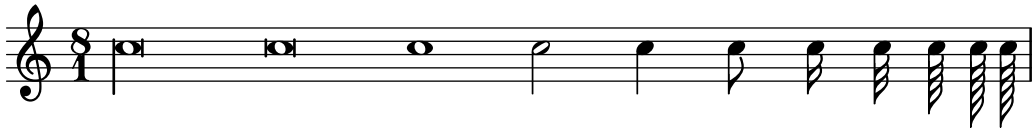
For notes longer than a whole use the `\longa` – double breve – and `\breve` commands. A note with the duration of a quadruple breve is possible using the `\maxima` command but is only supported within ancient music notation. See Chapter 17 [Ancient notation], page 538.

```
\relative {
  \time 8/1
  c''\longa c\breve c1 c2
  c4 c8 c16 c32 c64 c128 c128
}
```
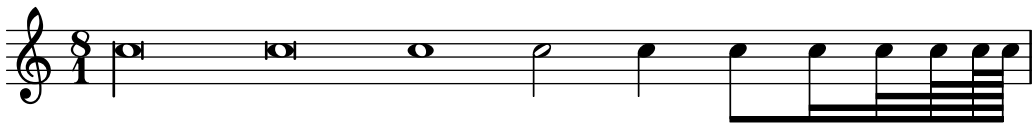


Here are the same durations with automatic beaming turned off.

```
\relative {
  \time 8/1
  \autoBeamOff
  c''\longa c\breve c1 c2
  c4 c8 c16 c32 c64 c128 c128
}
```
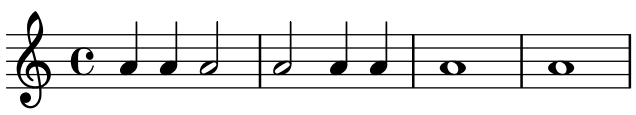


Isolated durations – durations without a pitch – that occur within a music sequence will take their pitch from the preceding note or chord.

```
\relative {
  \time 8/1
  c'' \longa \breve 1 2
  4 8 16 32 64 128 128
}
```
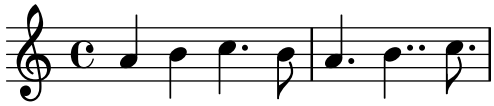


Isolated pitches – pitches without a duration – that occur within a music sequence will take their duration from the preceding note or chord. If there is no preceding duration, then default for the note is always `4`, a quarter note.

```
\relative { a' a a2 a a4 a a1 a }
```



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

```
\relative { a'4 b c4. b8 a4. b4.. c8. }
```



To avoid clashing with staff lines, dots on notes are normally moved up. In polyphonic situations however, they can be placed, manually, above or below the staff as required. See Section 36.1 [Direction and placement], page 774.

Some note durations cannot be represented using just numbers and dots but only by tying two or more notes together. See Section 2.1.4 [Ties], page 63.

To specify durations that align the syllables of lyrics and notes together see Chapter 9 [Vocal music], page 345.

Notes can also be spaced proportionately to their duration, see Section 30.6 [Proportional notation], page 722.

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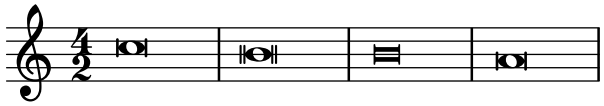
`\autoBeamOn`, `\autoBeamOff`, `\dotsUp`, `\dotsDown`, `\dotsNeutral`.

Selected snippets

Alternative breve notes

Breve notes are also available with two vertical lines on each side of the notehead instead of one line and in baroque style.
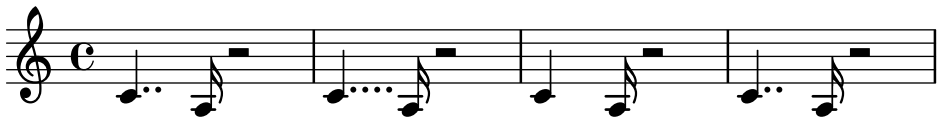
```
\relative c'' {
  \time 4/2
  c\breve |
  \override Staff.NoteHead.style = #'altdefault
  b\breve
  \override Staff.NoteHead.style = #'baroque
  b\breve
  \revert Staff.NoteHead.style
  a\breve
}
```



Changing the number of augmentation dots per note

The number of augmentation dots on a single note can be overridden by setting the `dot-count` property of the `Dots` grob.

```
\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: Section 2.4 [Beams], page 102, Section 2.1.4 [Ties], page 63, Section 7.1.9 [Stems], page 297, Section 2.1 [Writing rhythms], page 52, Section 2.2 [Writing rests], page 67, Chapter 9 [Vocal music], page 345, Chapter 17 [Ancient notation], page 538, Section 30.6 [Proportional notation], page 722.

Snippets: Section “Rhythms” in *Snippets*.

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

Known issues and warnings

While there is no fundamental limit to rest durations (longest or shortest), there is a limit to the number of glyphs possible so only rests between `1024` and `\maxima` may be printed.

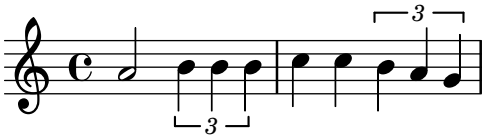
2.1.2 Tuplets

Tuplets are made from a music expression with the `\tuplet` command, multiplying the speed of the music expression by a fraction:

```
\tuplet fraction { music }
```
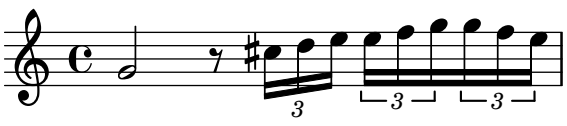
The fraction's numerator will be printed over or under the notes, optionally with a bracket. The most common tuplets are triplets (3 notes played within the duration normally allowed for 2).

```
\relative {
  a'2 \tuplet 3/2 { b4 4 4 }
  c4 c \tuplet 3/2 { b4 a g }
}
```
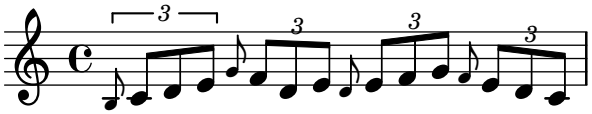


When entering long passages of tuplets, having to write a separate `\tuplet` command for each group is inconvenient. It is possible to specify the duration of one tuplet group directly before the music in order to have the tuplets grouped automatically:

```
\relative {
  g'2 r8 \tuplet 3/2 8 { cis16 d e e f g g f e }
}
```



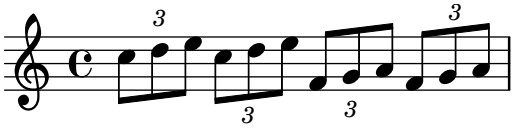
Other grouping methods are mentioned below. When employing any of the automatic grouping methods, LilyPond will not start a tuplet group while grace notes are pending. In contrast, when no automatic grouping is being used, the input is taken at face value.

```
\relative c' {
  \tuplet 3/2 { \grace b8 c d e }
  \tuplet 3/2 4 { \grace g8 f d e \grace d e f g \grace f e d c }
}
```
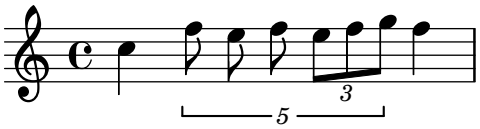


Tuplet brackets may be manually placed above or below the staff:

```
\relative {
  \tupletUp \tuplet 3/2 { c''8 d e }
  \tupletNeutral \tuplet 3/2 { c8 d e }
  \tupletDown \tuplet 3/2 { f,8 g a }
  \tupletNeutral \tuplet 3/2 { f8 g a }
}
```



Tuplets may be nested:
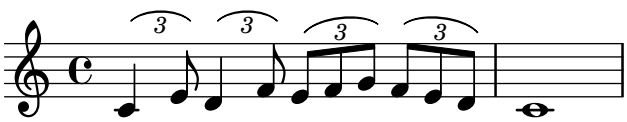
```
\relative {
  \autoBeamOff
  c''4 \tuplet 5/4 { f8 e f \tuplet 3/2 { e[ f g] } } f4
}
```



Modifying nested tuplets which begin at the same musical moment must be done with `\tweak`; see Section 35.7 [\tweak and \single], page 764.

Tuplet brackets may be replaced with slurs, as is preferred in many older editions:

```
\relative {
  \tuplet 3/2 4 {
    \override TupletBracket.tuplet-slur = ##t
    c'4 e8 d4 f8
    \override TupletBracket.bracket-visibility = ##t
    e f g f e d
  } c1
}
```



By default, a bracket is only printed if all of the notes it spans are not beamed together; in some cases (for example with slurs, as in the example above) it may be preferable to change that behavior, through the `bracket-visibility` property as detailed in one of the following snippets.

More generally, either or both the TupletBracket and TupletNumber objects may be hidden or shown as explained in Section 36.7 [Visibility of objects], page 784; however, a more flexible way of modifying the duration of notes without printing a tuplet bracket is also introduced in Section 2.1.3 [Scaling durations], page 61.

Predefined commands

\tupletUp, \tupletDown, \tupletNeutral.

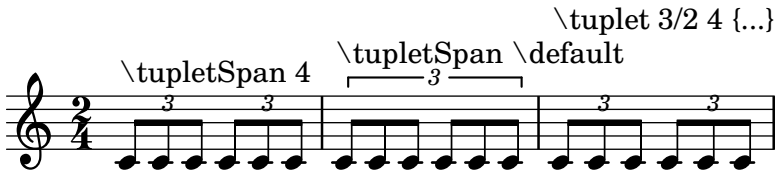
Selected snippets

Entering several tuplets using only one \tuplet command

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

There are ways to set tupletSpannerDuration besides using a \set command. The command \tupletSpan sets it to a given duration, or clears it when instead of a duration \default is specified. Another way is to use an optional argument with \tuplet.

```
\relative c' {
  \time 2/4
  \tupletSpan 4
  \tuplet 3/2 { c8^"\\tupletSpan 4" c c c c c }
  \tupletSpan \default
  \tuplet 3/2 { c8^"\\tupletSpan \\default" c c c c c }
  \tuplet 3/2 4 { c8^"\\tuplet 3/2 4 {...}" c c c c c }
}
```

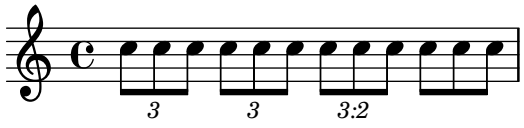


Changing the tuplet number

By default, only the numerator of the tuplet number is printed over the tuplet bracket, i.e., the numerator of the argument to the \tuplet command.

Alternatively, *num*:*den* of the tuplet number may be printed, or the tuplet number may be suppressed altogether.

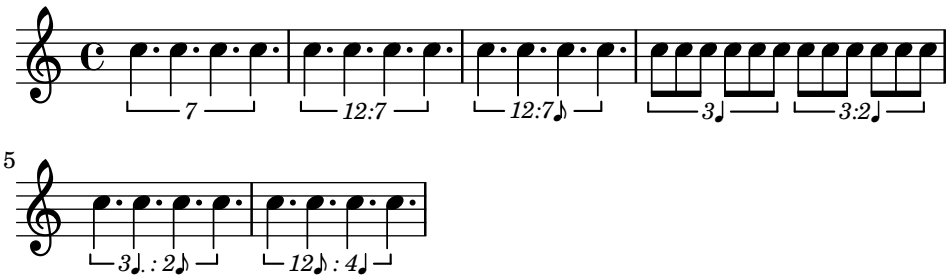
```
\relative c'' {
  \tuplet 3/2 { c8 c c }
  \tuplet 3/2 { c8 c c }
  \override TupletNumber.text = #tuplet-number::calc-fraction-text
  \tuplet 3/2 { c8 c c }
  \omit TupletNumber
  \tuplet 3/2 { c8 c c }
}
```



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)
  \tuplet 3/2  { c4. c4. c4. c4. }
  \once \override TupletNumber.text =
    #(tuplet-number::non-default-tuplet-fraction-text 12 7)
  \tuplet 3/2  { c4. c4. c4. c4. }
  \once \override TupletNumber.text =
    #(tuplet-number::append-note-wrapper
      (tuplet-number::non-default-tuplet-fraction-text 12 7)
      (ly:make-duration 3 0))
  \tuplet 3/2  { c4. c4. c4. c4. }
  \once \override TupletNumber.text =
    #(tuplet-number::append-note-wrapper
      tuplet-number::calc-denominator-text
      (ly:make-duration 2 0))
  \tuplet 3/2  { c8 c8 c8 c8 c8 c8 }
  \once \override TupletNumber.text =
    #(tuplet-number::append-note-wrapper
      tuplet-number::calc-fraction-text
      (ly:make-duration 2 0))
  \tuplet 3/2  { c8 c8 c8 c8 c8 c8 }
  \once \override TupletNumber.text =
    #(tuplet-number::fraction-with-notes
      (ly:make-duration 2 1) (ly:make-duration 3 0))
  \tuplet 3/2  { c4. c4. c4. c4. }
  \once \override TupletNumber.text =
    #(tuplet-number::non-default-fraction-with-notes 12
      (ly:make-duration 3 0) 4 (ly:make-duration 2 0))
  \tuplet 3/2  { c4. c4. c4. c4. }
}
```



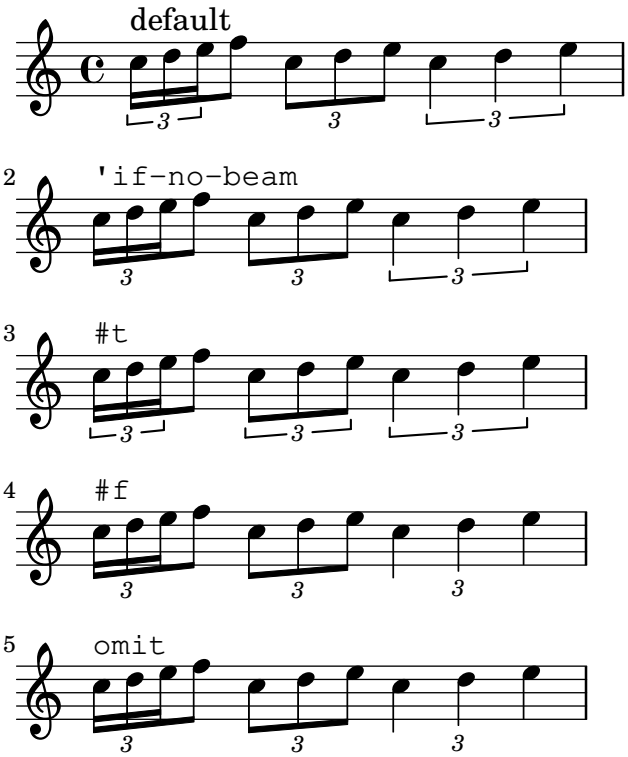
Controlling tuplet bracket visibility

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), `if-no-beam` (only print a bracket if there is no beam) or `#f` (never print a bracket). The latter is in fact equivalent to omitting the `TupletBracket` object altogether from the printed output.

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

\new Voice {
  \relative c' {
    \override Score.TextMark.non-musical = ##f
    \textMark "default" \music
    \override TupletBracket.bracket-visibility = #'if-no-beam
    \textMark \markup \typewriter "'if-no-beam" \music
    \override TupletBracket.bracket-visibility = ##t
    \textMark \markup \typewriter "#t" \music
    \override TupletBracket.bracket-visibility = ##f
    \textMark \markup \typewriter "#f" \music
    \omit TupletBracket
    \textMark \markup \typewriter "omit" \music
  }
}
```
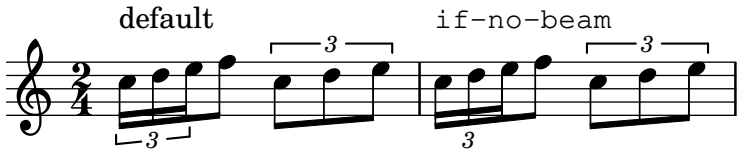


Printing tuplet brackets on the note head side

Whichever option you choose for controlling the tuplet bracket visibility, it will show or hide the tuplet bracket irrespectively of tuplet bracket placement (stem side or note head side). However, when placing the tuplet bracket on the note head side some authors recommend always printing the tuplet bracket. The option `visible-over-note-heads` can be used to achieve this.

```
music = \relative c'' {
  \tupletNeutral \tuplet 3/2 { c16[ d e } f8]
  \tupletUp \tuplet 3/2 { c8 d e }
}

\new Voice {
  \relative c' {
    \override TextScript.staff-padding = #2.5

    \time 2/4
    \override TupletBracket.visible-over-note-heads = ##t
    \override Score.TextMark.non-musical = ##f
    <>^\markup "default" \music
    \override TupletBracket.bracket-visibility = #'if-no-beam
    <>^\markup \typewriter "if-no-beam" \music
  }
}
```



Permitting line breaks within beamed tuplets

These artificial examples show how both manual and automatic line breaks may be permitted within beamed tuplets that can't be rhythmically split in an exact way.

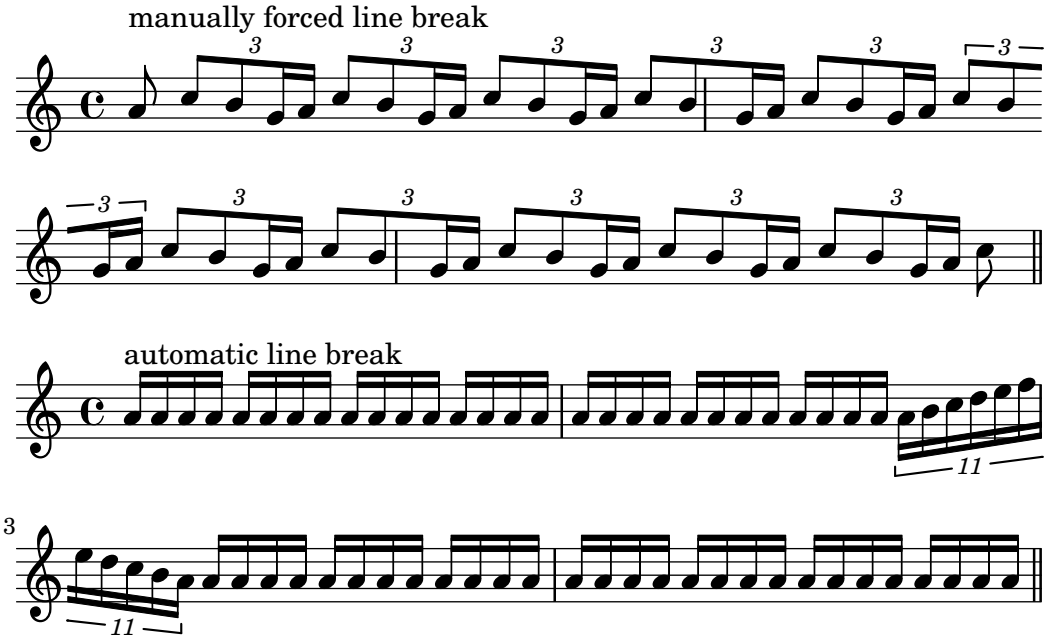
This feature only works with manually beamed tuplets.

```
\layout {
  \context {
    \Voice
    % Permit automatic line breaks within tuplets.
    \remove "Forbid_line_break_engraver"
    % Allow beams to be broken at line breaks.
    \override Beam.breakable = ##t
  }
}

\relative c'' {
  <>^"manually forced line break"
  a8
  \*5 { \tuplet 3/2 { c8[ b g16 a] } }
  \tuplet 3/2 { c8[ b \break g16 a] }
  \*5 { \tuplet 3/2 { c8[ b g16 a] } }
  c8 \bar "||"
}

\relative c'' {
  <>^"automatic line break"
  \*28 a16
  \tuplet 11/8 { a16[ b c d e f e d c b a] }
  \*28 a16 \bar "||"
```

```
}
```



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: Section 36.1 [Direction and placement], page 774, Section 36.7 [Visibility of objects], page 784, Section 2.3.1 [Measures], page 78, Section 2.1.3 [Scaling durations], page 61, Section 35.7 [`\tweak` and `\single`], page 764, Section 2.3.6 [Polymetric notation], page 95.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “TupletBracket” in *Internals Reference*, Section “TupletNumber” in *Internals Reference*, Section “TimeScaledMusic” in *Internals Reference*.
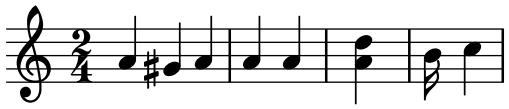
2.1.3 Scaling durations

The duration of single notes, rests or chords may be multiplied by a fraction `N/M` by appending `*N/M` (or `*N` if `M` is 1) to the duration. Factors may also be added by using Scheme expressions evaluating to a number or musical length like `*#(ly:music-length` *`music`*`)`. This is convenient for scaling a duration of ‘`1`’ to let a note or multi-measure rest stretch to a length derived from a music variable.

Adding a factor 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 like `*L*M/N`. Factors are part of the duration: if a duration is not specified for subsequent notes, the default duration taken from the preceding note will include any scaling factor.

In the following example, the first three notes take up exactly two beats, but no triplet bracket is printed.
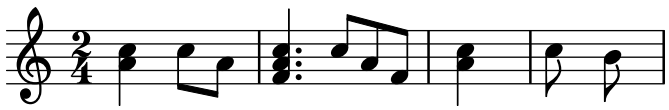
```
\relative {
  \time 2/4
  % Alter durations to triplets
  a'4*2/3 gis a
  % Normal durations
  a4 a
  % Double the duration of chord
  <a d>4*2
  % Duration of quarter, appears like sixteenth
  b16*4 c4
}
```



The duration of spacer rests 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 given scale factor, usually *num*/*den*. Here is an example showing how music can be compressed and expanded:

```
\relative {
  \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 {
    <c' a>4 c8 b
  }
}
```



One application of this command is in polymetric notation, see Section 2.3.6 [Polymetric notation], page 95.

See also

Notation Reference: Section 2.1.2 [Tuplets], page 55, Section 2.2.2 [Invisible rests], page 69, Section 2.3.6 [Polymetric notation], page 95.

Snippets: Section “Rhythms” in *Snippets*.

Known issues and warnings

The calculation of the position within a measure must take into account all the scaling factors applied to the notes within that measure and any fractional carry-out from earlier measures. This calculation is carried out using rational numbers. If an intermediate numerator or denominator

in that calculation exceeds 2^30 the execution and typesetting will stop at that point without indicating an error.

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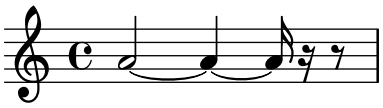
A tie connects two adjacent note heads of the same pitch. The tie in effect extends the duration of a note.

Ties that connect notes to nothing are called *laissez vibrer* articulation; see [Laissez vibrer], page 64, for the `\laissezVibrer` command. Ties that connect nothing to notes (as needed in seconda volta sections, for example), can be entered with the `\repeatTie` command; see [Repeat tie], page 64.

> **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.
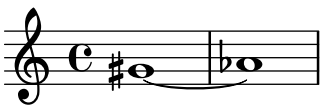
A tie is entered by appending a tilde symbol (‘~’) to the first of each pair of notes being tied. This indicates that the note should be tied to the following note, which must be at the same pitch. Note that ties make use of the ‘last explicit pitch’ interpretation of isolated durations:

```
{ a'2~ 4~ 16 r r8 }
```
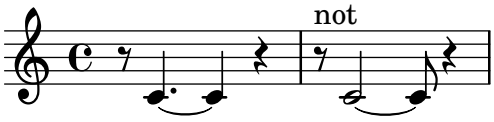


LilyPond also supports ties between notes that are enharmonically equivalent.

```
{ gis'1~ | as'1 }
```



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:

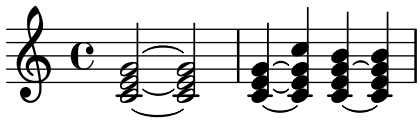
```
\relative {
  r8 c'4.~ 4 r4 |
  r8^"not" c2~ 8 r4
}
```



If you need to tie many notes across bar lines, it may be easier to use automatic note splitting, see Section 2.3.7 [Automatic note splitting], page 98. This mechanism automatically splits long notes, and ties them across bar lines.

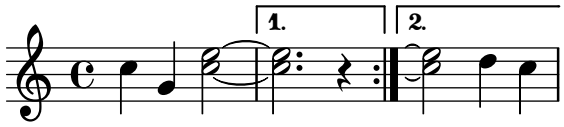
When a tie is applied to a chord, all note heads whose pitches match are connected. When no note heads match, no ties will be created. Chords may be partially tied by placing the ties inside the chord.

```
\relative c' {
  <c e g>2~ 2 |
  <c e g>4~ <c e g c>
    <c~ e g~ b> <c e g b> |
}
```
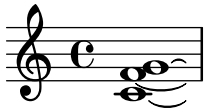


When a tie continues into alternative endings, you have to specify the repeated tie as follows:

```
\relative {
  \repeat volta 2 { c'' g <c e>2~ }
  \alternative {
    % the following note is tied normally
    \volta 1 { <c e>2. r4 }
    % the following note has a repeated tie
    \volta 2 { <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:
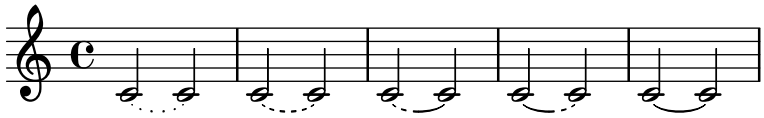
```
<c' f' g'>1\laissezVibrer
```



Ties may be made to curve up or down manually; see Section 36.1 [Direction and placement], page 774.
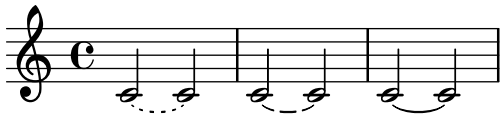
Ties may be made dashed, dotted, or a combination of solid and dashed.

```
\relative c' {
  \tieDotted
  c2~ 2
  \tieDashed
  c2~ 2
  \tieHalfDashed
  c2~ 2
  \tieHalfSolid
  c2~ 2
  \tieSolid
  c2~ 2
}
```



Custom dash patterns can be specified:
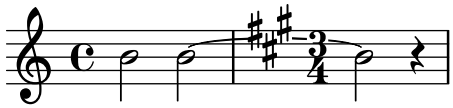
```
\relative c' {
  \tieDashPattern 0.3 0.75
  c2~ 2
  \tieDashPattern 0.7 1.5
  c2~ 2
  \tieSolid
  c2~ 2
}
```



Dash pattern definitions for ties have the same structure as dash pattern definitions for slurs. For more information about complex dash patterns, see Section 3.2.1 [Slurs], page 171.

Override *whiteout* and *layer* layout properties of objects that should cause a gap in ties.

```
\relative {
  \override Tie.layer = -2
  \override Staff.TimeSignature.layer = -1
  \override Staff.KeySignature.layer = -1
  \override Staff.TimeSignature.whiteout = ##t
  \override Staff.KeySignature.whiteout = ##t
  b'2 b~
  \time 3/4
  \key a \major
  b r4
}
```



Predefined commands

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

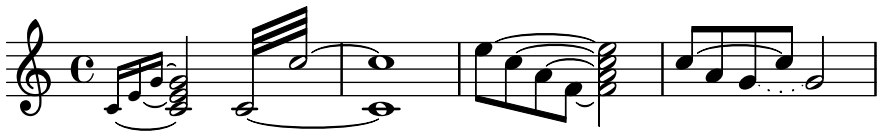
Selected snippets

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

A single tie may be engraved manually by changing the `staff-position` property (an offset) of the `Tie` grob; if there are multiple ties at the same musical moment, they can be adjusted manually by changing the `tie-configuration` property (a list of offset/direction pairs) of the `TieColumn` object.

The offset indicates the distance from the center of the staff in half-staff spaces, the direction can be either 1 (up) or -1 (down).

Note that LilyPond makes a distinction between exact and inexact values for the offset. If using an exact value (i.e., either an integer or a fraction like `(/ 4 5)`), the value serves as a rough vertical position that gets further tuned by LilyPond to make the tie avoid staff lines. If using an inexact value like a floating point number, it is taken as the precise vertical position without further adjustments.

```
\relative c' {
  <>^"default"
  g'1 ^~ g

  <>^"0"
  \once \override Tie.staff-position = 0
  g1 ^~ g

  <>^"0.0"
  \once \override Tie.staff-position = 0.0
  g1 ^~ g

  <>^"reset"
  \revert Tie.staff-position
  g1 ^~ g
}

\relative c' {
  \override TextScript.outside-staff-priority = ##f
  \override TextScript.padding = 0

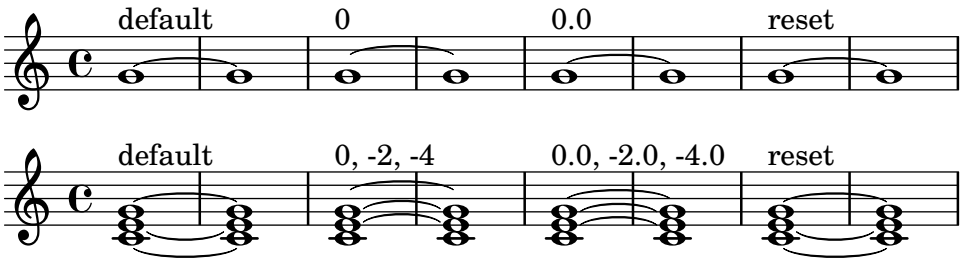
  <>^"default"
  <c e g>1~ <c e g>

  <>^"0, -2, -4"
  \override TieColumn.tie-configuration =
    #'((0 . 1) (-2 . 1) (-4 . 1))
  <c e g>1~ <c e g>

  <>^"0.0, -2.0, -4.0"
```

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

  <>^"reset"
  \override TieColumn.tie-configuration = ##f
  <c e g>1~ <c e g>
}
```



See also

Music Glossary: Section “tie” in *Music Glossary*, Section “laissez vibrer” in *Music Glossary*.

Notation Reference: Section 3.2.1 [Slurs], page 171, Section 2.3.7 [Automatic note splitting], page 98.

Snippets: Section “Expressive marks” in *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 ottavations during a tie is not really well-defined. In these cases, a slur may be preferable.

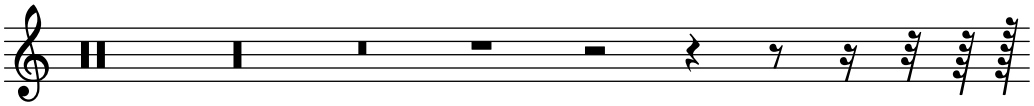
2.2 Writing rests

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

2.2.1 Rests

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

```
\new Staff {
  % These two lines are just to prettify this example
  \time 16/1
  \omit Staff.TimeSignature
  % 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 Section 2.2.3 [Full measure rests], page 71.

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.

```
\relative { a'4\rest d4\rest }
```



Selected snippets

Rest styles

Rests may be used in various styles.

```
restsA = {
  r\maxima r\longa r\breve r1 r2 r4 r8 r16 s32
  s64 s128 s256 s512 s1024 s1024
}
restsB = {
  r\maxima r\longa r\breve r1 r2 r4 r8 r16 r32
  r64 r128 r256 r512 r1024 s1024
}

\new Staff \relative c {
  \omit Score.TimeSignature
  \cadenzaOn

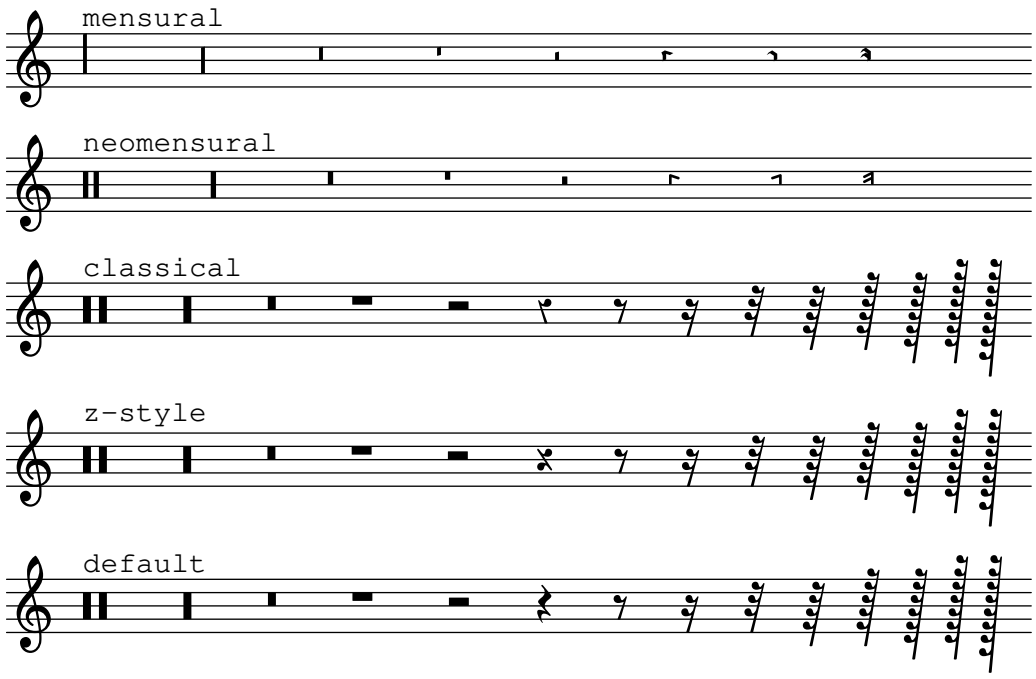
  \override Staff.Rest.style = #'mensural
  <>^\markup \typewriter { mensural } \restsA \bar "" \break

  \override Staff.Rest.style = #'neomensural
  <>^\markup \typewriter { neomensural } \restsA \bar "" \break

  \override Staff.Rest.style = #'classical
  <>^\markup \typewriter { classical } \restsB \bar "" \break

  \override Staff.Rest.style = #'z
  <>^\markup \typewriter { z-style } \restsB \bar "" \break

  \override Staff.Rest.style = #'default
  <>^\markup \typewriter { default } \restsB \bar "" \break
}
```



See also

Music Glossary: Section “breve” in *Music Glossary*, Section “longa” in *Music Glossary*, Section “maxima” in *Music Glossary*.

Notation Reference: Section 2.2.3 [Full measure rests], page 71.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “Rest” in *Internals Reference*.
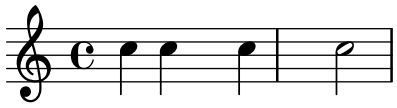
Known issues and warnings

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 1024th to maxima (8× whole).

2.2.2 Invisible rests

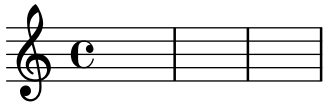
There are two forms of invisible rests: the *spacer rest* named ‘s’, and the `\skip` command. The spacer rest is a note that does not produce output. Like any other note or rest, its duration sets the default duration of following notes.

```
\relative c'' {
  c4 c s c |
  s2 c |
}
```
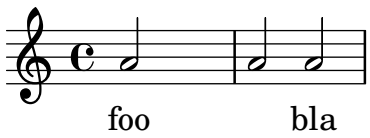


Also like other notes and rests, it implicitly causes `Staff` and `Voice` contexts to be created if none exist.

```
{ s1 s s }
```



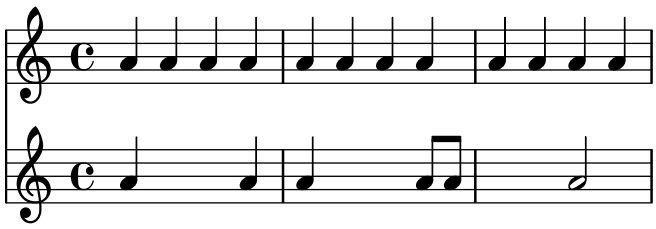
Spacer rests are available only in note mode and chord mode. In other situations, for example, when entering lyrics, the command `\skip` is used to skip a musical moment. The `\skip` command accepts either an explicit duration or a piece of music as an argument and skips the duration of the argument. The duration of the `\skip` is ignored if lyrics derive their durations from the notes in an associated melody through `\addlyrics` or `\lyricsto`.

```
<<
  {
    a'2 \skip2 a'2 a'2
  }
  \new Lyrics {
    \lyricmode {
      foo2 \skip 1 bla2
    }
  }
>>
```



When the argument to `\skip` is music, the default duration of the following note is implicitly set by the last note of the argument. However, to preserve backward compatibility with the legacy implementation of `\skip`, a numeric duration argument does not affect the duration of the subsequent note.

```
<<
  {
    \*12 a'4
  }
  {
    a'4 \skip 2 a' |
    a'4 \skip { a'8 a' a' a' } a' a' |
    s2 a'
  }
>>
```



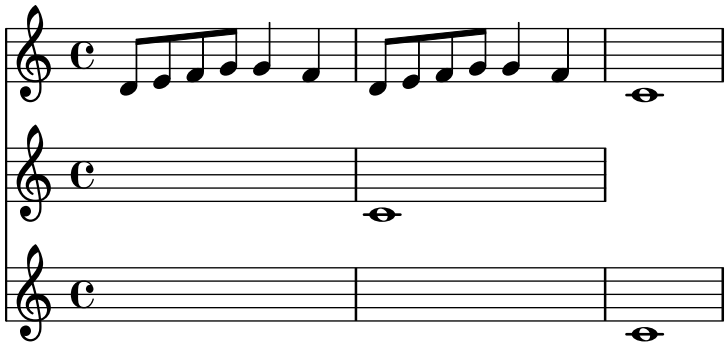
The `\skip` command preserves the effect of an enclosing `unfoldRepeats` command, unlike the `skip-of-length` Scheme function.

```
MyCadenza = \fixed c' {
  \repeat volta 2 {
    d8 e f g g4 f4
  }
}

music = <<
```

```
  \new Staff {
    \MyCadenza
    c'1
  }
  \new Staff {
    #(skip-of-length MyCadenza)
    c'1
  }
  \new Staff {
    \skip \MyCadenza
    c'1
  }
>>

\unfoldRepeats \music
```



The \skip command 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: Section 7.1.4 [Hidden notes], page 289, Section 36.7 [Visibility of objects], page 784.

Snippets: Section "Rhythms" in *Snippets*.

Internals Reference: Section "SkipMusic" in *Internals Reference*.

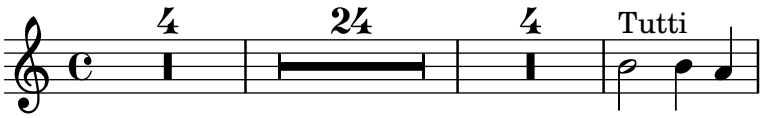
Known issues and warnings

Because duration and music arguments to the \skip command affect the duration of subsequent music differently, it is good practice to provide an explicit duration for the music immediately following the command.

2.2.3 Full measure rests

Rests for one or more full measures are entered like notes with the note name uppercase 'R'. Their duration is entered identically to the duration notation used for notes, including the ability to use duration multipliers, as explained in Section 2.1.3 [Scaling durations], page 61:

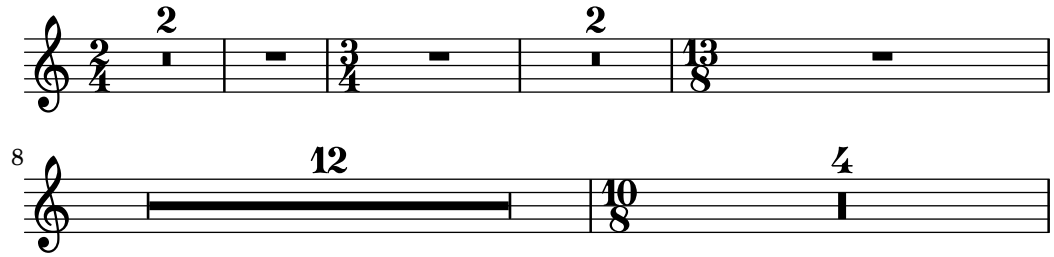
```
% Rest measures contracted to single measure
\compressMMRests {
  R1*4
  R1*24
  R1*4
  b'2^"Tutti" b'4 a'4
}
```



The example above also demonstrates how to compress multiple empty measures, as explained in Section 6.3.4 [Compressing empty measures], page 274.
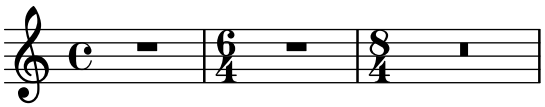
The duration of a multi-measure rest must always be equal to the length of one or several measures. Therefore, some time signatures require the use of augmentation dots or fractions:

```
\compressMMRests {
  \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 |
}
```



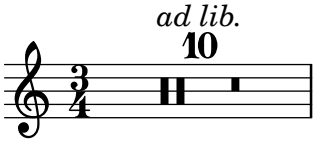
A full-measure rest is printed as either a whole or breve rest, centered in the measure, depending on the time signature.

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



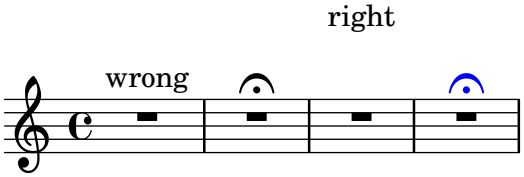
Markups can be added to multi-measure rests.

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



Note: Markups and articulations attached to multi-measure rests are `MultiMeasureRestText` and `MultiMeasureRestScript` types, not `TextScript` and `Script`. 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
\override Script.color = #blue
R1^"wrong"
R1\fermata
% This is the correct object name to be specified
\override MultiMeasureRestText.padding = 5
\override MultiMeasureRestScript.color = #blue
R1^"right"
R1\fermata
```



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

Predefined commands

`\textLengthOn`, `\textLengthOff`, `\compressMMRests`.

Selected snippets

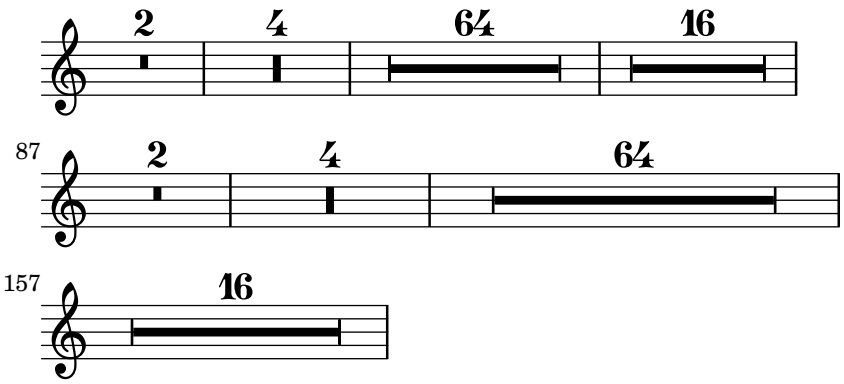
Multi-measure rest length control

Multi-measure rests have a length according to their total duration, which is under the control of the `space-increment` property of the `MultiMeasureRest` grob; its default value is 2.

```
\relative c' {
  \omit Staff.TimeSignature
  \compressEmptyMeasures

  R1*2 R1*4 R1*64 R1*16 \break
  \override MultiMeasureRest.space-increment = 4
  R1*2 R1*4 R1*64 R1*16
}
```

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



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, multi-measure rests in odd-numbered and even-numbered voices are vertically separated in polyphonic music.

This snippet shows how positioning of multi-measure rests can be controlled.

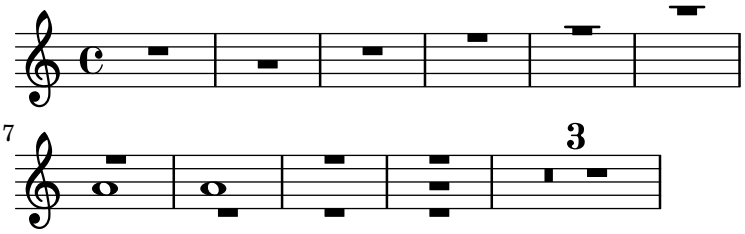
```
\relative c'' {
  % Multi-measure rests by default are set under the fourth line.
  R1
  % They can be moved using an override or tweak.
  \tweak staff-position -2 R1
  \tweak staff-position 0 R1
  \tweak staff-position 2 R1
  \override MultiMeasureRest.staff-position = 3 R1
  \override MultiMeasureRest.staff-position = 6 R1
  \revert MultiMeasureRest.staff-position
  \break

  % Odd-numbered voices are under the top line.
  << { R1 } \\ { a1 } >>
  % Even-numbered voices are under the bottom line.
  << { a1 } \\ { R1 } >>
  % Multi-measure rests in both voices remain separate.
  << { R1 } \\ { R1 } >>

  % Separating multi-measure rests in more than two voices
  % requires an override or tweak.
  << { R1 } \\ { R1 } \\ { \tweak staff-position -2 R1 } >>

  % Using compressed bars in multiple voices requires another override
  % in all voices to avoid multiple instances being printed.
  \compressMMRests
  <<
    \revert MultiMeasureRest.direction
    { R1*3 } \\
    \revert MultiMeasureRest.direction
    { R1*3 }
```

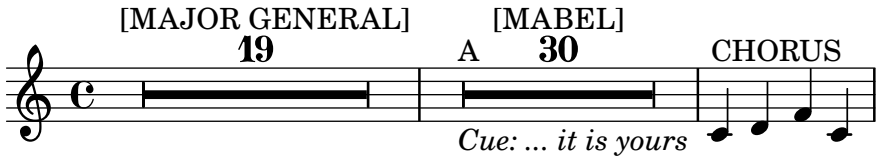
```
  >>
}
```



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 an empty chord with an attached markup before the multi-measure rest. 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' {
  \compressMMRests {
    \textLengthOn
    <>^\markup { [MAJOR GENERAL] }
    R1*19
    <>_\markup { \italic { Cue: ... it is yours } }
    <>^\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: Section 2.1.1 [Durations], page 52, Section 2.1.3 [Scaling durations], page 61, Section 6.3.4 [Compressing empty measures], page 274, Chapter 8 [Text], page 308, Section 8.2 [Formatting text], page 320, Section 8.1.2 [Text scripts], page 310.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “MultiMeasureRest” in *Internals Reference*, Section “MultiMeasureRestNumber” in *Internals Reference*, Section “MultiMeasureRestScript” in *Internals Reference*, Section “MultiMeasureRestText” in *Internals Reference*.

Known issues and warnings

Fingerings over multi-measure rests (e.g., `R1*10-4`) may result in the fingering numeral colliding with the bar counter numeral.

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.

2.2.4 Caesuras

The \caesura command calls for unmetered silence: typically, a short break in sound that does not shorten the previous note.

```
\fixed c'' { c2. \caesura d4 }
```



In chants and hymns, \caesura can serve more generally as a phrase division; for more information, see the references at the end of this section. For a break in sound that shortens the previous note, see Section 3.2.3 [Breath marks], page 176.

Articulations may follow \caesura to indicate the relative duration or significance of the break; these create CaesuraScript grobs.

```
\fixed c'' { c2. \caesura \fermata d4 }
```



By default, \caesura creates a BreathingSign grob. The breath element of the caesuraType context property controls which of several predefined signs \caesura creates. See Section B.14 [List of breath marks], page 931.
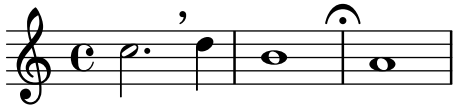
```
\fixed c'' {
  \set Score.caesuraType = #'((breath . curvedcaesura))
  c2. \caesura d4
}
```



To designate one or more CaesuraScript grobs to be created as a normal part of an unarticulated caesura, set the scripts element of the caesuraType context property. (Additional scripts can still be attached as articulations.) In conjunction with the breath element, the scripts listed in the script element attach to the BreathingSign; otherwise, if a BarLine is present, they attach to it.

The caesuraTypeTransform context property can be set to a Scheme function to enable a degree of automatic adaptation. The at-bar-line-substitute-caesura-type function generator supports styles where the notation differs at a bar line.

```
\fixed c' {
  \set Score.caesuraType =
    #'((breath . spacer)
       (scripts . (outsidecomma)))
  \set Score.caesuraTypeTransform =
    #(at-bar-line-substitute-caesura-type
      '((scripts . (fermata))))
  c'2. \caesura d'4
  b1 \caesura
  a1
}
```



Predefined commands

`\caesura`.

Selected snippets

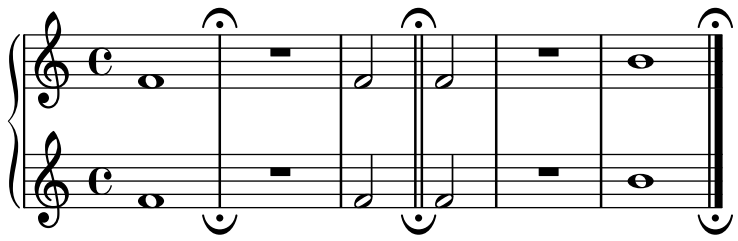
Positioning opposing fermatas on a bar line

This snippet demonstrates a command that prints fermatas both above and below a bar line. If there would not otherwise be a bar line, it adds a double bar line. Semantically, the command codes a longer-than-normal caesura, which might be considered misuse depending on the situation.

```
twoWayFermata = {
  \once \set Staff.caesuraType = #'((underlying-bar-line . "||"))
  \once \set Staff.caesuraTypeTransform = ##f
  \caesura ^\fermata _\fermata
}

music = {
  f'1 \twoWayFermata
  R1
  f'2 \twoWayFermata f'2
  R1
  b'1 \twoWayFermata \fine
}

\new GrandStaff <<
  \new Staff \music
  \new Staff \music
>>
```



See also

Music Glossary: Section “caesura” in *Music Glossary*.

Notation Reference: Section 3.2.3 [Breath marks], page 176, Section 17.4.4 [Divisiones], page 556, Section 17.5.5 [Kievan bar lines], page 566, Section 9.7.4 [Phrase bar lines in hymn tunes], page 409.

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

Internals Reference: Section “BreathingSign” in *Internals Reference*, Section “Caesura_engraver” in *Internals Reference*, Section “CaesuraEvent” in *Internals Reference*, Section “CaesuraScript” in *Internals Reference*, Section “Tunable context properties” in *Internals Reference*.

2.3 Displaying rhythms

2.3.1 Measures

Dividing music into numbered measures (regularly or irregularly) and tracking time within measures are duties of the Timing_translator. This translator is not directly responsible for any engraving, but it sets context properties that guide engravers and performers.

The context in which the Timing_translator is placed, which is Score by default, is assigned the alias Timing. If you add Timing_translator to another context (such as Voice or Staff), you may need to instantiate that context explicitly before using the Timing alias to address properties.

The Timing_translator uses the following properties of the Timing context to keep track of time and describe it to engravers and performers.

currentBarNumber
: The current measure number. For an example showing the use of this property see Section 2.5.3 [Bar numbers], page 133.

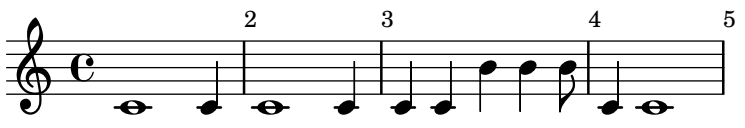
measureLength
: The length of the current measure. For example, \time 4/4 sets this to 1, \time 6/8 sets it to 3/4, and \senzaMisura makes it infinite. This property may be set manually to create a shorter or longer measure than usual, but it cannot reasonably be set less than or equal to measurePosition, since the opportunity to start a new measure at the correct time is past. Changing the measure length with \measure, \measureRemainder, \setMeasureLengthFromHere, or \setDefaultMeasureLength is recommended over setting this property explicitly.

measurePosition
: This internal property is the current point in the current measure. When it reaches measureLength, it is reset to zero and currentBarNumber is incremented. measurePosition should not be set explicitly, but may be changed with the \partial or \premeasure commands.

timing
: When set to #t, measurePosition is updated at every note, rest, or skip. When set to #f, measurePosition keeps its current value indefinitely.

Timing can be changed by setting any of these variables. 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 \partial command advances the measure position to leave only 5/8 remaining, which shortens that bar by 1/8. The next bar line then falls at 9/8 rather than 5/4.

```
\fixed c' {
  \override Score.BarNumber.break-visibility = #all-visible
  \set Timing.measureLength = #5/4
  c1 c4
  c1 c4
  c4 c \partial 8*5 b4 b b8
  c4 c1
}
```



Selected snippets

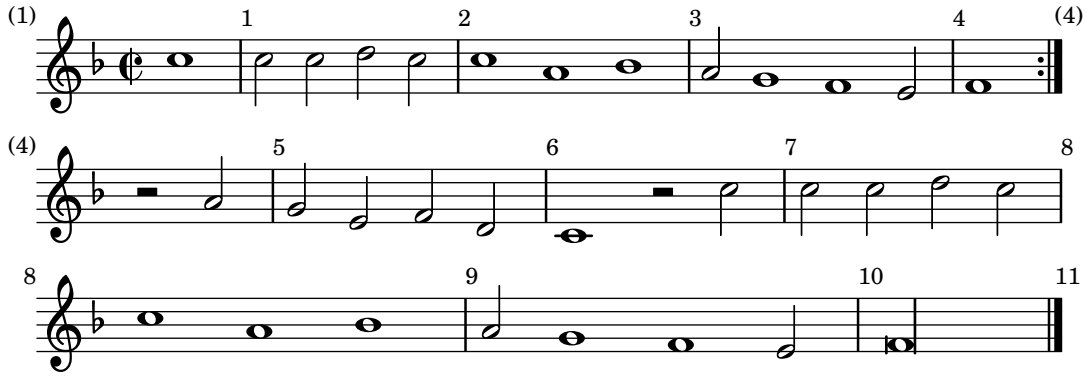
Irregular measures

"Es ist ein Ros entsprungen" was first published without bar lines in *Musae Sioniae* (1609). This example adds irregular measures to the melody.

The `\measure` command works for complete measures. The full length of the incomplete measure at the end of the repeated section is specified with `\setMeasureLengthFromHere` to preserve correctness if the repeat were unfolded. In isolation, `\setMeasureLengthFromHere` would normally be followed at the end of the measure by `\setDefaultMeasureLength` to revert to the time signature's measure length, but that is unnecessary in this case because the following measure is also irregular.

```
\layout {
  \verboseBarNumbers
}

\fixed c' {
  \time 2/2
  \key f \major
  \repeat volta 2 {
    \premeasure { c'1 }
    \measure { c'2 c' d' c' }
    \measure { c'1 a bes }
    \measure { a2 g1 f e2 }
    \setMeasureLengthFromHere 1*2
    f1
    \break
  }
  r2 a |
  \measure { g2 e f d }
  \measure { c1 r2 c' }
  \measure { c'2 c' d' c' }
  \measure { c'1 a bes }
  \measure { a2 g1 f e2 }
  \measure { f\longa }
  \fine
}
```



See also

Notation Reference: Section 2.5.3 [Bar numbers], page 133, Section 2.3.5 [Unmetered music], page 93.

Snippets: Section “Rhythms” in *Snippets*.
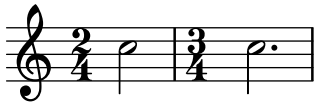
Internals Reference: Section “Timing_translator” in *Internals Reference*, Section “Score” in *Internals Reference*.

2.3.2 Time signature

To set a basic time signature, use the `\time` command with the fraction as argument.

```
\time 2/4
c''2
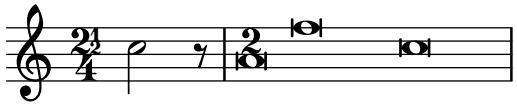
\time 3/4
c''2.
```



Fractional time signatures and denominators longer than a whole note require Scheme syntax.
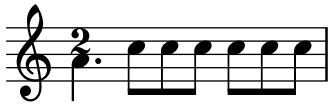
```
\time #'(5/2 . 4)
c''2 r8

\override Timing.TimeSignature.denominator-style = #'note
\time #'(2 . 1/2)
f''\breve c''
```



Other unusual denominators may be used to add augmentation dots in the number-over-note style; however, to benefit the most from features covered elsewhere, using a conventional fraction for `\time` and overriding `TimeSignature.time-signature` is recommended in such cases.

```
\override Timing.TimeSignature.denominator-style = #'note
\once \override Timing.TimeSignature.time-signature = #'(2 . 8/3)
\time 6/8
c''8 8 8 8 8 8
```

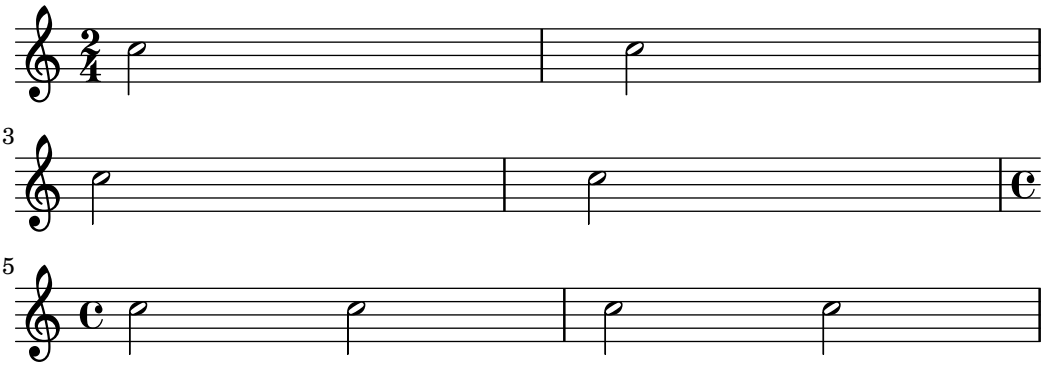


Mid-measure time signature changes are covered in Section 2.3.4 [Upbeats], page 92.

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 36.7 [Visibility of objects], page 784.
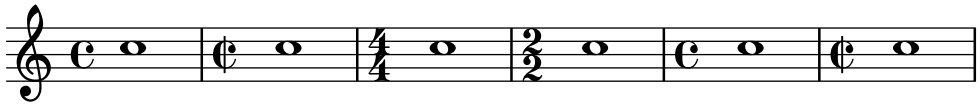
```
\relative c'' {
  \time 2/4
  c2 c
  \break
  c c
  \break
  \time 4/4
  c c c c
```

```
}
```



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

```
\relative c'' {
  % 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
}
```
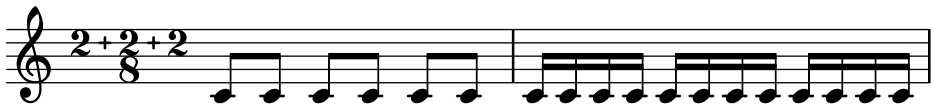


Mensural time signatures are covered in Section 17.3.3 [Mensural time signatures], page 544.

Complex time signatures
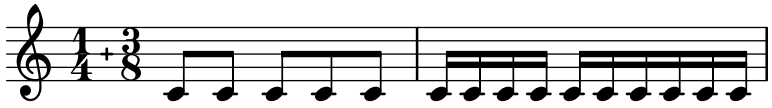
The `\time` function supports a variety of complex time signatures represented as Scheme expressions. Fractions are represented as pairs, (*numerator . denominator*), where the denominator is always a number. The numerator is one number or a list of two or more numbers. A list represents concatenation.

```
\fixed c' {
  \time #'((2 2 2) . 8)
  \*6 c8
  \*12 c16
}
```
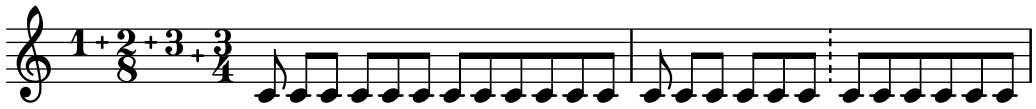


Meters that alternate strictly through multiple fractions can be constructed by grouping the fractions into a list. Automatic beaming settings will be adjusted depending on the values. Submeasure bar lines can be enabled to help clarify complex rhythms.

```
\fixed c' {
  \time #'((1 . 4) (3 . 8))
  \*5 c8
  \*10 c16
}
```
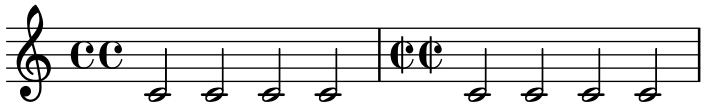


```
\fixed c' {
  \time #'(((1 2 3) . 8) (3 . 4))
  \*12 c8
  \submeasureBarsOn
  \*12 c8
}
```



By default, strictly alternating meters consisting of 4/4 and 2/2 are printed in a historic style using multiple C or cut-C symbols. It is not necessary to use this interpretation of these time signatures for the underlying meter: it is also possible to override `TimeSignature.time-signature` to print these symbols while setting a different meter with `\time`.

```
\fixed c' {
  \time #'((4 . 4) (4 . 4))
  \*4 c2
  \once \override Timing.TimeSignature.time-signature
    = #'((2 . 2) (2 . 2))
  \time 4/2
  \*4 c2
}
```
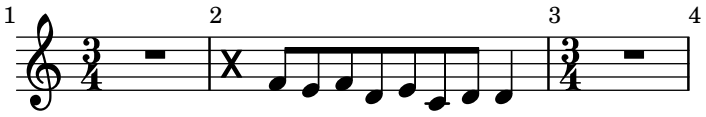


Free time

Meter can be canceled with the `\senzaMisura` command, allowing unstructured measures of arbitrary length. With no meter in effect, measures must be ended manually. See Section 2.3.1 [Measures], page 78.

By default, nothing is printed at a transition into free time. To allow printing an 'X' in place of a time signature, use the `\senzaMisuraTimeSignatureX` command. To revert to the default behavior, use the `\senzaMisuraTimeSignatureOff` command.

```
\fixed c' {
  \verboseBarNumbers
  \senzaMisuraTimeSignatureX
  \time 3/4
  R2.
  \senzaMisura
  \measure { f8 e f d e c d d4 }
  \time 3/4
```

```
  R2.
}
```
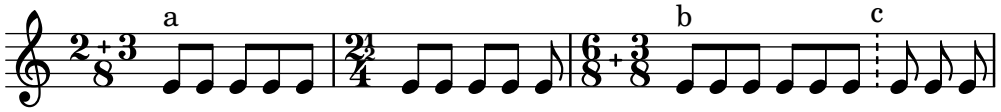


Metric properties

In addition to setting the printed time signature, the `\time` and `\senzaMisura` commands also set the values of the context properties `beatBase`, `beatStructure`, `submeasureStructure`, and `beamExceptions`.

The default values of these properties are derived formulaically from the time signature, with various exceptions defined in `scm/time-signature-settings.scm`. Setting aside the exceptions,
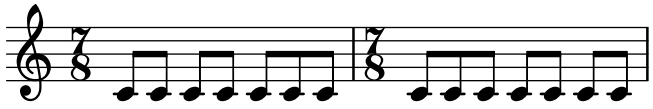
- `beatBase` is one over the denominator of the time signature. For example, in 3/2 time, `beatBase` is 1/2.
- `beatStructure` divides the measure
  - according to a structured numerator (a), or
  - into beats of 3 × `beatBase` when the numerator is 6, 9, 12, ... (b), or
  - into beats of 1 × `beatBase`.
- `submeasureStructure` divides the measure between strictly alternating time signatures (c).



`\senzaMisura` makes `beatBase` infinite and the structure properties empty.
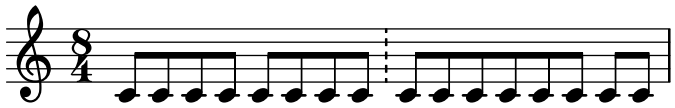
The default value of `beatStructure` can be overridden in the `\time` command itself by supplying it as the optional first argument:

```
\score {
  \new Staff {
    \relative {
      \time 2,2,3 7/8
      \*7 c'8 |
      \time 3,2,2 7/8
      \*7 c8 |
    }
  }
}
```



Alternatively, the default values of these metric properties can be set as a group for a given time signature. The new values take effect at a subsequent `\time` command with that time signature:

```
\score {
  \new Staff {
    \relative c' {
      \overrideTimeSignatureSettings
        8/4             % timeSignature
        #1/4            % beatBase
        #'((2 2) (3 1)) % beatStructure, submeasureStructure
        #'()            % beamExceptions
      \time 8/4
      \submeasureBarsOn
      \*16 c8 |
    }
  }
}
```
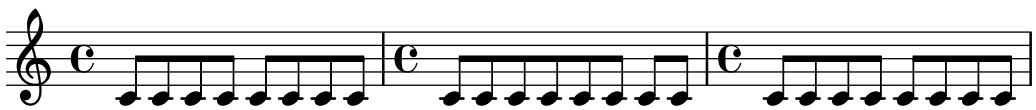


`\overrideTimeSignatureSettings` takes four arguments:

1. *`timeSignature`*, a fraction describing the time signature to which these values apply.
2. *`beatBase`*, the musical length corresponding to one unit of *beatStructure*.
3. *`structure`*, a Scheme list describing the length of each beat in the measure in units of *beatBase* (i.e., `beatStructure`), optionally grouped with inner lists describing larger subdivisions of the measure (i.e., `submeasureStructure`).
4. *`beamExceptions`*, an alist containing any beaming rules for the time signature that go beyond ending at every beat, as described in Section 2.4.2 [Setting automatic beam behavior], page 106.

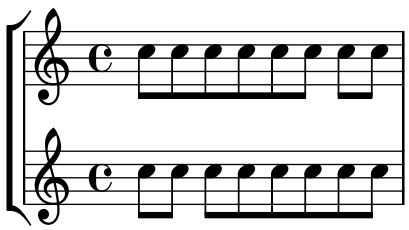
Changed values of default time signature properties can be restored to the original values:

```
\score {
  \relative {
    \*8 c'8 |
    \overrideTimeSignatureSettings
      4/4         % timeSignature
      #1/4        % beatBase
      3,1         % beatStructure
      #'()        % beamExceptions
    \time 4/4
    \*8 c8 |
    \revertTimeSignatureSettings 4/4
    \time 4/4
    \*8 c8 |
  }
}
```



Different values of default time signature properties can be established for different staves by enabling polymetric notation (see Section 2.3.6 [Polymetric notation], page 95).

```
\score {
  \new StaffGroup <<
     \new Staff {
        \overrideTimeSignatureSettings
          4/4        % timeSignature
          #1/4       % beatBase
          3,1        % beatStructure
          #'()       % beamExceptions
        \time 4/4
        \*8 c''8
     }
     \new Staff {
        \overrideTimeSignatureSettings
          4/4        % timeSignature
          #1/4       % beatBase
          1,3        % beatStructure
          #'()       % beamExceptions
        \time 4/4
        \*8 c''8
     }
  >>
  \layout {
    \enablePerStaffTiming
  }
}
```



A further method of changing these metric properties, which avoids reprinting the time signature at the time of the change, is shown in Section 2.4.2 [Setting automatic beam behavior], page 106.

Predefined commands
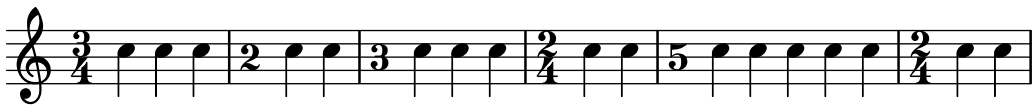
`\numericTimeSignature`, `\defaultTimeSignature`.

Selected snippets

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

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

```
\relative c'' {
  \time 3/4
  c4 c c
  % Change the style permanently
  \override Staff.TimeSignature.style = #'single-number
  \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-number style only for the next time signature
  \tweak style #'single-number \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: Section 2.3.1 [Measures], page 78, Section 17.3.3 [Mensural time signatures], page 544, Section 2.3.6 [Polymetric notation], page 95, Section 2.4.2 [Setting automatic beam behavior], page 106.

Installed Files: `scm/time-signature-settings.scm`.
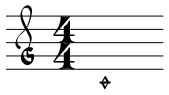
Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “TimeSignature” in *Internals Reference*, Section “Timing_translator” in *Internals Reference*.

Known issues and warnings
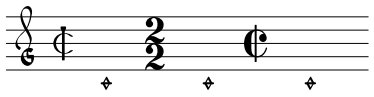
`\numericTimeSignature` and `\defaultTimeSignature` have no effect when used in a `MensuralStaff`. To use these modern time signatures in a `MensuralStaff`, either select them already when creating the context:

```
\new MensuralStaff \with { \numericTimeSignature } {
  c'1
}
```



or use an explicit `\override`:
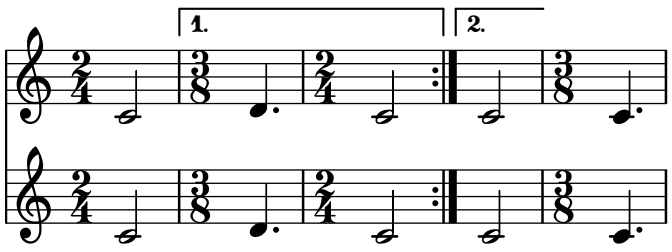
```
\new MensuralStaff {
  \time 2/2
  c'1
  \override MensuralStaff.TimeSignature.style = #'numbered
  \time 2/2
  c'
  \override MensuralStaff.TimeSignature.style = #'default
  \time 2/2
  c'
}
```



If there is more than a single staff, and a time signature starts a prima volta but not the seconda volta, it is necessary to help LilyPond synchronize this situation by adding an explicit but invisible time signature in the seconda volta.

```
music = {
  \repeat volta 2 {
    \time 2/4 c'2 |
    \alternative {
      \volta 1 {
        \time 3/8 d'4. |
        \time 2/4 c'2 | }
      \volta 2 {
        \once \omit Staff.TimeSignature
        \time 2/4 c'2 |
      }
    }
  }
  \time 3/8 c'4. |
}
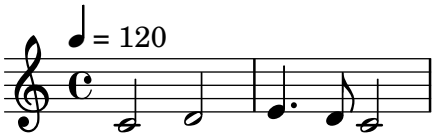
<<
  \new Staff \music
  \new Staff \music
>>
```



2.3.3 Metronome marks
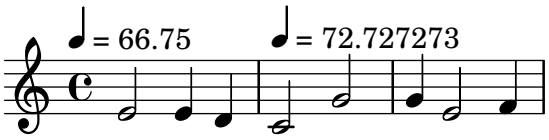
A basic metronome mark is simple to write:

```
\relative {
  \tempo 4 = 120
  c'2 d
  e4. d8 c2
}
```
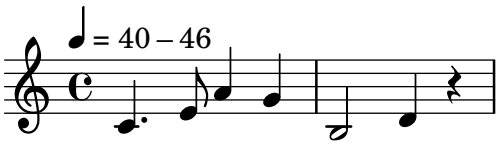


The stated rate does not have to be an integer, but it must be an exact number. The default formatter rounds to the nearest 0.25; the context property `tempoCountPrecision` controls this.

```
tempoI = 100
\fixed c' {
  \tempo 4 = #(* tempoI 2/3)   % 66.66...
  e2 e4 d
  \once \set Timing.tempoCountPrecision = #1/1000000
  \tempo 4 = #(* tempoI 8/11)  % 72.72...
  c2 g2
  g4 e2 f4
}
```
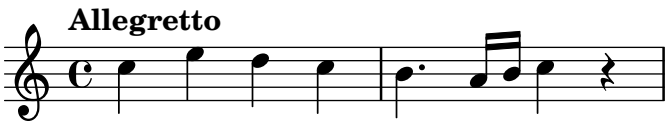


Metronome marks may also be printed as a range of two numbers:

```
\relative {
  \tempo 4 = 40 - 46
  c'4. e8 a4 g
  b,2 d4 r
}
```
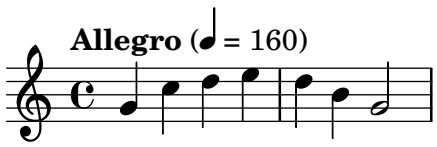


Tempo indications with text can be used instead:

```
\relative {
  \tempo "Allegretto"
    c''4 e d c
    b4. a16 b c4 r4
}
```
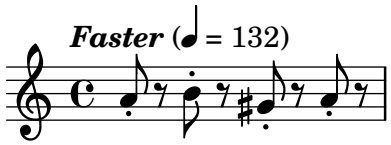


Combining a metronome mark and text will automatically place the metronome mark within parentheses:

```
\relative {
  \tempo "Allegro" 4 = 160
  g'4 c d e
  d4 b g2
}
```
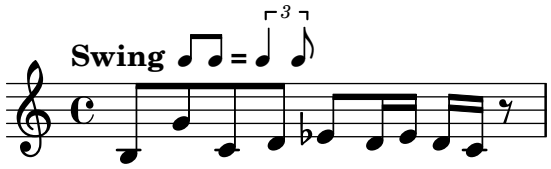


In general, the text can be any markup object:

```
\relative {
  \tempo \markup { \italic Faster } 4 = 132
  a'8-. r8 b-. r gis-. r a-. r
}
```
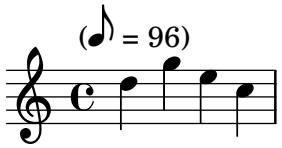


A particularly useful markup command is `\rhythm`, which prints a rhythmic pattern. See Section A.1.4 [Markup for music and musical symbols], page 851.

```
\relative {
  \tempo \markup {
    Swing
    \hspace #0.4
    \rhythm { 8[ 8] } = \rhythm { \tuplet 3/2 { 4 8 } }
  }
  b8 g' c, d ees d16 ees d c r8
}
```



A parenthesized metronome mark with no textual indication may be written by including an empty string in the input:

```
\relative {
  \tempo "" 8 = 96
  d''4 g e c
}
```
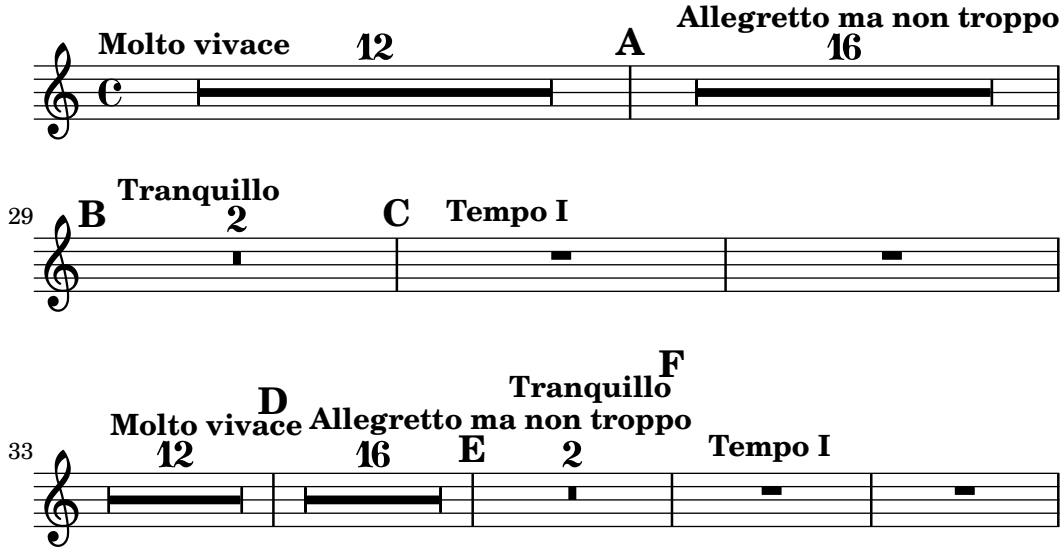


In a part for an instrument with long periods of rests (see Section 2.2.3 [Full measure rests], page 71) it happens quite frequently that tempo indications, rehearsal marks, and text marks sometimes follow each other closely. The command `\markLengthOn` provides extra horizontal

(and vertical) space to prevent such marks from horizontal overlapping, often causing unwanted vertical stacking. Use `\markLengthOff` to restore the default behavior of ignoring these marks for the horizontal spacing algorithm.

```
\compressMMRests {
  \markLengthOn
  \tempo "Molto vivace"
  R1*12
  \mark \default \tempo "Allegretto ma non troppo"
  R1*16
  \mark \default \tempo "Tranquillo"
  R1*2
  \markLengthOff
  \mark \default \tempo "Tempo I"
  R1 R1 \break

  \markLengthOff
  \tempo "Molto vivace"
  R1*12
  \mark \default \tempo "Allegretto ma non troppo"
  R1*16
  \mark \default \tempo "Tranquillo"
  R1*2
  \mark \default \tempo "Tempo I"
  R1 R1
}
```



Selected snippets

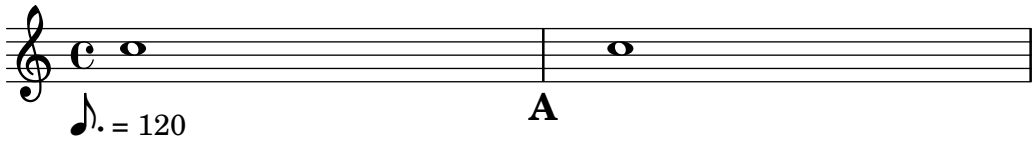
Printing metronome and rehearsal marks below the staff

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
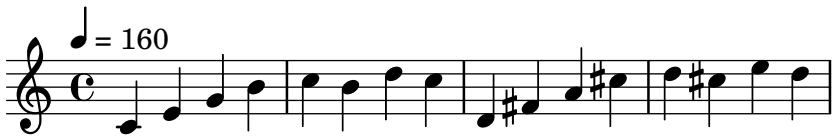
  % Rehearsal marks below the staff
  \override Score.RehearsalMark.direction = #DOWN
  \mark \default
  c''1
}
```



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 { }
}
```

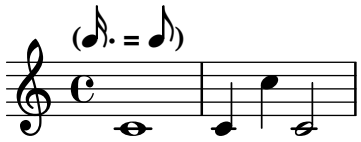


Creating metronome marks in markup mode

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. } #UP
      " = "
```

```
      \smaller \general-align #Y #DOWN \note { 8 } #UP
      )
    }
  }
  c1
  c4 c' c,2
}
```



For more details, see Section 8.2 [Formatting text], page 320.

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 8.2 [Formatting text], page 320, Chapter 24 [Creating MIDI output], page 654, Section 2.2.3 [Full measure rests], page 71.
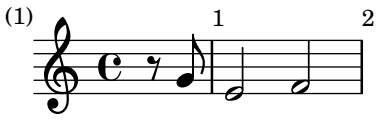
Snippets: Section “Staff notation” in *Snippets*.

Internals Reference: Section “MetronomeMark” in *Internals Reference*.
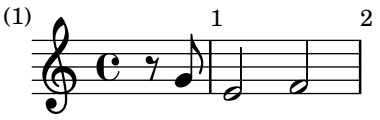
2.3.4 Upbeats

Partial or pickup measures, such as an *anacrusis* or an *upbeat*, are entered using the \partial or \premeasure commands. The two have the same effect, but \partial takes an explicit duration and \premeasure uses the duration of the music expression that follows it. When used at the beginning of a score, these commands align the music to precede the first bar.

```
\fixed c' {
  \verboseBarNumbers
  \partial 4
  r8 g8 | e2 f |
}
```

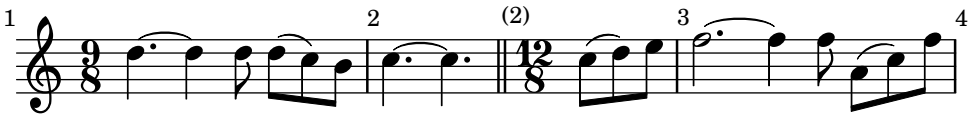



```
\fixed c' {
  \verboseBarNumbers
  \premeasure { r8 g8 }
  e2 f |
}
```
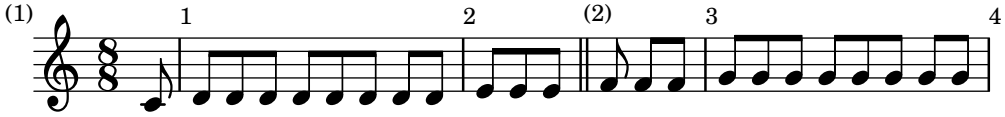


When used after the beginning of a score, these commands end the current measure after the given duration or music.

```
\relative {
  \verboseBarNumbers
  \time 9/8
  d''4.~ 4 d8 d( c) b
  c4.~ 4. \section
  \time 12/8
  \premeasure { c8( d) e }
  f2.~ 4 f8 a,( c) f
}
```



A \partial or \premeasure command is *required* when the time signature changes in mid measure, but they may also be used alone. In any case, notational elements that communicate metric structure (e.g., beam breaks) are engraved appropriately for the final part of the measure.

```
\fixed c' {
  \verboseBarNumbers
  \time 3,3,2 8/8
  \partial 8
  c8 | d d d d d d d d | e e e \section
  \partial 4.
  f8 f f | g g g g g g g g |
}
```



For technical reasons, the arguments to \partial and \premeasure must have a non-zero duration.

See also

Music Glossary: Section “anacrusis” in *Music Glossary*.

Notation Reference: Section 2.6.1 [Grace notes], page 148.
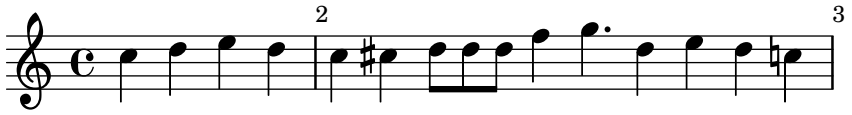
Snippets: Section “Rhythms” in *Snippets*.

Internal Reference: Section “Timing_translator” in *Internals Reference*.

2.3.5 Unmetered music

In music such as cadenzas, it may be desirable to disable automatic measure demarcation and all that it entails: numbering bars, resetting accidentals, etc. Music between \cadenzaOn and \cadenzaOff does not count toward the length of a measure.
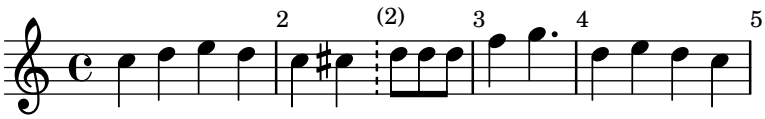
```
\relative c'' {
  % Show all bar numbers
  \override Score.BarNumber.break-visibility = #all-visible
  c4 d e d
  \cadenzaOn
  c4 cis d8[ d d] f4 g4.
  \cadenzaOff
  d4 e d c
}
```



To divide an unmetered passage into irregular measures, temporarily re-enable timing and use \partial to create a tiny measure. The \bar command alone does not start a new measure.
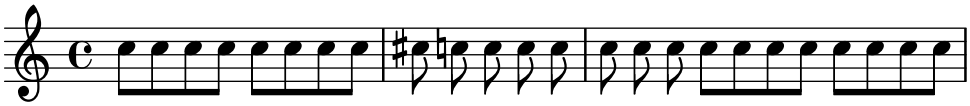
```
cadenzaMeasure = {
  \cadenzaOff
  \partial 1024 s1024
  \cadenzaOn
}

\relative c'' {
  % Show all bar numbers
  \override Score.BarNumber.break-visibility = #all-visible
  c4 d e d
  \cadenzaOn
  c4 cis \bar "!" d8[ d d] \cadenzaMeasure f4 g4.
  \cadenzaMeasure
  \cadenzaOff
  d4 e d c
}
```



Automatic beaming is disabled by `\cadenzaOn`. Therefore, all beaming in cadenzas must be entered manually. See Section 2.4.3 [Manual beams], page 115.

```
\relative {
  \*8 c''8
  \cadenzaOn
  cis8 c c c c
  \bar"|"
  c8 c c
  \cadenzaOff
  \*8 c8
}
```
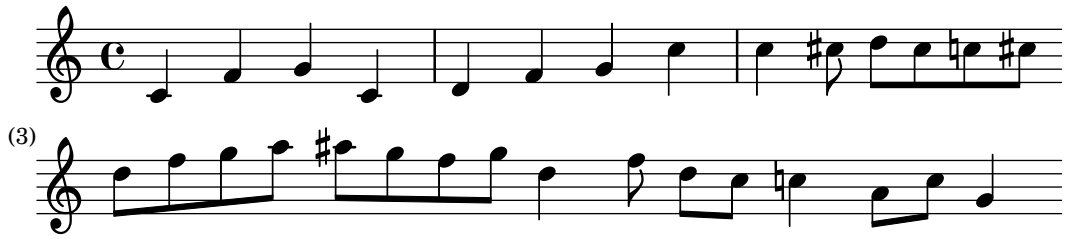


These predefined commands affect all staves in the score, even when placed in just one `Voice` context. To change this, move the `Timing_translator` from the `Score` context to the `Staff` context. See Section 2.3.6 [Polymetric notation], page 95.

Within a cadenza section, automatic breaks are disabled: since there is no metric, it is not possible to determine automatically where they would be appropriate. Therefore, in a long cadenza passage, you must insert possible break points at appropriate places using the `\allowBreak` command or other solutions in Section 28.1 [Line breaking], page 689.

```
\relative {
  c'4 f g c, d f g c
  \cadenzaOn
  c4 cis8
  \allowBreak
  d[ cis c cis]
  \allowBreak
  d[ f g a]
```

```
  \allowBreak
  ais[ g f g]
  \allowBreak
  d4 f8
  \allowBreak
  d[ cis] c4
  \allowBreak
  a8[ c] g4
}
```



Predefined commands

`\cadenzaOn`, `\cadenzaOff`.

See also

Music Glossary: Section “cadenza” in *Music Glossary*.

Notation Reference: Section 36.7 [Visibility of objects], page 784, Section 2.3.6 [Polymetric notation], page 95, Section 2.4.3 [Manual beams], page 115, Section 1.1.3 [Accidentals], page 8.

Snippets: Section “Rhythms” in *Snippets*.

2.3.6 Polymetric notation

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

Different time signatures with equal-length measures

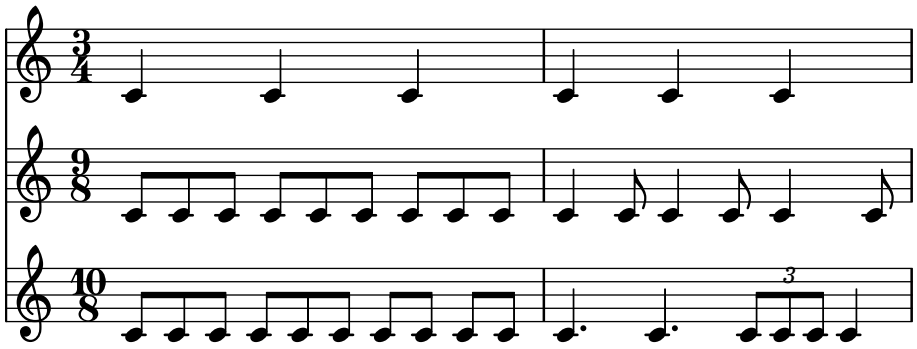
Use `\polymetric \time` to set a local time signature and `\scaleDurations` to make its measure length match `Timing.measureLength`. This sets the same set of properties as the `\time` command, except for `measureLength`.

To unset the local properties and resume using the values from the `Timing` context, use `\polymetric \default`.

In the following example, staves with time signatures 3/4, 9/8, and 10/8 are used in parallel, with 3/4 chosen as the reference measure length. The music in the first staff is spaced normally. In the second staff, the notated durations are multiplied by 2/3 (because 2/3 * 9/8 = 3/4) and in the third staff, the notated durations are multiplied by 3/5 (because 3/5 * 10/8 = 3/4).

```
\relative <<
  \new Staff {
    \time 3/4
    c'4 c c |
    c4 c c |
  }
  \new Staff {
    \scaleDurations 2/3 {
      \context Staff \polymetric \time 9/8
      \*3 { c8 c c }
```

```
      \*3 { c4 c8 }
    }
  }
  \new Staff {
    \scaleDurations 3/5 {
      \context Staff \polymetric \time 3,3,2,2 10/8
      \*2 { c8 c c }
      \*2 { c8 c } |
      c4. c \tuplet 3/2 { c8 c c } c4
    }
  }
>>
```



Different time signatures with unequal-length measures

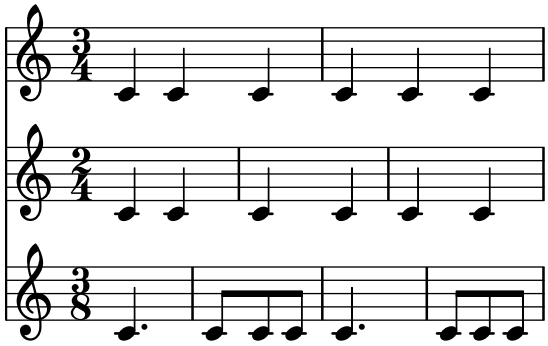
Each staff can be given its own independent time signature as soon as `\enablePerStaffTiming` is placed in the `\layout` block.

```
\layout {
  \enablePerStaffTiming
}

% Now each staff has its own time signature.

\relative <<
  \new Staff {
    \time 3/4
    c'4 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 |
  }
```
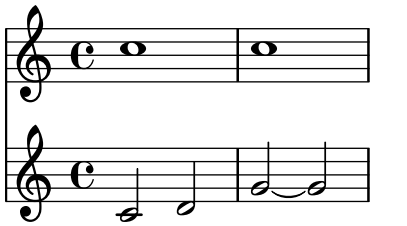
```
>>
```

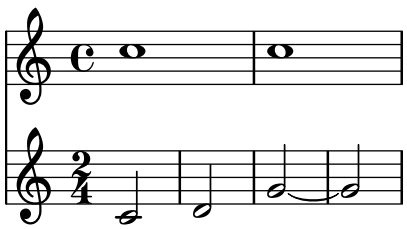


To have just one polymetric score, include `\enablePerStaffTiming` in a `\layout` block inside the `\score` block.

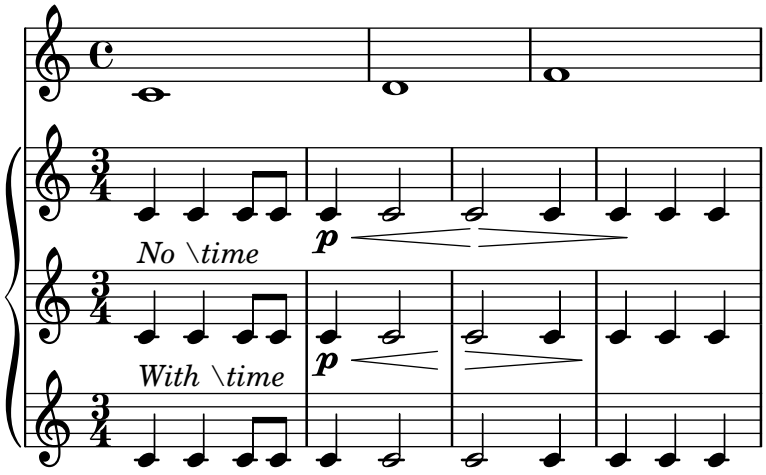
```
\score {
  <<
    \new Staff { c''1 1 }
    \new Staff { c'2 d' g'2~ 2 }
  >>
}

\score {
  \layout {
    \enablePerStaffTiming
  }
  <<
    \new Staff { \time 4/4 c''1 1 }
    \new Staff { \time 2/4 c'2 d' g'2~ 2 }
  >>
}
```





When using polymeter, all staves should include a `\time` command if their meter is not the default 4/4. This is true even for special staves without actual staff lines, such as `Dynamics` contexts, since the placement of certain spanners like hairpins is synchronized with bar lines.



In order to use this feature with MIDI output, also include `\enablePerStaffTiming` in a `\midi` block.

```
\layout {
  \enablePerStaffTiming
}

\midi {
  \enablePerStaffTiming
}
```

See also

Music Glossary: Section “polymetric” in *Music Glossary*, Section “polymetric time signature” in *Music Glossary*, Section “meter” in *Music Glossary*.

Notation Reference: Section 2.4.1 [Automatic beams], page 102, Section 2.4.3 [Manual beams], page 115, Section 2.3.2 [Time signature], page 80, Section 2.1.3 [Scaling durations], page 61.

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “TimeSignature” in *Internals Reference*, Section “Timing_translator” in *Internals Reference*, Section “Staff” in *Internals Reference*.
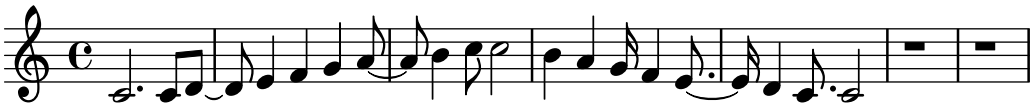
Known issues and warnings

Although notes that occur at the same moment in each of the different staves will be placed at the same horizontal location, bar lines (in each staff) may cause inconsistent spacing within each of the different time signatures.

2.3.7 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`. Similarly, long rests which overrun bar lines are split automatically by replacing the `Rest_engraver` with the `Completion_rest_engraver`. In the following example, notes and rests crossing the bar lines are split, notes are also tied.
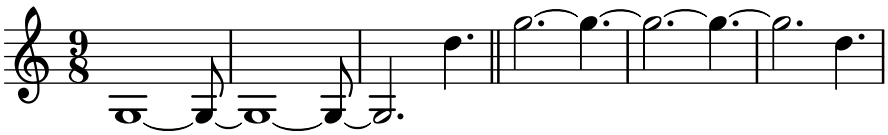
```
\new Voice \with {
  \remove Note_heads_engraver
  \consists Completion_heads_engraver
  \remove Rest_engraver
  \consists Completion_rest_engraver
}
\relative {
  c'2. c8 d4 e f g a b c8 c2 b4 a g16 f4 e d c8. c2 r1*2
}
```



These engravers split all running notes and rests at the bar line, and inserts ties for notes. 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.
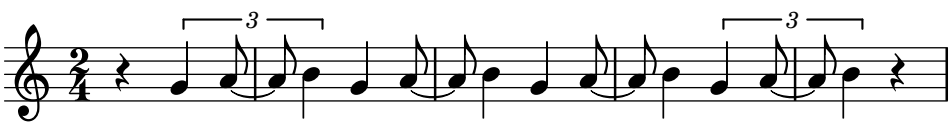
The property `completionUnit` sets a preferred duration for the split notes.

```
\new Voice \with {
  \remove Note_heads_engraver
  \consists Completion_heads_engraver
} \relative {
  \time 9/8 g\breve. d''4. \bar "||"
  \set completionUnit = #3/8
  g\breve. d4.
}
```



These engravers split notes with scaled duration, such as those in tuplets, into notes with the same scale factor as in the input note.

```
\new Voice \with {
  \remove Note_heads_engraver
  \consists Completion_heads_engraver
} \relative {
  \time 2/4 r4
  \tuplet 3/2 {g'4 a b}
  \scaleDurations 2/3 {g a b}
  g4*2/3 a b
  \tuplet 3/2 {g4 a b}
  r4
}
```



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 “Rest_engraver” in *Internals Reference*, Section “Completion_rest_engraver” in *Internals Reference*, Section “Forbid_line_break_engraver” in *Internals Reference*.

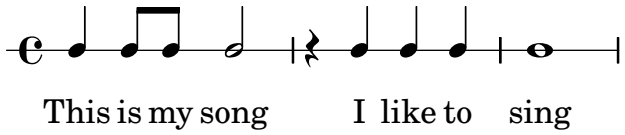
Known issues and warnings

For consistency with previous behavior, notes and rests with duration longer than a measure, such as `c1*2`, are split into notes without any scale factor, `{ c1 c1 }`. The property `completionFactor` controls this behavior, and setting it to `#f` cause split notes and rests to have the scale factor of the input durations.

2.3.8 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

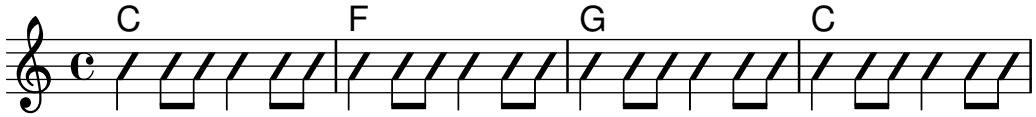
```
<<
  \new RhythmicStaff {
    \new Voice = "myRhythm" \relative {
      \time 4/4
      c'4 e8 f g2
      r4 g g f
      g1
    }
  }
  \new Lyrics {
    \lyricsto "myRhythm" {
      This is my song
      I like to sing
    }
  }
>>
```



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
  }
>>
```



Music containing chords can also be used as input to `RhythmicStaff` and for use with the `Pitch_squash_engraver` if the chords are first reduced to single notes with the `\reduceChords` music function:

```
\new RhythmicStaff {
  \time 4/4
  \reduceChords {
    <c>2
    <e>2
    <c e g>2
    <c e g>4
    <c e g>4
  }
}
```



Predefined commands

`\improvisationOn`, `\improvisationOff`, `\reduceChords`.

Selected snippets

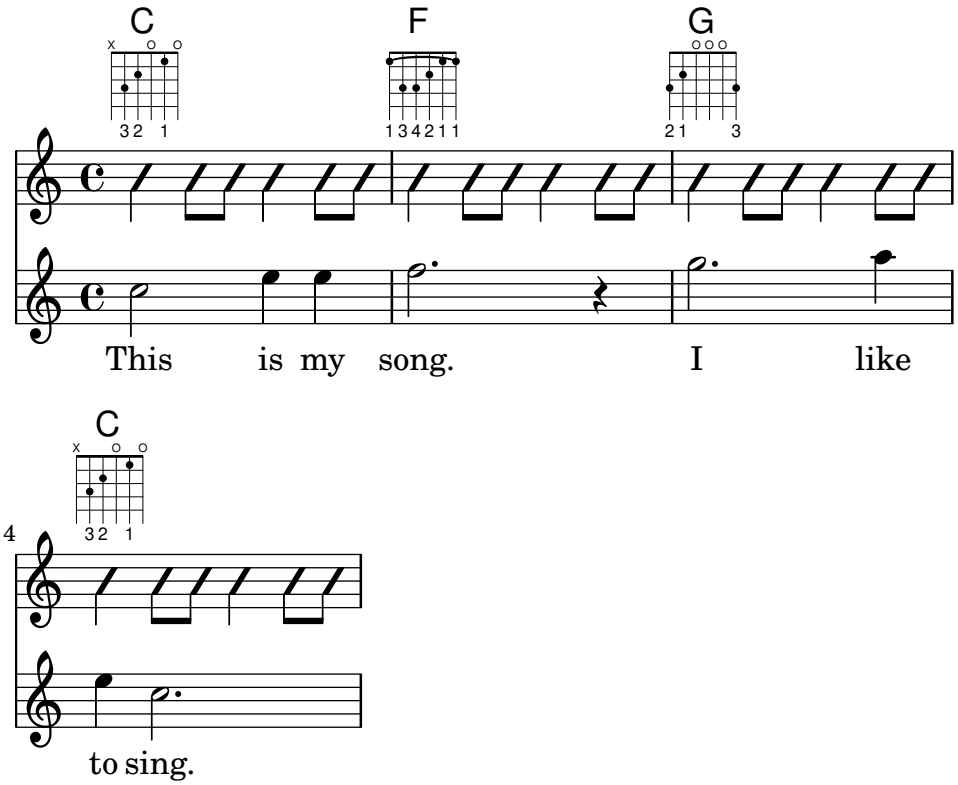
Guitar strum rhythms

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.
  }
>>
```



See also

Snippets: Section “Rhythms” in *Snippets*.

Internals Reference: Section “RhythmicStaff” in *Internals Reference*, Section “Pitch_squash_engraver” in *Internals Reference*.

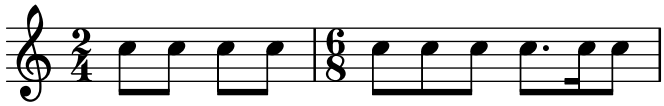
2.4 Beams

LilyPond provides two different possibilities for entering beams: automatic and manual input, which can be also mixed.

2.4.1 Automatic beams

By default, beams are inserted automatically:
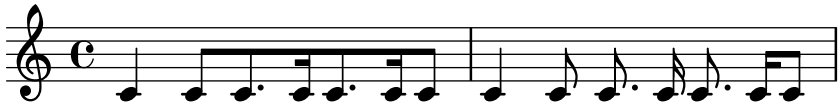
```
\relative c'' {
  \time 2/4 c8 c c c
  \time 6/8 c8 c c c8. c16 c8
}
```



If these automatic decisions are not satisfactory, beaming can be entered explicitly; see Section 2.4.3 [Manual beams], page 115. 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`:

```
\relative c' {
  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. Using `\partCombine` with `\autoBeamOff` can produce unintended results. See the snippets for more information.

Beaming patterns that differ from the automatic defaults can be created; see Section 2.4.2 [Setting automatic beam behavior], page 106.

Predefined commands

`\autoBeamOff`, `\autoBeamOn`.

Selected snippets

Beams across line breaks

Normally, LilyPond refuses to automatically break a line at places where a beam crosses a bar line. This behavior can be changed by setting the `Beam.breakable` property to `#t`.

This property does not affect manual breaks inserted with commands like `\break`.

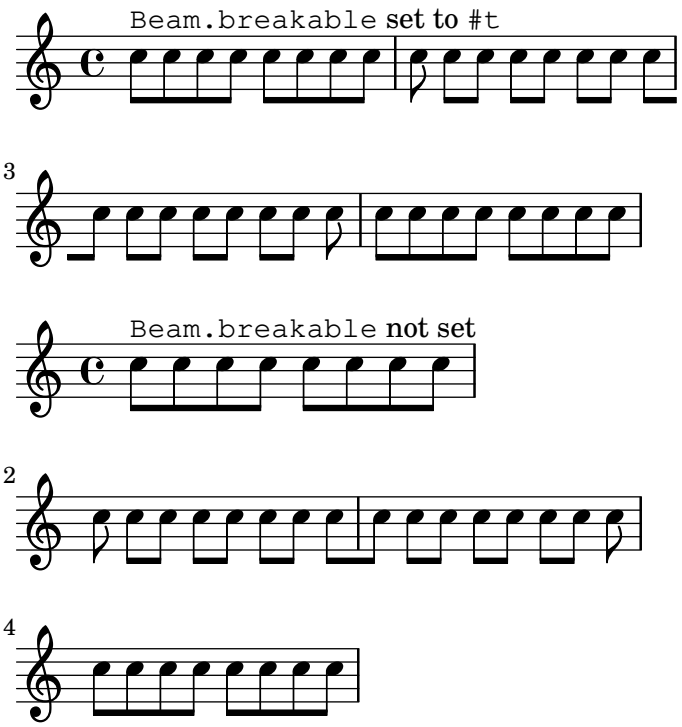
```
music = {
  \*8 c8
  c8 \*7 { c[ c] }  c
  \*8 c8
}

\relative c'' {
  <>^\markup { \typewriter Beam.breakable set to \typewriter "#t" }
```

```
  \override Beam.breakable = ##t
  \music
}

\relative c'' {
  <>^\markup { \typewriter Beam.breakable not set }
  \music
}
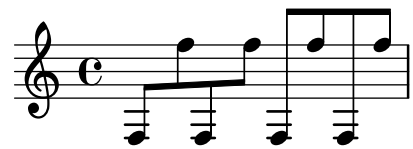
\paper {
  line-width = 100\mm
}
```



Changing beam knee gap

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

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



Partcombine and \autoBeamOff

The function of `\autoBeamOff` when used with `\partCombine` can be difficult to understand. It may be preferable to use
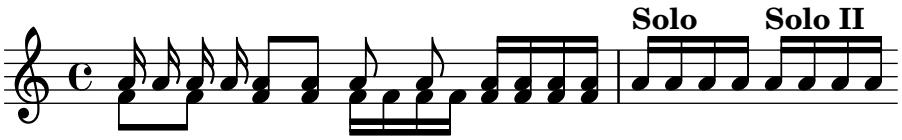
```
\set Staff.autoBeaming = ##f
```

instead to ensure that auto-beaming is turned off for the entire staff. Use this at a spot in your score where no beam generated by the auto-beamer is still active.

Internally, `\partCombine` works with four voices – up-stem single, down-stem single, combined, and solo. In order to use `\autoBeamOff` to stop all auto-beaming when used with `\partCombine`, it is necessary to use *four* calls to `\autoBeamOff`.

```
{
  % \set Staff.autoBeaming = ##f % turns off all auto-beaming

  \partCombine {
    \autoBeamOff   % applies to split up-stems
    \*4 a'16
    % \autoBeamOff % applies to combined stems
    \*4 a'8
    \*4 a'16
    % \autoBeamOff % applies to solo
    \*4 a'16
    r4
  } {
    % \autoBeamOff % applies to split down-stems
    \*4 f'8
    \*8 f'16 |
    r4
    \*4 a'16
  }
}
```



See also

Notation Reference: Section 2.4.3 [Manual beams], page 115, Section 2.4.2 [Setting automatic beam behavior], page 106.

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

The properties of a beam are determined at the *start* of its construction and any additional beam property changes that occur before the beam has been completed will not take effect until the *next*, new beam starts.

2.4.2 Setting automatic beam behavior

When automatic beaming is enabled, the placement of automatic beams is determined by three context properties: `beatBase`, `beatStructure`, and `beamExceptions`. Any of these may be overridden until the next `\time` command as described below; alternatively, their default values per time signature may changed persistently as explained in Section 2.3.2 [Time signature], page 80.

When `beamExceptions` contains an applicable rule, it takes precedence over `beatBase` and `beatStructure`.

Beaming based on `beatBase` and `beatStructure`
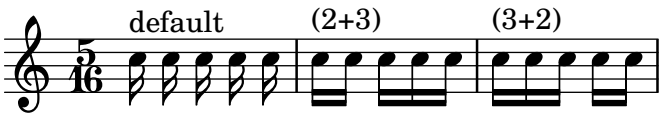
Built-in `beamExceptions` rules are defined for many common time signatures. They must be disabled if automatic beaming is to be based strictly on `beatBase` and `beatStructure`. The `beamExceptions` rules are disabled by

```
\set Timing.beamExceptions = #'()
```
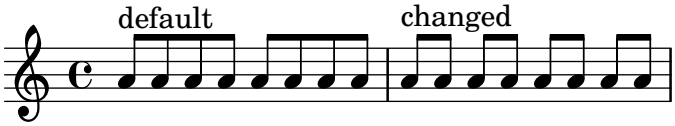
When `beamExceptions` is set to `#'()`, either due to an explicit setting or because no built-in `beamExceptions` rules are defined for the time signature in force, beams end on beats specified by the context properties `beatBase` and `beatStructure`.

The `\time` command resets `beatBase`, `beatStructure`, and `beamExceptions`, so desired modifications must be placed after the `\time` command, not before it.

```
\relative c'' {
  \time 5/16
  c16^"default" c c c c |
  % beamExceptions are unlikely to be defined for 5/16 time,
  % but let's disable them anyway to be sure
  \set Timing.beamExceptions = #'()
  \set Timing.beatStructure = 2,3
  c16^"(2+3)" c c c c |
  \set Timing.beatStructure = 3,2
  c16^"(3+2)" c c c c |
}
```

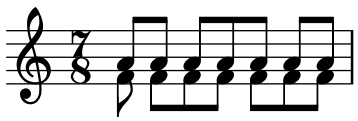



```
\relative {
  \time 4/4
  a'8^"default" a a a a a a a
  % Disable beamExceptions because they are definitely
  % defined for 4/4 time
  \set Timing.beamExceptions = #'()
  \set Timing.beatBase = #1/4
  \set Timing.beatStructure = 1,1,1,1
  a8^"changed" a a a a a a a
}
```



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
  % No need to disable beamExceptions
  % as they are not defined for 7/8 time
  \set Staff.beatStructure = 2,3,2
  <<
    \new Voice = one {
      \relative {
        a'8 a a a a a a
      }
    }
    \new Voice = two {
      \relative {
        \voiceTwo
        \set Voice.beatStructure = 1,3,3
        f'8 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:
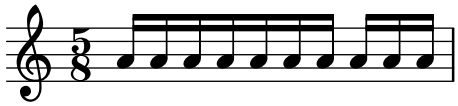
```
\time 7/8
% rhythm 3-1-1-2
% Change applied to Voice by default --  does not work correctly
% Because of auto-generated voices, all beating will
% be at beatBase #1/8
\set beatStructure = 3,1,1,2
<< \relative {a'8 a a a16 a a a a8 a} \\ \relative {f'4. f8 f f f} >>

% Works correctly with context Staff specified
\set Staff.beatStructure = 3,1,1,2
<< \relative {a'8 a a a16 a a a a8 a} \\ \relative {f'4. f8 f f f} >>
```



The value of beatBase can be adjusted to change the beaming behavior, if desired. When this is done, the value of beatStructure must be set to be compatible with the new value of beatBase.

```
\time 5/8
% No need to disable beamExceptions
% as they are not defined for 5/8 time
\set Timing.beatBase = #1/16
\set Timing.beatStructure = 7,3
\*10 a'16
```
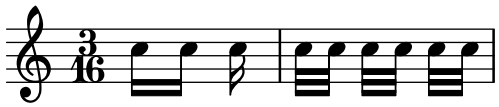


Beaming based on beamExceptions

Special auto-beaming rules (other than ending a beam on a beat) are defined in the beamExceptions property.

The value for beamExceptions, a somewhat complex Scheme data structure, is easiest generated with the \beamExceptions function. This function is given one or more manually beamed measure-long rhythmic patterns (measures have to be separated by a bar check | since the function has no other way to discern the measure length). Here is a simple example:

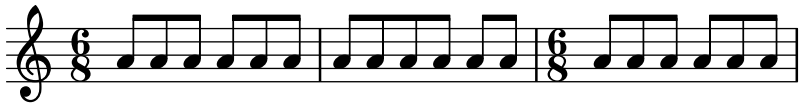
```
\relative c'' {
  \time 3/16
  \set Timing.beatStructure = 2,1
  \set Timing.beamExceptions =
    \beamExceptions { 32[ 32] 32[ 32] 32[ 32] }
  c16 c c |
  \*6 c32 |
}
```



> **Note:** A beamExceptions value must be *complete* exceptions list. That is, every exception that should be applied must be included in the setting. It is not possible to add, remove, or change only one of the exceptions. While this may seem cumbersome, it means that the current beaming settings need not be known in order to specify a new beaming pattern.

When the time signature is changed, default values of Timing.beatBase, Timing .beatStructure, and Timing.beamExceptions are set. Setting the time signature will reset the automatic beaming settings for the Timing context to the default behavior.
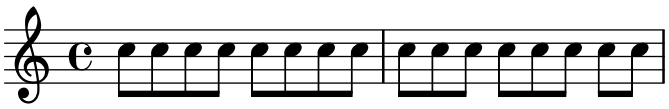
```
\relative a' {
  \time 6/8
  \*6 a8
  % group (4 + 2)
  \set Timing.beatStructure = 4,2
  \*6 a8
  % go back to default behavior
  \time 6/8
  \*6 a8
}
```



The default automatic beaming settings for a time signature are determined in scm/time-signature-settings.scm. Changing the default automatic beaming settings for a time signature is described in Section 2.3.2 [Time signature], page 80.

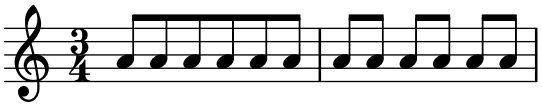
Many automatic beaming settings for a time signature contain an entry for beamExceptions. For example, 4/4 time tries to beam the measure in two if there are only eighth notes. The beamExceptions rule can override the beatStructure setting if beamExceptions is not reset.

```
\time 4/4
\set Timing.beatBase = #1/8
\set Timing.beatStructure = 3,3,2
% This won't beam (3 3 2) because of beamExceptions
\*8 c''8 |
% This will beam (3 3 2) because we clear beamExceptions
\set Timing.beamExceptions = #'()
\*8 c''8
```



In a similar fashion, eighth notes in 3/4 time are beamed as a full measure by default. To beam eighth notes in 3/4 time on the beat, reset beamExceptions.

```
\time 3/4
% by default we beam in (6) due to beamExceptions
\*6 a'8 |
% This will beam (1 1 1) due to default beatBase and beatStructure
\set Timing.beamExceptions = #'()
\*6 a'8
```



In engraving from the romantic and classical periods, beams often begin midway through the measure in 3/4 time, but modern practice is to avoid the false impression of 6/8 time (see Gould, p. 153). Similar situations arise in 3/8 time. This behavior is controlled by the context property beamHalfMeasure, which has effect only in time signatures with 3 in the numerator:

```
\relative a' {
  \time 3/4
  r4. a8 a a |
  \set Timing.beamHalfMeasure = ##f
  r4. a8 a a |
}
```



How automatic beaming works

When automatic beaming is enabled, the placement of automatic beams is determined by the context properties beatBase, beatStructure, and beamExceptions.

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 a beam ending rule is defined in beamExceptions for the beam type, use it to determine the valid places where beams may end, otherwise

- if a beam ending rule is defined in `beamExceptions` for a longer beam type, use it to determine the valid places where beams may end, otherwise
- use the values of `beatBase` and `beatStructure` to determine the ends of the beats in the measure, and end beams at the end of beats.

In the rules above, the *beam type* is the duration of the shortest note in the beamed group.

The default beaming rules can be found in `scm/time-signature-settings.scm`.

Selected snippets

Subdividing beams

The beams of consecutive 16th (or shorter) notes are, by default, not subdivided. That is, the beams of more than two stems stretch over the entire group of notes without a break. This behavior can be modified to subdivide the beams into sub-groups by setting the property `subdivideBeams` to `#t`. When set, beams are subdivided at (rhythmic) intervals to match the metric value of the subdivision.

Using the properties `beamMinimumSubdivision` and `beamMaximumSubdivision` it is possible to configure the limits of automatic beam subdivision, namely the minimum and maximum rhythmic lengths at which beamlets are removed. The default values are `0` for the former and `+inf.0` for the latter, making LilyPond subdivide beams as much as possible.

There are two special cases to consider.

- If the numerator of `beamMaximumSubdivision` is not a power of 2, the rhythmic lengths considered for subdivision are `beamMaximumSubdivision` divided by powers of 2 that stay greater than or equal to `beamMinimumSubdivision`.
- If `beamMaximumSubdivision` is smaller than `beamMinimumSubdivision`, the depth of beam subdivisions is limited by `beamMaximumSubdivision`, but not the frequency and rhythmic intervals, therefore possibly deviating from the correct, expected metric value.

If `respectIncompleteBeams` is set to `#t`, incomplete subdivisions with more than two stems are treated as an 'extension' of the previous subdivision group, i.e., the length of the previous subdivision group gets extended to also cover the incomplete subdivision. If set to `#f` (which is the default), a new subdivision group gets started instead.

```
\relative c'' {
  \time 1/4

  <>^"default"
  c32 c c c c c c c

  <>^"with subdivision"
  \set subdivideBeams = ##t
  c32 c c c c c c c

  <>^"min 1/8"
  \once \set beamMinimumSubdivision = #1/8
  c32 c c c c c c c

  <>^"max 1/16"
  \once \set beamMaximumSubdivision = #1/16
  c32 c c c c c c c

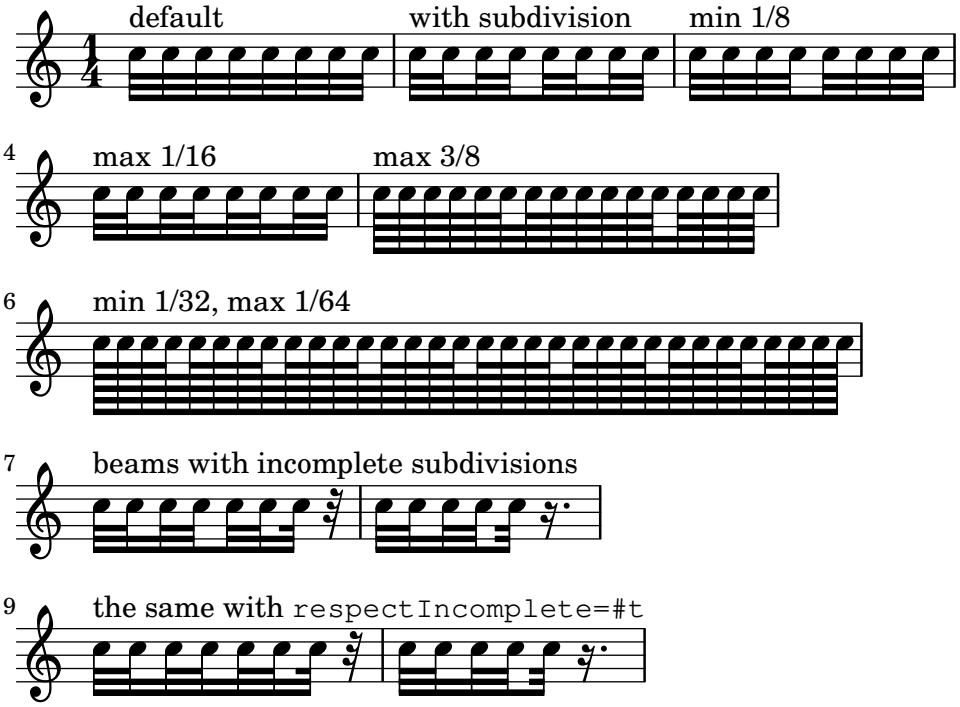
  <>^"max 3/8"
```

```
  \once \set beamMaximumSubdivision = #3/8
  \*16 c64

  <>^"min 1/32, max 1/64"
  % Set maximum beam subdivision interval to 1/64 to limit
  % subdivision depth, despite not being metrically correct.
  \once \set beamMinimumSubdivision = #1/32
  \once \set beamMaximumSubdivision = #1/64
  \*32 c128
  \break

  <>^"beams with incomplete subdivisions"
  c32 c c c c c c r32
  c32 c c c c r16.
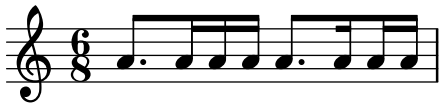
  <>^\markup { "the same with"
               \typewriter { "respectIncomplete=#t" } }
  \set respectIncompleteBeams = ##t
  % The incomplete subgroup extends the completed subgroup.
  c32 c c c c c c r32
  % No visual change since we have only two stems in the
  % incomplete subgroup.
  c32 c c c c r16.
}
```



Strict beat beaming

Beamlets can be set to point in the direction of the beat to which they belong. The first beam avoids sticking out flags (the default); the second beam strictly follows the beat.

```
\relative c'' {
  \time 6/8
  a8. a16 a a
  \set strictBeatBeaming = ##t
  a8. a16 a a
}
```



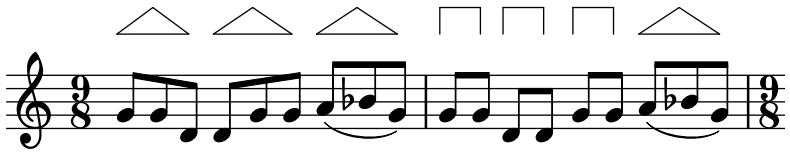
Conducting signs, measure grouping signs

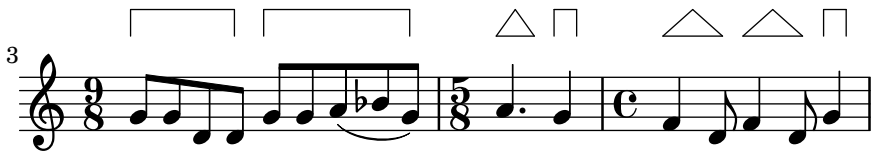
Context properties control the grouping of beats within a measure: `beatStructure` lists the length of each beat in units of `beatBase`. Default values are established in `scm/time-signature-settings.scm`. These properties may be changed particularly with `\set`.

Alternatively, `\time` optionally accepts a beat structure to use instead of the default. `\time` applies to the `Timing` context, so it does not reset values of properties that are set in lower-level contexts such as `Voice`.

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/time-signature-settings.scm`. For the 4/4 measure you have to explicitly set `beatBase` to eighths so that the bar's irregular pattern gets displayed.

```
\score {
  \new Voice \relative c'' {
    \time 9/8
    g8 g d d g g a( bes g) |
    \set Timing.beatStructure = 2,2,2,3
    g8 g d d g g a( bes g) |
    \time 4,5 9/8
    g8 g d d g g a( bes g) |
    \time 5/8
    a4. g4 |
    \time 3,3,2 4/4
    \set Timing.beatBase = #1/8
    f4 d8 f4 d8 g4
  }
  \layout {
    \context {
      \Staff
      \consists "Measure_grouping_engraver"
    }
  }
}
```
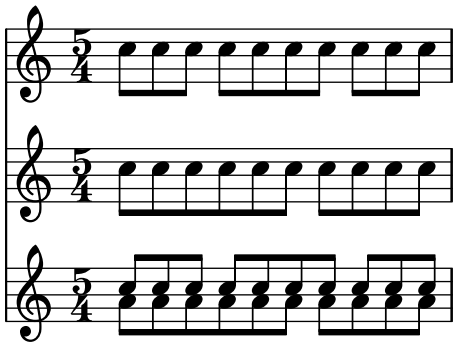




Beam endings in Score context

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
  \set Score.beatBase = #1/8
  \set Score.beatStructure = 3,4,3
  <<
    \new Staff {
      c8 c c c c c c c c c
    }
    \new Staff {
      % Modify beaming for just this staff
      \set Staff.beatStructure = 6,4
      c8 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
          \set Voice.beatStructure = 6,4
          a8 a a a a a a a a a
        }
      >>
    }
  >>
}
```



See also

Notation Reference: Section 2.3.2 [Time signature], page 80.

Installed Files: `scm/time-signature-settings.scm`.

Snippets: Section "Rhythms" in *Snippets*.

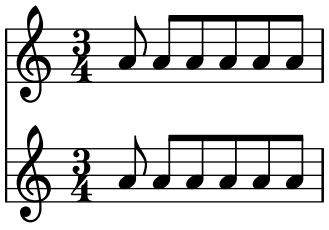
Internals Reference: Section "Auto_beam_engraver" in *Internals Reference*, Section "Beam" in *Internals Reference*, Section "BeamForbidEvent" in *Internals Reference*, Section "beam-interface" in *Internals Reference*.

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.

By default, the `Timing` translator is aliased to the `Score` context. This means that setting the time signature in one staff will affect the beaming of the other staves as well. Thus, a time signature setting in a later staff will reset custom beaming that was set in an earlier staff. One way to avoid this problem is to set the time signature in only one staff.

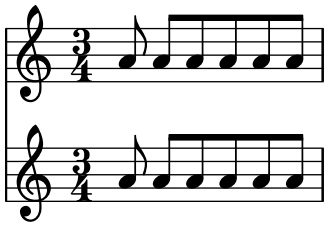
```
<<
  \new Staff {
    \time 3/4
    \set Timing.beatBase = #1/8
    \set Timing.beatStructure = 1,5
    \set Timing.beamExceptions = #'()
    \*6 a'8
  }
  \new Staff {
    \*6 a'8
  }
>>
```



The default beam settings for the time signature can also be changed, so that the desired beaming will always be used. Changes in automatic beaming settings for a time signature are described in Section 2.3.2 [Time signature], page 80.

```
<<
  \new Staff {
    \overrideTimeSignatureSettings
      3/4               % timeSignature
      #1/8              % beatBase
      1,5               % beatStructure
      #'()              % beamExceptions
    \time 3/4
    \*6 a'8
```
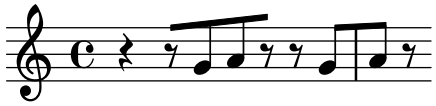
```
  }
  \new Staff {
    \time 3/4
    \*6 a'8
  }
>>
```



2.4.3 Manual beams
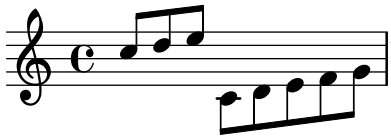
In some cases it may be necessary to override the automatic beaming algorithm. For example, the auto-beamer will not put beams over rests or bar lines, and in choral scores the beaming is often set to follow the meter of the lyrics rather than the notes. For complete manual control, beams can be specified by marking the begin and end point with `[` and `]`, respectively:

```
\relative { r4 r8[ g' a r] r g[ | a] r }
```
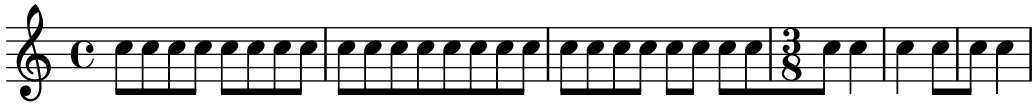


Beaming direction can then be set manually using direction indicators:

```
\relative { c''8^[ d e] c,_[ d e f g] }
```
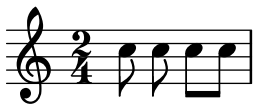


Instead of taking full manual control over the beams, it often suffices to give the automatic beaming algorithm some hints. With `\beamBreak`, you demand that an automatically created beam be interrupted. With `\noBeamBreak`, on the contrary, you force two adjoining notes to get connected by a beam. This may even cause a completely new beam to be created:

```
{
  c''8 8 8 8 8 8 8 8 |
  c''8 8 8 8 \noBeamBreak 8 8 8 8 |
  8 8 8 8 8 8 \beamBreak 8 8 \noBeamBreak |
  \time 3/8 |
  8 4 |
  4 8 \noBeamBreak |
  8 4 |
}
```
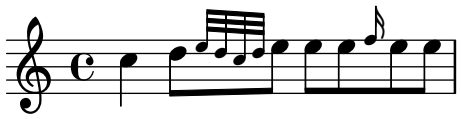


Also, individual notes may be marked with `\noBeam` to completely prevent them from being beamed:

```
\relative {
  \time 2/4
  c''8 c\noBeam c c
}
```
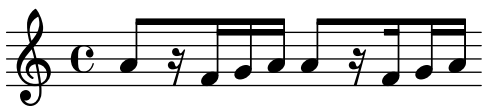


Grace note beams and normal note beams can occur simultaneously. Unbeamed grace notes are not put into normal note beams.

```
\relative {
  c''4 d8[
  \grace { e32 d c d }
  e8] e[ 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.

```
\relative a' {
  a8[ r16 f g a]
  a8[ r16
  \set stemLeftBeamCount = 2
  \set stemRightBeamCount = 1
  f16
  \set stemLeftBeamCount = 1
  g16 a]
}
```



Predefined commands

`\noBeam`.

Selected snippets

Beam nibs

Beam nibs at the start and end of beams together with beams attached to solitary notes that look like flat flags are possible with a combination of `stemLeftBeamCount`, `stemRightBeamCount`, and paired `[]` beam indicators.

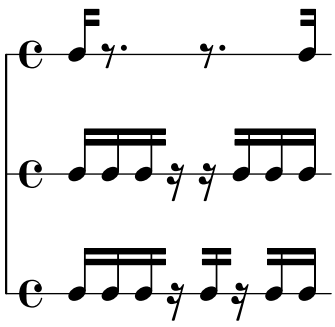
For imitating right-pointing flat flags on lone notes, use paired `[]` beam indicators and set `stemLeftBeamCount` to zero. For imitating left-pointing flat flags on lone notes, set `stemRightBeamCount` to zero instead (line one).

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

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

Note that `\set stemLeftBeamCount` is always equivalent to `\once \set`. In other words, the beam count settings are not "sticky", so the pair of nibs attached to the lone 16th note in the last example has nothing to do with the `\set` command for the beam before.

```
\score {
  <<
    \new RhythmicStaff {
      \set stemLeftBeamCount = 0
      c16[] r8.
      r8.
      \set stemRightBeamCount = 0
      16[]
    }
    \new RhythmicStaff {
      16 16
      \set stemRightBeamCount = 2
      16 r r
      \set stemLeftBeamCount = 2
      16 16 16
    }
    \new RhythmicStaff {
      16 16
      \set stemRightBeamCount = 2
      16 r16
      16[] r16
      \set stemLeftBeamCount = 2
      16 16
    }
  >>
}
```



Using alternative flag styles

Alternative shapes for flags on eighth and shorter notes can be displayed by overriding the `stencil` property of `Flag`. LilyPond provides the following functions: `modern-straight-flag`, `old-straight-flag`, and `flat-flag`. Use `\revert` to restore the default shape.

To get stacked (i.e., vertically more compact) flags, call the command `\flagStyleStacked`, which can be reset with `\flagStyleDefault`.

Overriding the `Flag` stencil does not change how flag elements are positioned vertically. This is especially noticeable for flat flags: LilyPond doesn't dynamically adjust the vertical gaps between flag elements in the same way as it does for beams. A possible solution to harmonize the appearance is to replace flat flags with half beams, as shown in the second staff; however, this can't be done automatically. In the code of this snippet, such half beams are entered with `@` as a prefix, for example `@c8`.

Be aware that half beams are *not* `Flag` grobs. This means in particular that modifying `Flag` properties won't have any effect on them (you have to use `Beam` properties instead), and properties for their associated `Stem` grob will also behave beam-like.

```
"@" =
#(define-music-function (music) (ly:music?)
  #{ \set stemLeftBeamCount = 0 $music [] #})

testnotes = {
  \autoBeamOff
  c8 d16 e''32 f64 \acciaccatura { g,,,8 } a128 b
}

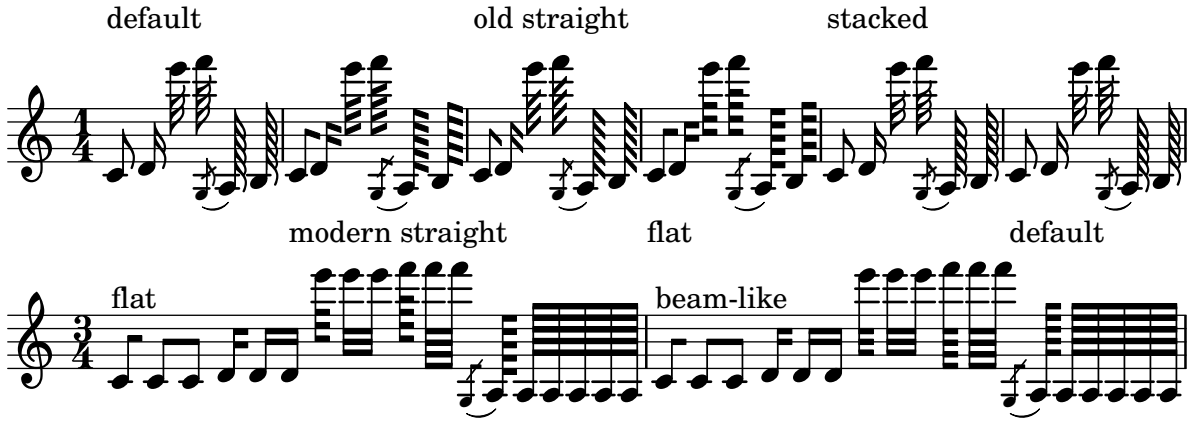
\relative c' {
  \override TextScript.staff-padding = 6
  \time 1/4
    <>^"default" \testnotes
  \override Flag.stencil = #modern-straight-flag
    <>_"modern straight" \testnotes
  \override Flag.stencil = #old-straight-flag
    <>^"old straight" \testnotes
  \override Flag.stencil = #flat-flag
    <>_"flat" \testnotes
  \revert Flag.stencil

  \flagStyleStacked
    <>^"stacked" \testnotes
  \flagStyleDefault
    <>_"default" \testnotes
}
```

```
\relative c' {
  \time 3/4
  \override Flag.stencil = #flat-flag

  <>^"flat" c8 c[ c] d16 d[ d] e''32 e[ e] f64 f[ f]
    \acciaccatura { g,,,8 } a128 a[ a a a]
  <>^"beam-like" @c8 c[ c] @d16 d[ d] @e''32 e[ e] @f64 f[ f]
    \acciaccatura { g,,,8 } @a128 a[ a a a]
}

\layout {
  indent = 0
  \context {
    \Score
    \override NonMusicalPaperColumn.line-break-permission = ##f
  }
}
```



See also

Notation Reference: Section 36.1 [Direction and placement], page 774, Section 2.6.1 [Grace notes], page 148.

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*.

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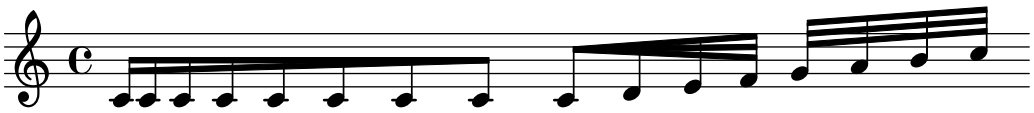
Feathered beams are used to indicate that a small group of notes should be played at an increasing (or decreasing) tempo, without changing the overall tempo of the piece. The extent of the feathered beam must be indicated manually using `[` and `]`, and the beam feathering is turned on by specifying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

If the placement of the notes and the sound in the MIDI output is to reflect the *ritardando* or *accelerando* indicated by the feathered beam the notes must be grouped as a music expression delimited by braces and preceded by a `\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.

```
\relative c' {
  \override Beam.grow-direction = #LEFT
  \featherDurations 2/1
  { c16[ c c c c c c c] }
  \override Beam.grow-direction = #RIGHT
  \featherDurations 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.
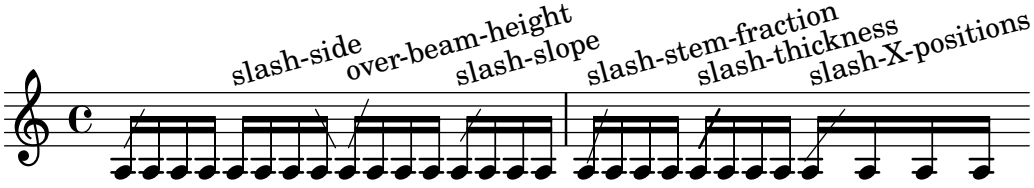
2.4.5 Slashed beams

Slashed beams are printed by using the special stencil procedure `beam::slashed-stencil`. The slash may be printed at the left or right side of the beam and is further customizable by overrides of the `details` subproperties `over-beam-height`, `slash-slope`, `slash-side`, `slash-stem-fraction`, `slash-thickness`, and `slash-X-positions`. Note that those subproperties negotiate with each other to get a pleasing output, i.e., changing one of them may have impact on others.

```
mus = \*4 a16

{
  \override TextScript.rotation = #'(15 1 0)
  \override Beam.stencil = #beam::slashed-stencil
  \mus
  <>^"slash-side" %% default: LEFT
  \once \override Beam.details.slash-side = #RIGHT
  \mus
  <>^"over-beam-height" %% default: 0.75
  \once \override Beam.details.over-beam-height = #1.5
```

```
  \mus
  <>^"slash-slope" %% default: 2
  \once \override Beam.details.slash-slope = #1.0
  \mus
  <>^"slash-stem-fraction" %% default: 0.3
  \once \override Beam.details.slash-stem-fraction = #0.6
  \mus
  <>^"slash-thickness" %% default: 0.1
  \once \override Beam.details.slash-thickness = #0.2
  \mus
  <>^"slash-X-positions" %% default: (-0.5 . 1)
  \once \override Beam.details.slash-X-positions = #'(-1 . 2)
  \mus
}
```
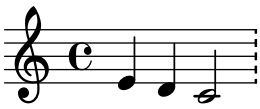


2.5 Bars

2.5.1 Bar lines

Bar lines are used to delimit measures and sections, and to indicate repetition. Normally, simple bar lines are automatically inserted into the printed output at places according to the current time signature. Various commands insert other kinds of bar lines automatically as part of their effect (see Section 2.5.2 [Automatic bar lines], page 131).

A bar line inserted automatically can be changed to another type with the `\bar` command:

```
\relative { e'4 d c2 \bar "!" }
```



The final note of a measure is not required to end on the automatically inserted 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 inhibit line breaks, leading to a line of highly compressed music or music that flows off the page.

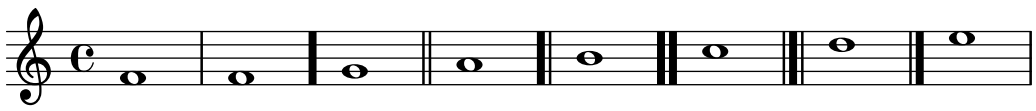
Line breaks are also permitted at manually inserted bar lines even within incomplete measures. To allow a line break without printing a bar line, use `\allowBreak`; see Section 28.1 [Line breaking], page 689.

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.

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 and accidentals. 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.

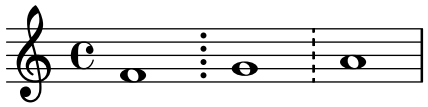
Various single and double bar lines are available for manual insertion:

```
\relative {
  f'1 \bar "|"
  f1 \bar "."
  g1 \bar "||"      % see \section
  a1 \bar ".|"
  b1 \bar ".."
  c1 \bar "|.|"
  d1 \bar "|."      % see \fine
  e1
}
```



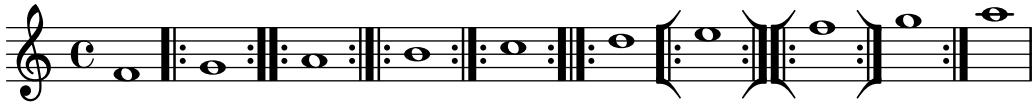
together with dotted and dashed bar lines:

```
\relative {
  f'1 \bar ";"
  g1 \bar "!"
  a1
}
```



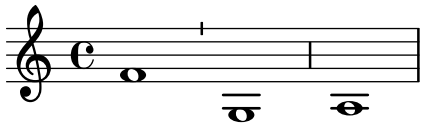
and various repeat bar lines:

```
\relative {
  f'1 \bar ".|:"
  g1 \bar ":..:"
  a1 \bar ":|.|:"
  b1 \bar ":|.:"
  c1 \bar ":.|.:"
  d1 \bar "[|:"
  e1 \bar ":|][|:"
  f1 \bar ":|]"
  g1 \bar ":|."
  a1
}
```



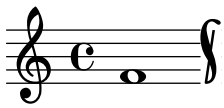
Ticks and short bar lines are also available; however, in the context of Gregorian chant, using `\divisioMinima` and `\divisioMaior` is preferable (see Section 17.4.4 [Divisiones], page 556).

```
f'1 \bar "'"
g1 \bar ","
a1
```



LilyPond supports Kievan notation and provides a special Kievan bar line:

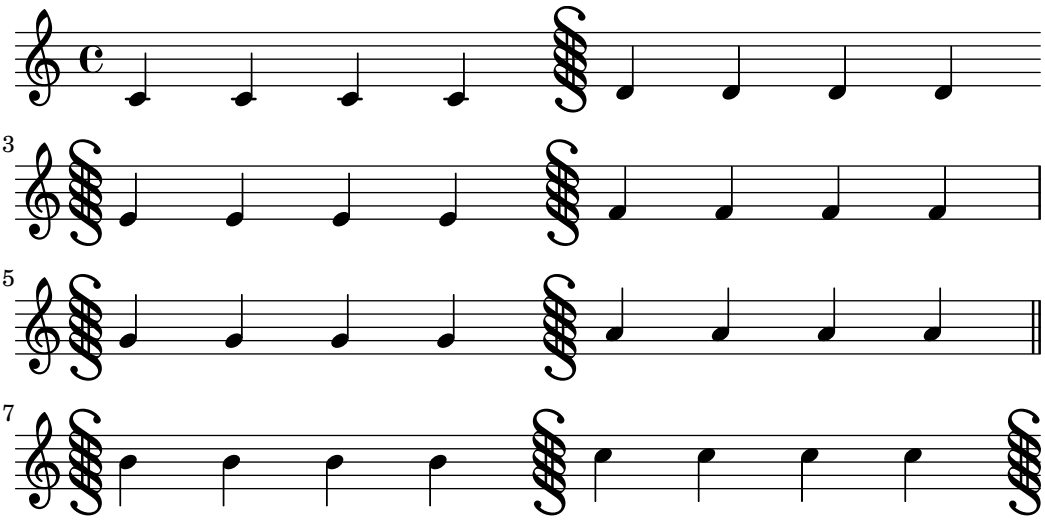
```
f'1 \bar "k"
```

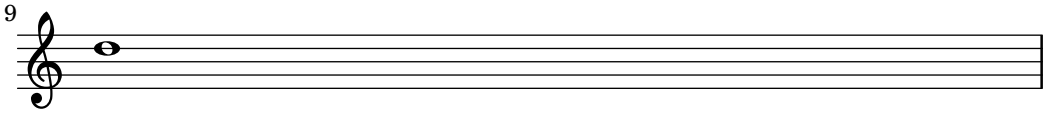


Further details of this notation are explained in Section 17.5 [Typesetting Kievan square notation], page 564.

There are various in-staff segno signs which differ in their behavior at line breaks:

```
\fixed c' {
  c4 4 4 4
  \bar "S"
  d4 4 4 4 \break
  \bar "S"
  e4 4 4 4
  \bar "S-|"
  f4 4 4 4 \break
  \bar "S-|"
  g4 4 4 4
  \bar "S-||"
  a4 4 4 4 \break
  \bar "S-||"
  b4 4 4 4
  \bar "S-S"
  c'4 4 4 4 \break
  \bar "S-S"
  d'1
}
```
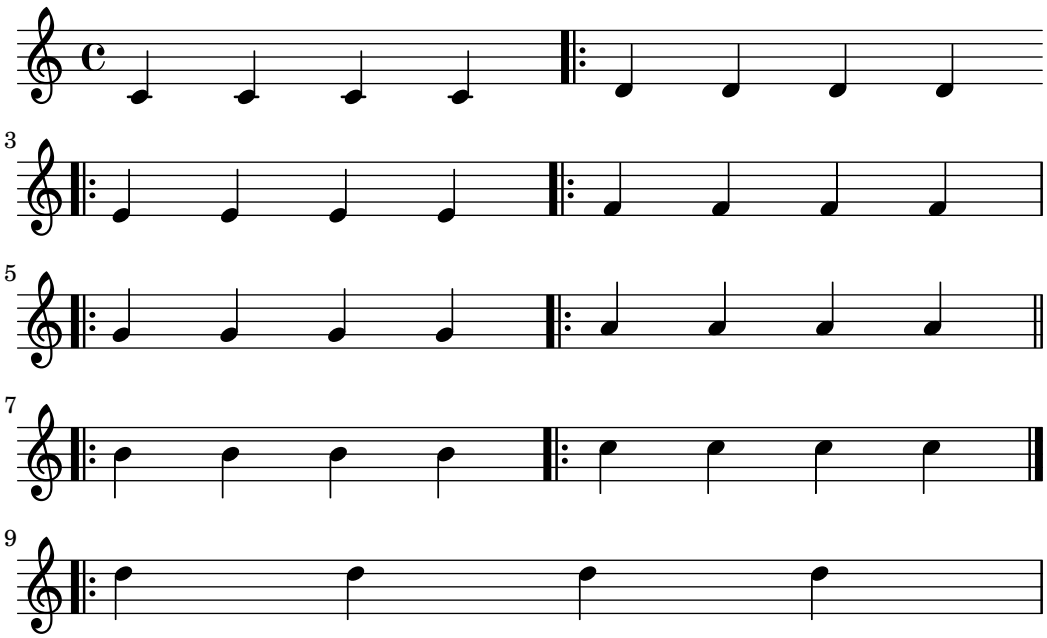




Although the bar line types signifying repeats may be inserted manually they do not in themselves cause LilyPond to recognize a repeated section. Such repeated sections are better entered using the various repeat commands (see Chapter 4 [Repeats], page 189), which automatically print the appropriate bar lines, which can be customized (see Section 2.5.2 [Automatic bar lines], page 131).

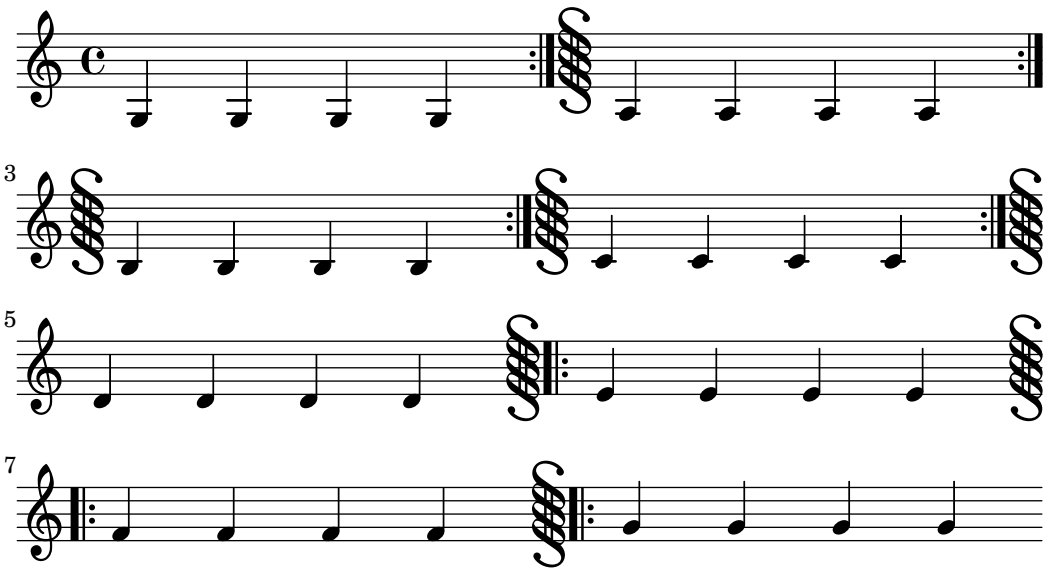
In addition, you can specify `".|:-||"`, which is equivalent to `".|:"` except at line breaks, where it gives a double bar line at the end of the line and a start repeat at the beginning of the next line.
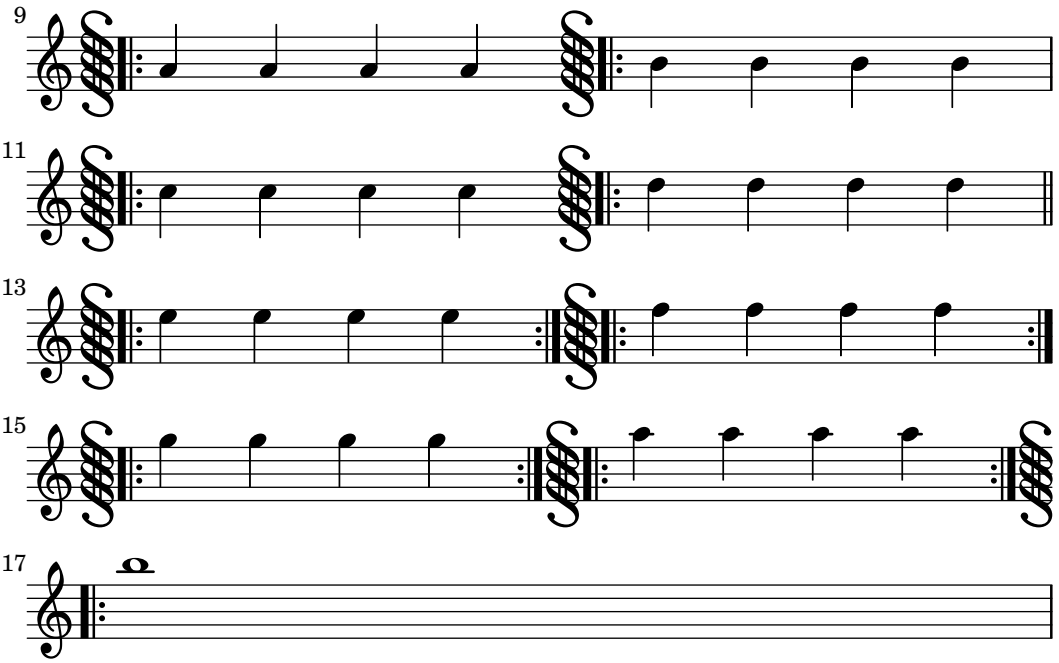
```
\fixed c' {
  c4 4 4 4
  \bar ".|:"
  d4 4 4 4 \break
  \bar ".|:"
  e4 4 4 4
  \bar ".|:-|"
  f4 4 4 4 \break
  \bar ".|:-|"
  g4 4 4 4
  \bar ".|:-||"
  a4 4 4 4 \break
  \bar ".|:-||"
  b4 4 4 4
  \bar ".|:-|."
  c'4 4 4 4 \break
  \bar ".|:-|."
  d'4 4 4 4
}
```



There are various combinations of repeats with the segno sign:

```
\fixed c' {
  g,4 4 4 4
  \bar ":|.S"
  a,4 4 4 4 \break
  \bar ":|.S"
  b,4 4 4 4
  \bar ":|.S-S"
  c4 4 4 4 \break
  \bar ":|.S-S"
  d4 4 4 4
  \bar "S.|:-S"
  e4 4 4 4 \break
  \bar "S.|:-S"
  f4 4 4 4
  \bar "S.|:"
  g4 4 4 4 \break
  \bar "S.|:"
  a4 4 4 4
  \bar "S.|:-|"
  b4 4 4 4 \break
  \bar "S.|:-|"
  c'4 4 4 4
  \bar "S.|:-||"
  d'4 4 4 4 \break
  \bar "S.|:-||"
  e'4 4 4 4
  \bar ":|.S.|:"
  f'4 4 4 4 \break
  \bar ":|.S.|:"
  g'4 4 4 4
  \bar ":|.S.|:-S"
  a'4 4 4 4 \break
  \bar ":|.S.|:-S"
  b'1
}
```





Many of the repeat and segno bar lines above can be inserted automatically by `\repeat` commands (see Chapter 4 [Repeats], page 189).

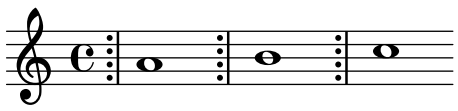
New bar line types can be defined with `\defineBarLine`:

`\defineBarLine` *bar-type* `#'(`*eol-bar bol-bar span-bar*`)`

Briefly, the *bar-type* argument specifies the bar line glyph to use in the middle of a staff line, and also serves as the name by which this bar line type is referenced. The other arguments specify the bar line glyph to use at the end or beginning of a line, or in the span between multiple staves. Setting any of *eol-bar*, *bol-bar*, or *span-bar* to `#t` means to use the same bar line type specified by *bar-type* for the corresponding position. Setting them to `#f` means to print no bar line in the corresponding position.

In more detail, the *bar-type* argument is a string that serves a dual purpose: It specifies the bar line glyph to be printed when it occurs in the middle of a staff line; and it identifies the bar line object that can be invoked with `\bar` *bar-type*. It must have the form *midglyph* or *midglyph-annotation* (with a literal hyphen), where *annotation* is an arbitrary string, and *midglyph* is a string each of whose characters is the name of one of the predefined bar line elements listed below. The resulting bar line glyph to be used in the middle of a line is the concatenation of these elements. For example, a *bar-type* of either `";|"` or `";|-other"` specifies a compound bar line consisting of a dotted line (‘`;`’) paired with a solid line (‘`|`’):

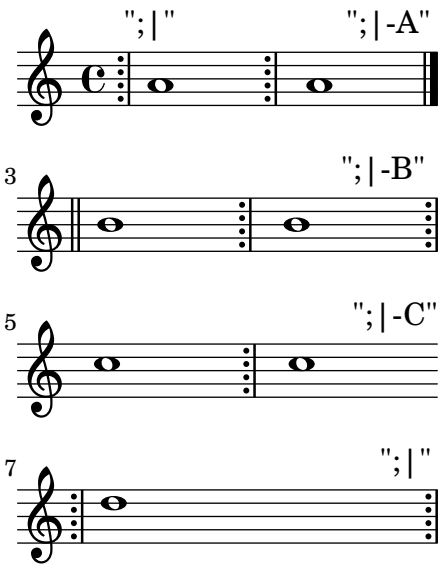
```
\defineBarLine ";|" #'(#t #t #t)
\defineBarLine ";|-other" #'(#f #f #f)
\fixed c' {
   \bar ";|" a1 \bar ";|" b1 \bar ";|-other" c'1 \bar ";|-other"
}
```



The *annotation* (‘`other`’ in the second example above) is used to distinguish this bar type from others with the same *midglyph* but different line break or multi-staff behavior. (By convention, the string specified in *eol-bar* is often used as the annotation, so we might have named the second example ‘`";|-f"`’.)

The arguments *eol-bar* and *bol-bar* specify the bar line to be printed at the end of the line and beginning of the next line, when `\bar` *bar-type* occurs at a line break. *bol-bar* also applies when `\bar` *bar-type* is used at the beginning of a score. The format of these arguments is the same as that of *bar-type*. The string of bar line elements specifies the bar line glyph to print at the corresponding line position. In addition, either of these arguments can be `#t` as a shorthand for copying the value of *bar-type*; or `#f` to print no bar line. For example, all of the bar line types in this example print a dotted-solid line pair in the middle of a staff line, but have different behavior at ends or beginnings of lines:

```
%   dotted-solid everywhere
\defineBarLine  ";|"  #'( #t #t #t)
%   solid-bold at EOL, solid-solid at BOL
\defineBarLine  ";|-A" #'( "|."  "||"  #f )
%   dotted-solid at EOL, nothing at BOL
\defineBarLine  ";|-B" #'( #t  #f  #f )
%   nothing at EOL, dotted-solid at BOL
\defineBarLine  ";|-C" #'( #f  #t  #f )
\relative c'' {
  \bar ";|"  \textMark "\";|\""
  a1  \bar ";|-A"
  a1  \bar ";|-A"  \textEndMark "\";|-A\""  \break
  b1  \bar ";|-B" b \bar ";|-B"  \textEndMark "\";|-B\""  \break
  c1  \bar ";|-C" c \bar ";|-C"  \textEndMark "\";|-C\""   \break
  d1  \bar ";|" \textEndMark "\";|\""
}
```

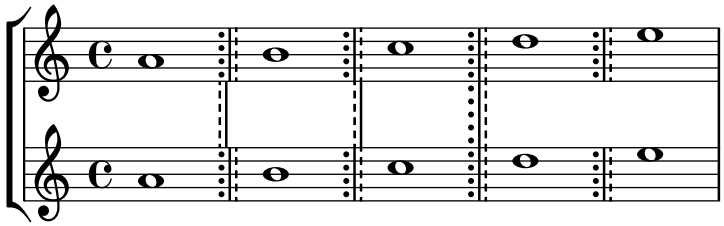


Note: The *eol-bar* or *bol-bar* strings may be names of previously defined bar line types. In a single staff context, it does not matter: the bar line elements in the given string are used regardless of any features of the defined bar line named by the string. In particular, any annotation is ignored in this context. However, in a multi-staff system it is important that *eol-bar* and *bol-bar* refer to previously defined bar line types (including *bar-type* itself), or be `#t` or `#f`. This is explained in more detail below.

The argument *span-bar* has an effect only in multi-staff systems (see Section 6.1.2 [Grouping staves], page 244), where it specifies what to print between grouped staves. This argument should be a string of bar line elements, of the same length as *bar-type* or shorter. (Extra elements are ignored.) Each element will be printed in line with the corresponding element of

bar-type. A space character (‘ ’) can be used to omit a bar line element but leave space for it. Setting *span-bar* to `#t` makes it the same as the mid-line glyph. Setting it to `#f` omits the span bar, and setting it to `""` (an empty string) makes a zero-width span bar. Here are some examples:

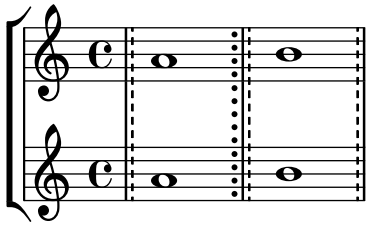
```
\defineBarLine  ";|!-A"  #'(#t #t "!|")
\defineBarLine  ";|!-B"  #'(#t #t " !|")
\defineBarLine  ";|!-C"  #'(#t #t #t)
\defineBarLine  ";|!-D"  #'(#t #t #f)
\fixed c' {
  \new StaffGroup <<
    \new Staff {
      a1 \bar ";|!-A"
      b  \bar ";|!-B"
      c' \bar ";|!-C"
      d' \bar ";|!-D"
      e'
    }
    \new Staff {
      a1 b c' d' e'
    }
  >>
}
```



At line breaks, instead of using *span-bar*, the bar line types referenced by *eol-bar* and *bol-bar* are used to determine the span bar to print. In the context of a staff group, if *eol-bar* and *bol-bar* do not refer to defined bar line types, LilyPond issues a warning, and no span bar is printed at line breaks. It is allowed for *eol-bar* or *bol-bar* to be `#f`, in which case no span bar is printed at the corresponding line position. These arguments can also be `#t`, or equivalently, equal to *bar-type*, in which case the current *span-bar* does determine the span bar to print at the corresponding line position:

```
\defineBarLine  "!|-t"  #'(#t #t #t)
\defineBarLine  "|!-t"  #'(#t #t #t)
\defineBarLine  ";|!-bad"  #'("|!" "!|" #t)  % fails at line breaks
\defineBarLine  ";|!-good" #'("!|-t" "|!-t" #t)
\relative c'' {
  \new StaffGroup <<
    \new Staff {
      \bar ";|!-good"
      a1 \bar ";|!-good"
      % \bar ";|!-bad"     % "WARNING: No span bar glyph defined..."
      b1 \bar ";|!-good"
    }
    \new Staff {
      a1 b1
    }
```

```
  >>
}
```

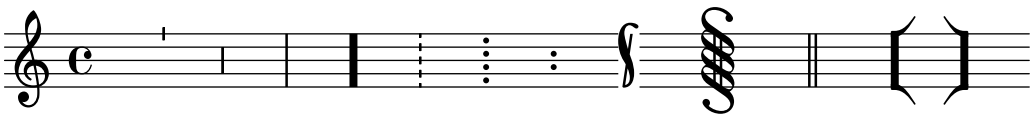


Note: If *span-bar* is a string, it should contain *only* bar line elements, or space (‘ ’), and not an annotation such as allowed in the other arguments. If *span-bar* contains an annotation, LilyPond issues a warning. The one exception is that it may be equal to *bar-type*, in which case no warning is issued even if *bar-type* includes an annotation.

All the available bar line elements are shown below. Most also have predefined bar types (for arguments to `\bar`) that reference them individually. Some elements are primarily intended to be combined with others and so do not have predefined individual bar types.

```
\defineBarLine ":" #'(#f #t #f)
\defineBarLine "=" #'(#t #f #t)
\defineBarLine "[" #'(#f #t #f)
\defineBarLine "]" #'(#t #f #f)

\new Staff {
  s1 \bar "'"
  s1 \bar ","
  s1 \bar "|"
  s1 \bar "."
  s1 \bar "!"
  s1 \bar ";"
  s1 \bar ":"
  s1 \bar "k"
  s1 \bar "S"
  s1 \bar "="
  s1 \bar "["
  s1 \bar "]"
  s1 \bar ""
}
```
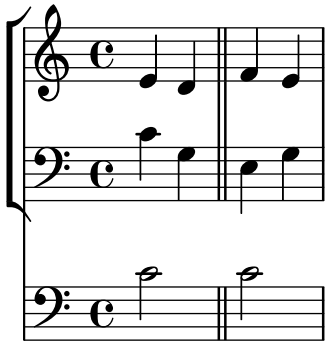


The `"="` bar line provides a double span bar line for use in combination with the segno sign. Using it as a stand-alone double thin bar line is not recommended; `\bar "||"` is preferred.

If additional elements are needed, LilyPond provides a simple way to define them. For more information on modifying or adding bar lines, see file `scm/bar-line.scm`.

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 \relative {
      e'4 d
      \bar "||"
      f4 e
    }
    \new Staff \relative { \clef bass c'4 g e g }
  >>
  \new Staff \relative { \clef bass c'2 c2 }
>>
```



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

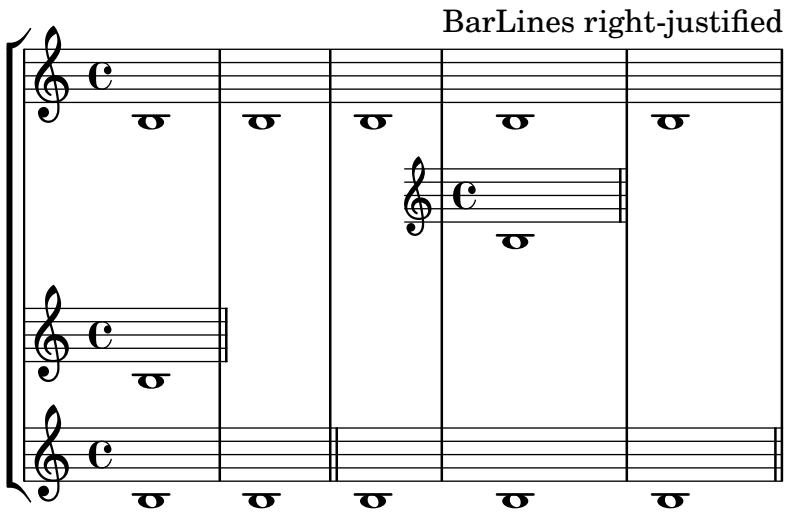
It is also possible to set different types of bar lines in a score with multiple staves, using one of the possible commands or properties explained below (see Section 2.5.2 [Automatic bar lines], page 131). Of course this may lead to a mismatch of bar lines and span bars, due to their different width.

Usually bar lines are left-aligned (disregarding colon signs as in repeat bar lines). To get them right-aligned the command

```
[\once] \override Context.BarLine.right-justified = ##t
```

needs to be applied, where *Context* is a context suitable for multiple staves, like *Score*, *StaffGroup*, *Grandstaff*, etc.

```
\new StaffGroup
  <<
    \new Staff = "a" {
      b1  b b
      <<
        { \textMark "BarLines right-justified" b b }
        \new Staff \with { alignAboveContext = "b" }
        {
          \override StaffGroup.BarLine.right-justified = ##t
          b
          \section
        }
      >>
    }
    \new Staff = "b" { b \section }
    \new Staff = "c" { b b \section b b b \section }
  >>
```



After a line-break bar lines are never right-aligned. For mid-line and right-aligned bar lines the anchor-point (used to align `BarNumber`, `RehearsalMark`, etc.) moves acordingly.

See also

Notation Reference: Section 28.1 [Line breaking], page 689, Chapter 4 [Repeats], page 189, Section 6.1.2 [Grouping staves], page 244.

Installed Files: `scm/bar-line.scm`.

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).

2.5.2 Automatic bar lines

Various commands other than `\bar` can also create bar lines as part of their effect. The bar lines created in these cases can be changed by setting context properties. If a property is set to `'()` or is unset, it is ignored; otherwise, the value must be a predefined bar type or one previously defined with the `\defineBarLine` command (see Section 2.5.1 [Bar lines], page 121).

Multiple reasons for creating different automatic bar lines may apply at the same time. Conflicts are resolved in part by providing properties for predetermined combinations and in part by a priority scheme. The table below presents the available properties in order of increasing priority.

`underlyingRepeatBarType`
: Used at points of repetition or departure where no bar line would otherwise appear. This is expected when repeated sections are not aligned to measures. Several commands employ this bar type: `\codaMark`, `\inStaffSegno`, `\repeat segno`, `\repeat volta`, and `\segnoMark`.

`caesuraType underlying-bar-line`
: Used at `\caesura`; see Section 9.7.4 [Phrase bar lines in hymn tunes], page 409.

`submeasureBarType`
: Used at submeasure boundaries specified by `submeasureStructure`, when `\submeasureBarsOn` allows; see Section 2.3.2 [Time signature], page 80.

`measureBarType`
: Used at a measure boundary.

`caesuraType bar-line`
: Used at `\caesura`; see Section 9.7.4 [Phrase bar lines in hymn tunes], page 409.

`sectionBarType`
: Used at a section break created by `\section`.

`fineBarType`
: Used at `\fine`.

`doubleRepeatBarType`
`doubleRepeatSegnoBarType`
`endRepeatBarType`
`endRepeatSegnoBarType`
`fineSegnoBarType`
`fineStartRepeatSegnoBarType`
`segnoBarType`
`startRepeatBarType`
`startRepeatSegnoBarType`
: Only one of these bar types is used at a time; which one is used depends on the structure of the piece.

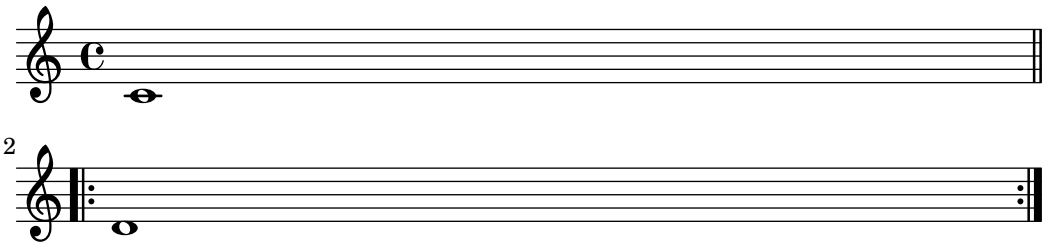
  Properties with `startRepeat` or `endRepeat` in the name are used at the start or end of a repeated section created by `\repeat volta`, and properties with `doubleRepeat` in the name are used where the end of one repeated section and the start of another coincide.

  Properties with `segno` in the name are used at an in-staff segno, which can be created by `\repeat segno` or `\segnoMark` when the `segnoStyle` property is set to `bar-line`, or created by `\inStaffSegno`.

  Properties with `fine` in the name are used at `\fine`.

Priority applies independently to beginning-, middle-, and end-of-line bar lines, allowing a lower-priority bar line to appear where higher-priority bar types have no glyphs defined (see Section 2.5.1 [Bar lines], page 121).

```
\fixed c' {
  c1 \section \break
  \repeat volta 2 d1
}
```

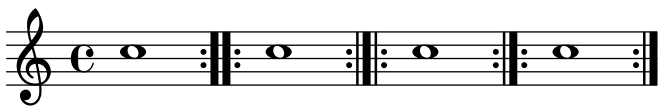


Selected snippets

Setting the double repeat default for volte

There are different double repeat styles for volte that can be selected using the context property `doubleRepeatBarType`.

```
\relative c'' {
  \repeat volta 2 { c1 }
  \set Score.doubleRepeatBarType = ":..:"
  \repeat volta 2 { c1 }
  \set Score.doubleRepeatBarType = ":|.|:"
  \repeat volta 2 { c1 }
  \set Score.doubleRepeatBarType = ":|.:"
  \repeat volta 2 { c1 }
}
```



See also

Notation Reference: Section 2.5.1 [Bar lines], page 121, Chapter 4 [Repeats], page 189.

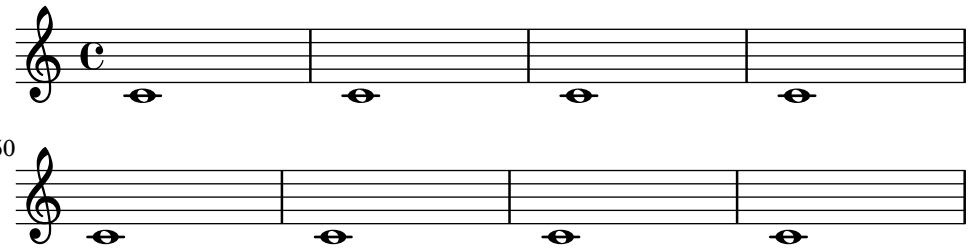
Snippets: Section "Rhythms" in *Snippets*.

Internals Reference: Section "Repeat_acknowledge_engraver" in *Internals Reference*.

2.5.3 Bar numbers

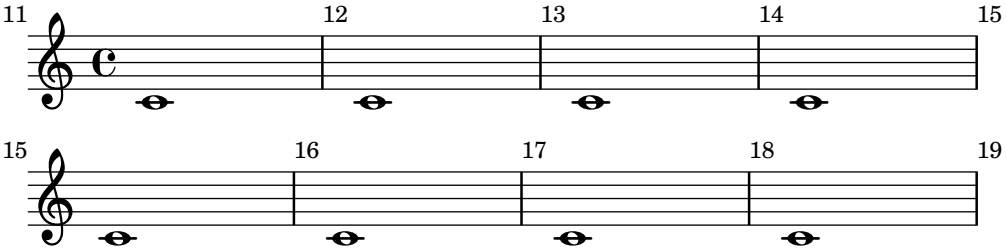
Bar numbers are typeset by default at the start of every line except the first line. The number itself is stored in the `currentBarNumber` property, which is normally updated automatically for every measure. It may also be set manually:

```
\relative c' {
  c1 c c c
  \break
  \set Score.currentBarNumber = 50
  c1 c c c
}
```



The default behavior of only printing bar numbers at the start of every line can be changed through 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:

```
\relative c' {
  \override Score.BarNumber.break-visibility = ##(#t #t #t)
  \set Score.currentBarNumber = 11
  c1 | c | c | c |
  \break
  c1 | c | c | c |
}
```



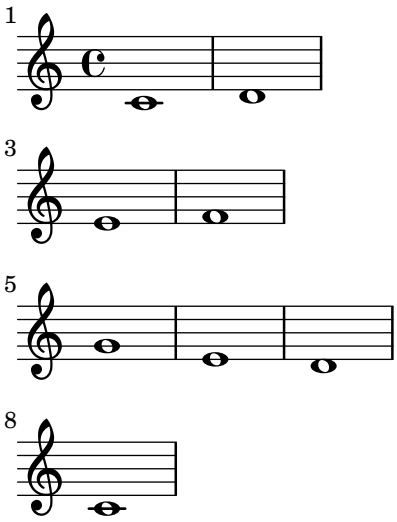
Selected snippets

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. This can be changed by setting the barNumberVisibility context property to value all-bar-numbers-visible.

```
\paper {
  line-width = 50\mm
}

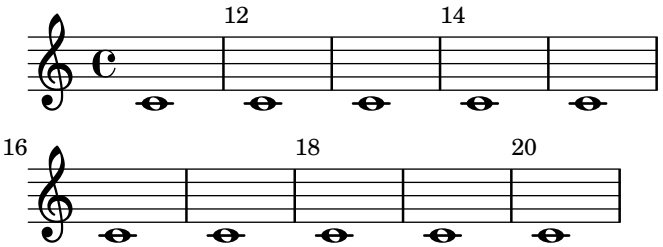
\relative c' {
  \set Score.barNumberVisibility = #all-bar-numbers-visible
  c1 | d | e | f \break
  g1 | e | d | c
}
```



Printing bar numbers at regular intervals

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

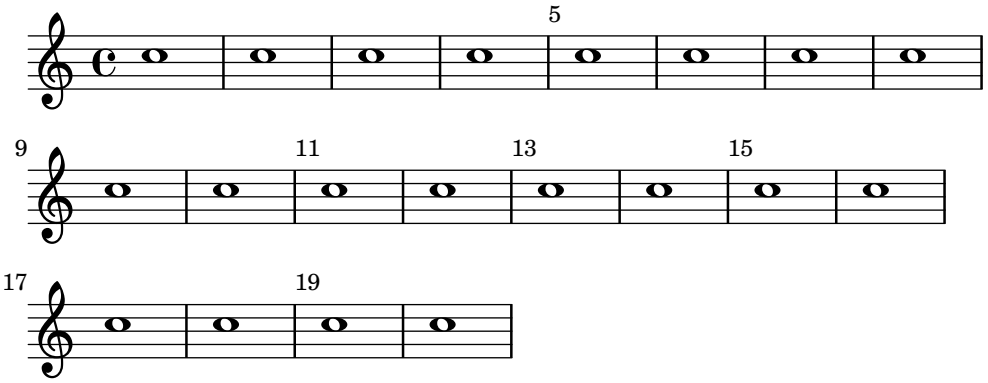
```
\relative c' {
  \override Score.BarNumber.break-visibility = #end-of-line-invisible
  \set Score.currentBarNumber = 11
  % Print a bar number every second measure
  \set Score.barNumberVisibility = #(every-nth-bar-number-visible 2)
  c1 | c | c | c | c
  \break
  c1 | c | c | c | c
}
```



Printing bar numbers with changing regular intervals

Using the `set-bar-number-visibility` context function, bar number intervals can be changed.

```
\relative c' {
  \override Score.BarNumber.break-visibility = #end-of-line-invisible
  \context Score \applyContext #(set-bar-number-visibility 4)
  \*10 c'1
  \context Score \applyContext #(set-bar-number-visibility 2)
  \*10 c
}
```

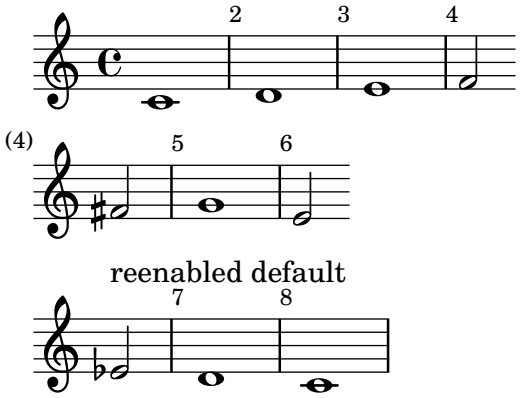


Printing bar numbers for broken measures

By default, a bar number of a broken measure is not repeated at the beginning of the new line. Use `first-bar-number-invisible-save-broken-bars` for `barNumberVisibility` to get a parenthesized `BarNumber` there.

```
\layout {
  \context {
    \Score
    barNumberVisibility = #first-bar-number-invisible-save-broken-bars
    \override BarNumber.break-visibility = ##(#f #t #t)
  }
}
```

```
\relative c' {
  c1 | d | e | f2 \break
  fis2 | g1 | e2 \break
  <>^"reenabled default"
  % back to default -
  % \unset Score.barNumberVisibility would do so as well
  \set Score.barNumberVisibility =
    #first-bar-number-invisible-and-no-parenthesized-bar-numbers
  es2 | d1 | c
}
```



Printing bar numbers using modulo-bar-number-visible

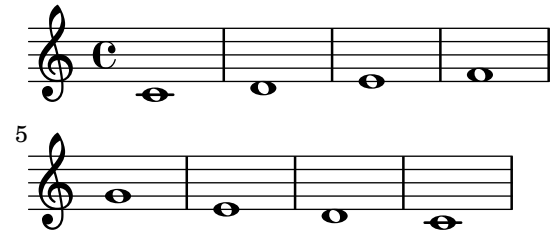
If the remainder of the division of the current bar number by the first argument of `modulo-bar-number-visible` equals its second argument, print a bar number.

This is useful to print the bar number at certain distances. Some examples:

- `(modulo-bar-number-visible 3 2)` → prints 2, 5, 8, ...
- `(modulo-bar-number-visible 4 2)` → prints 2, 6, 10, ...
- `(modulo-bar-number-visible 2 1)` → prints 3, 5, 7, ...
- `(modulo-bar-number-visible 5 0)` → prints 5, 10, 15, ...

```
\layout {
  \context {
    \Score
    \override BarNumber.break-visibility = ##(#f #t #t)
    barNumberVisibility = #(modulo-bar-number-visible 5 0)
  }
}

\relative c' {
  c1 | d | e | f \break
  g1 | e | d | c
}
```



Printing bar numbers inside boxes or circles

Bar numbers can also be printed inside boxes or circles.

```
\relative c' {
  % Center bar numbers except at the beginning of a staff.
  \override Score.BarNumber.self-alignment-X =
    #(break-alignment-list CENTER CENTER 0.3)

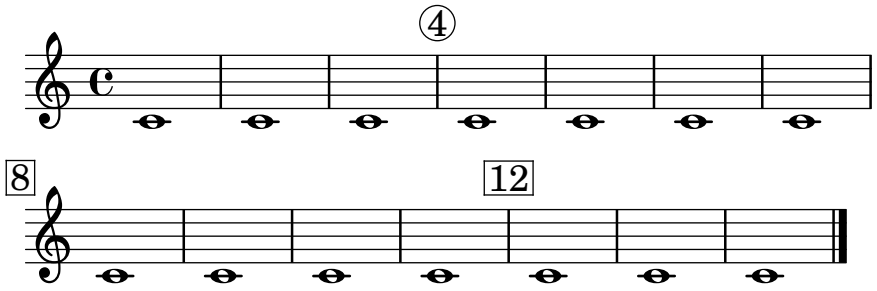
  % Prevent bar numbers at the end of a line and permit them elsewhere.
  \override Score.BarNumber.break-visibility = #end-of-line-invisible

  \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 circle round the following bar number(s).
  \override Score.BarNumber.stencil
    = #(make-stencil-circler 0.1 0.25 ly:text-interface::print)
  \*7 c1 \break

  % Draw a box round the following bar number(s).
  \override Score.BarNumber.stencil
    = #(make-stencil-boxer 0.1 0.25 ly:text-interface::print)
  \*7 c1 \bar "|."
}
```



Alternative bar numbering

Setting the `alternativeNumberingStyle` context property, two additional methods are available for enumerating bar numbers in repeats.

```
music = \relative c' {
  \repeat volta 3 {
    c4 d e f |
    \alternative {
      \volta 1 { c4 d e f | c2 d \break }
      \volta 2 { f4 g a b | f4 g a b | f2 a | \break }
      \volta 3 { c4 d e f | c2 d } } }
  c1 \bar "|."
}
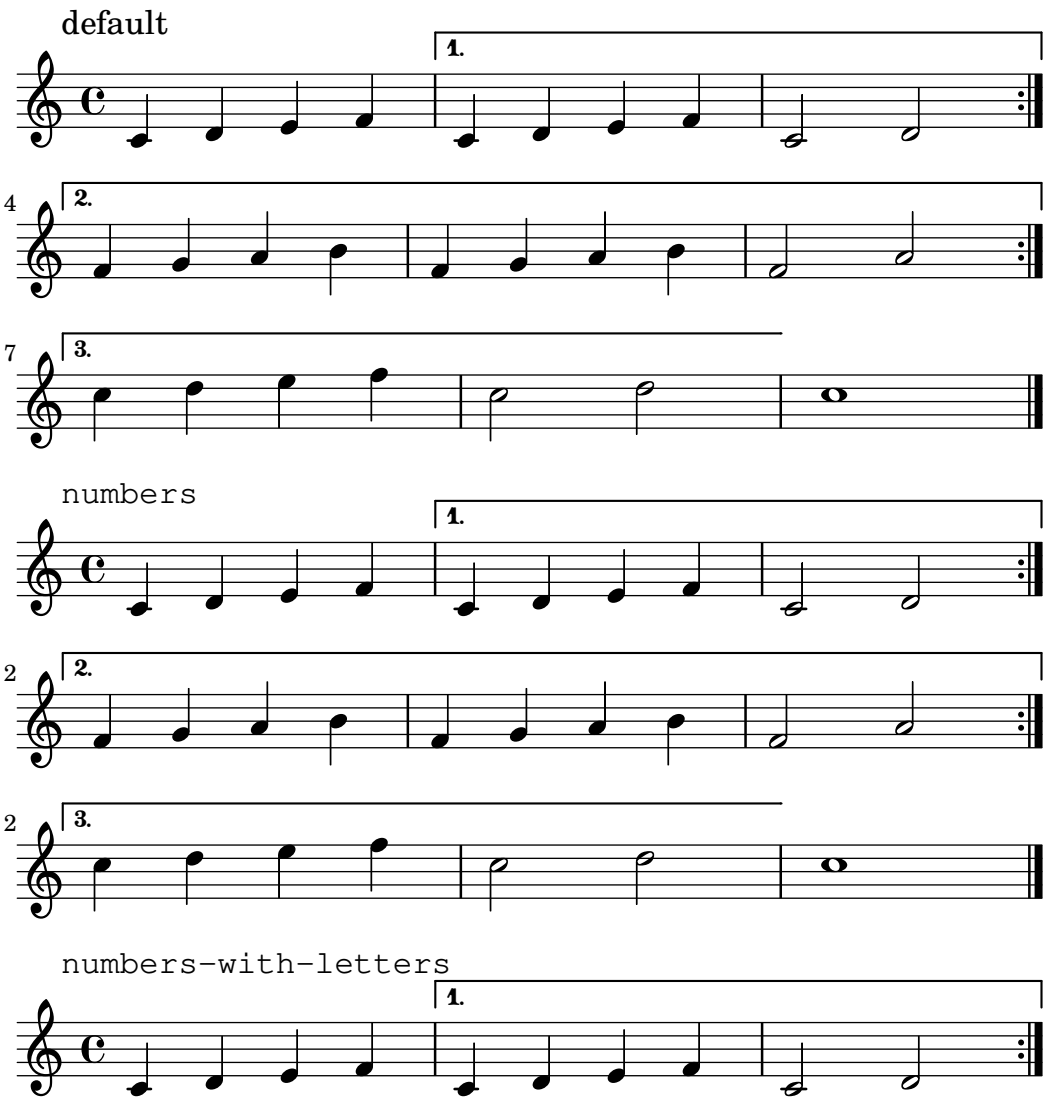
{
  \textMark \markup \large "default"
  \music
```
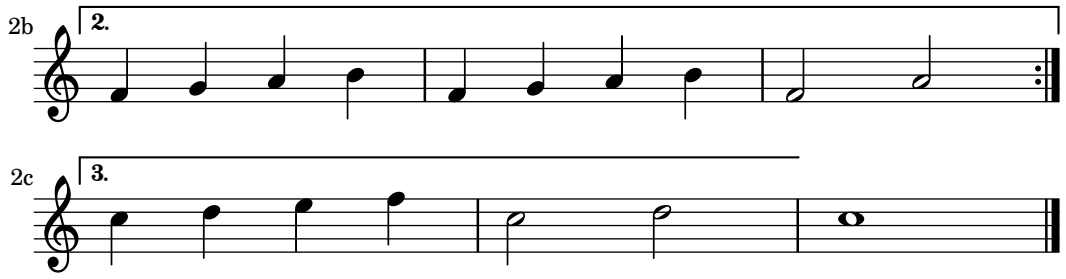
```
}

{
  \textMark \markup \large \typewriter "numbers"
  \set Score.alternativeNumberingStyle = #'numbers
  \music
}

{
  \textMark \markup \large \typewriter "numbers-with-letters"
  \set Score.alternativeNumberingStyle = #'numbers-with-letters
  \music
}

\layout {
  \context {
    \Score
    \override TextMark.Y-offset = #5
  }
}
```





Aligning bar numbers

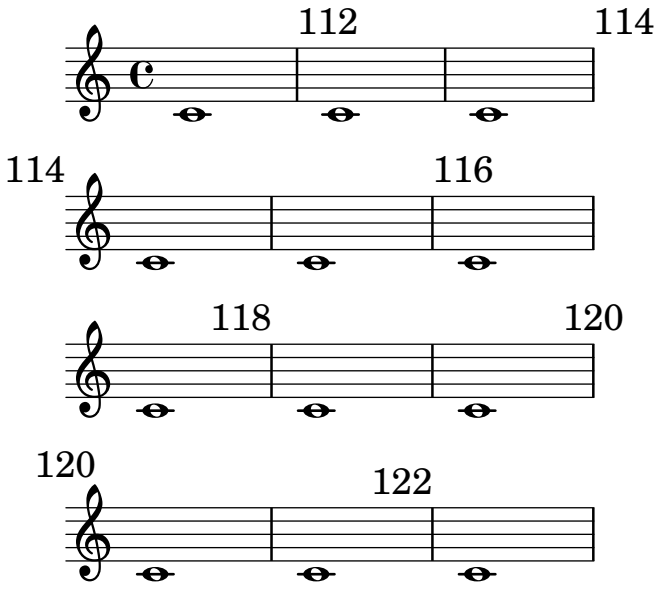
The default alignment of bar numbers depends on its position: at the beginning of a staff, bar numbers are right-aligned; at all other positions, they are left-aligned. Using Scheme function `break-alignment-list`, this can be changed; the three arguments of this function are the alignment for end-of-line, middle-of-line, and start-of-line position (in this order).

```
\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 | c1 | \break
  c1 | c1 | c1 | \break

  \override Score.BarNumber.self-alignment-X =
    #(break-alignment-list CENTER RIGHT CENTER)
  c1 | c1 | c1 | \break
  c1 | c1 | c1 |
}

\paper {
  line-width = 70\mm
}
```

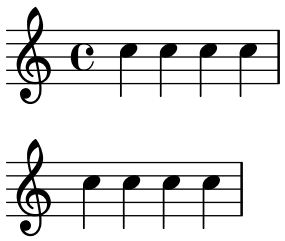


Removing bar numbers from a score

Bar numbers can be removed entirely by removing the `Bar_number_engraver` from the `Score` context.

```
\layout {
  \context {
    \Score
    \omit BarNumber
    % or:
    % \remove "Bar_number_engraver"
  }
}

\relative c'' {
  c4 c c c \break
  c4 c c c
}
```



Measure-centered bar numbers

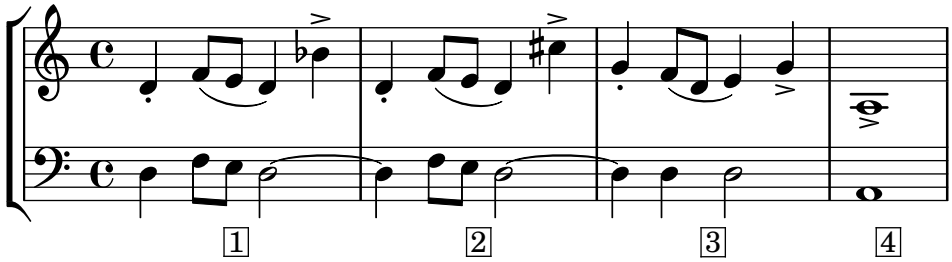
For film scores, a common convention is to center bar numbers within their measure. This is achieved through setting the `centerBarNumbers` context property to `#t`. When this is used, the type of the bar number grobs is `CenteredBarNumber` rather than `BarNumber`.

This example demonstrates a number of settings: the centered bar numbers are boxed and placed below the staves.

```
\layout {
  \context {
    \Score
    centerBarNumbers = ##t
    barNumberVisibility = #all-bar-numbers-visible
    \override CenteredBarNumber.stencil
      = #(make-stencil-boxer 0.1 0.25 ly:text-interface::print)
    \override CenteredBarNumberLineSpanner.direction = #DOWN
  }
}

\new StaffGroup <<
  \new Staff \relative c' {
    d4-. f8( e d4) bes'-> |
    d,-. f8( e d4) cis'-> |
    g-. f8( d e4) g-> |
    a,1-> |
  }
  \new Staff \relative c {
    \clef bass
```

```
    d4 f8 e d2~ |
    4 f8 e d2~ |
    4 4 2 |
    a1 |
  }
>>
```



See also

Snippets: Section "Rhythms" in *Snippets*.

Internals Reference: Section "BarNumber" in *Internals Reference*, Section "CenteredBarNumber" in *Internals Reference*, Section "CenteredBarNumberLineSpanner" in *Internals Reference*, Section "Bar_number_engraver" in *Internals Reference*, Section "Centered_bar_number_align_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.

2.5.4 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 |
```

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.

Bar checks can also be inserted in lyrics:
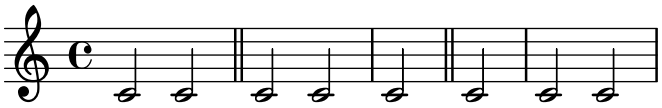
```
\lyricmode {
  \time 2/4
  Twin -- kle | Twin -- kle |
}
```

Note that bar check marks in lyrics are evaluated at the musical moment when the syllable *following* the check mark is processed. If the lyrics are associated with the notes of a voice which has a rest at the beginning of a bar, then no syllable can be located at the start of that bar and a warning will be issued if a bar check mark is placed in the lyrics at that position.

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 "|". 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.

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



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.

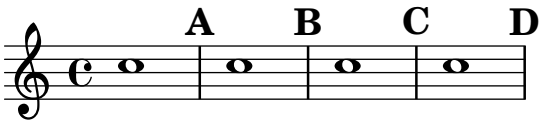
See also

Snippets: Section "Rhythms" in *Snippets*.

2.5.5 Rehearsal marks

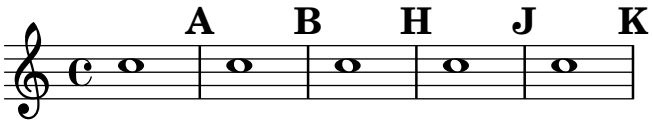
To print a rehearsal mark, use the `\mark` command.

```
\relative c'' {
  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`.

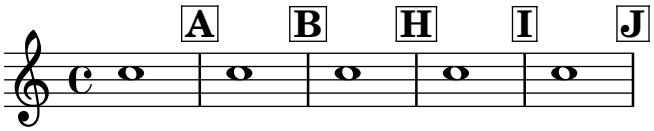
```
\relative c'' {
  c1 \mark \default
  c1 \mark \default
  c1 \mark 8
  c1 \mark \default
  c1 \mark \default
}
```



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).

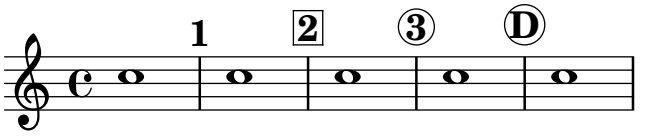
```
\set Score.rehearsalMarkFormatter = #format-mark-alphabet
\set Score.rehearsalMarkFormatter = #format-mark-box-alphabet
\set Score.rehearsalMarkFormatter = #format-mark-circle-alphabet
```

```
\relative c'' {
  \set Score.rehearsalMarkFormatter = #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 `rehearsalMarkFormatter`. 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, `rehearsalMarkFormatter` is set to a predefined procedure. After a few measures, it is set to a procedure that produces a boxed number.

```
\relative c'' {
  \set Score.rehearsalMarkFormatter = #format-mark-numbers
  c1 \mark \default
  c1 \mark \default
  \set Score.rehearsalMarkFormatter = #format-mark-box-numbers
  c1 \mark \default
  \set Score.rehearsalMarkFormatter = #format-mark-circle-numbers
  c1 \mark \default
  \set Score.rehearsalMarkFormatter = #format-mark-circle-letters
  c1
}
```



The file `scm/translation-functions.scm` contains the definitions of `format-mark-letters` (the default format), `format-mark-box-letters`, `format-mark-numbers`, and `format-mark-box-numbers`. 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.

For common tweaks to the positioning of rehearsal marks, see Section 8.2 [Formatting text], page 320. For more precise control, consider `break-alignable-interface` (see Section 36.9 [Aligning objects], page 791).

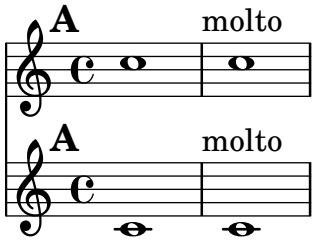
The file `scm/translation-functions.scm` contains the definitions of `format-mark-numbers` and `format-mark-letters`. They can be used as inspiration for other formatting functions.

Selected snippets

Printing marks on every staff

Although rehearsal and text marks are normally only printed above the topmost staff, they may also be printed on every staff.

```
\score {
  <<
    \new Staff { \mark \default c''1 \textMark "molto" c'' }
    \new Staff { \mark \default c'1 \textMark "molto" c' }
  >>
  \layout {
    \context {
      \Score
      \remove Mark_engraver
      \remove Text_mark_engraver
      \remove Staff_collecting_engraver
    }
    \context {
      \Staff
      \consists Mark_engraver
      \consists Text_mark_engraver
      \consists Staff_collecting_engraver
    }
  }
}
```



See also

Notation Reference: Section B.8 [The Emmentaler font], page 907, Section 8.2 [Formatting text], page 320, Section 36.9 [Aligning objects], page 791, Section 8.2.5 [Music notation inside markup], page 335, Section 2.3.3 [Metronome marks], page 87, Section 8.1.4 [Section labels], page 313.

Installed Files: `scm/translation-functions.scm`.

Snippets: Section "Rhythms" in *Snippets*.
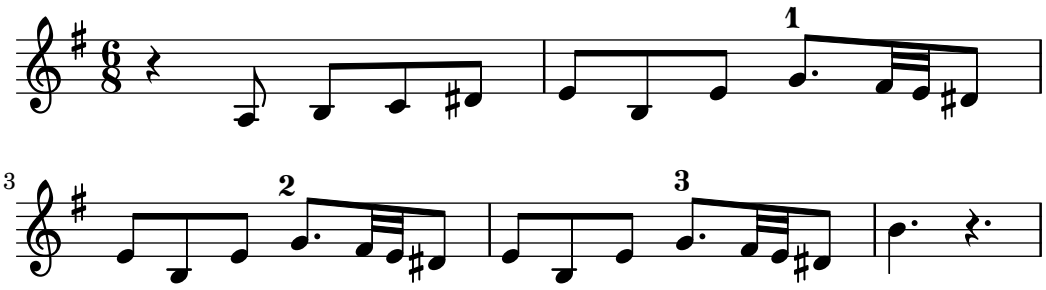
Internals Reference: Section "AdHocMarkEvent" in *Internals Reference*, Section "RehearsalMark" in *Internals Reference*, Section "RehearsalMarkEvent" in *Internals Reference*.

2.5.6 Measure counts

Measure counts are a way to number consecutive measures, for example as an aid for musicians to count measures in written-out repeats. Using this feature requires adding the `Measure_counter_engraver` to a context type, usually `Staff` or `Score`.

```
\layout {
  \context {
    \Staff
    \consists Measure_counter_engraver
  }
}
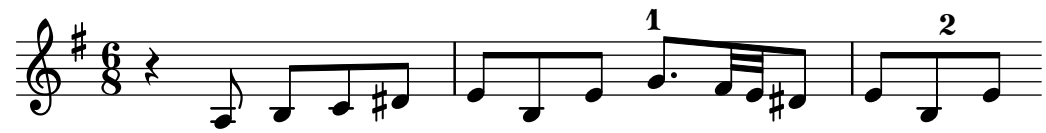
\relative c' {
  \time 6/8
  \key e \minor
  r4 a8 b c dis
  \startMeasureCount
  \*3 {
    e8 b e g8. fis32 e dis8
  }
  \stopMeasureCount
  b'4. r
}
```
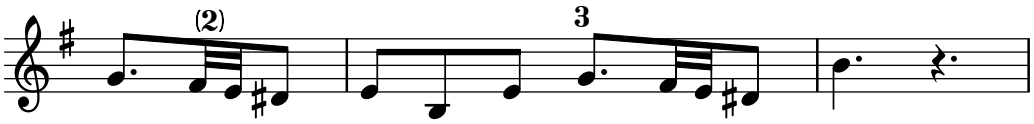


Broken measures are numbered in parentheses.

```
\layout {
  \context {
    \Staff
    \consists Measure_counter_engraver
  }
}

\relative c' {
  \time 6/8
  \key e \minor
  r4 a8 b c dis
  \startMeasureCount
  e8 b e g8. fis32 e dis8
  e8 b e \break g8. fis32 e dis8
  e8 b e g8. fis32 e dis8
  \stopMeasureCount
  b'4. r
}
```
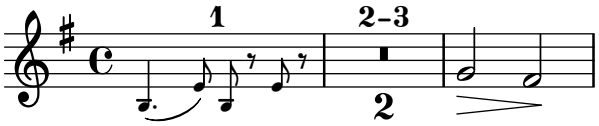




Compressed multi-measure rests receive special treatment: the full measure range is shown.

```
\layout {
  \context {
    \Staff
    \consists Measure_counter_engraver
  }
  \context {
    \Voice
    \override MultiMeasureRestNumber.direction = #DOWN
  }
}

\compressMMRests {
  \key e \minor
  \startMeasureCount
  \new CueVoice {
    b4.( e'8) b8 r e' r
  }
  R1*2
  \stopMeasureCount
  g'2\> fis'2\!
}
```

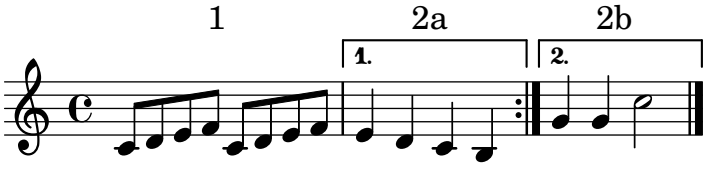


Measure counters honor alternative numbering styles. If the style is `numbers-with-letters`, they render best with a textual font.

```
\layout {
  \context {
    \Score
    alternativeNumberingStyle = #'numbers-with-letters
  }
  \context {
    \Staff
    \consists Measure_counter_engraver
    \override MeasureCounter.Y-offset = 6
    \override MeasureCounter.font-encoding = #'latin1
    \override MeasureCounter.font-size = 1
  }
}

\relative c' {
  \startMeasureCount
  \repeat volta 2 {
    c8 d e f c d e f
  }
```

```
  \alternative {
    { e4 d c b }
    { g'4 g c2 }
  }
  \bar "|."
  \stopMeasureCount
}
```



Predefined commands

`\startMeasureCount`, `\stopMeasureCount`.

See also

Notation Reference: Section 33.4 [Modifying context plug-ins], page 745, Section 6.3.4 [Compressing empty measures], page 274, Chapter 28 [Breaks], page 689, Section 2.5.3 [Bar numbers], page 133.

Internals Reference: Section "Measure_counter_engraver" in *Internals Reference*, Section "MeasureCounter" in *Internals Reference*, Section "measure-counter-interface" in *Internals Reference*.

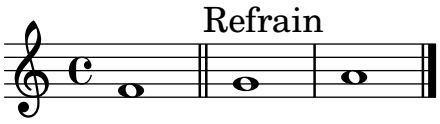
2.5.7 Section divisions

The `\section` command marks a point where one section of music ends and another begins. It does not have to be followed by more music: it may also be used to emphasize that the written end of the music is not the end of the piece, such as at a *D.C.* instruction or where one movement continues into the next without a break. `\section` normally creates a double bar line, but its effect can depend on other notation, e.g., repeat bar lines.

The `\fine` command ends the piece, normally with a final bar line. It is not limited to use at the written end of the music: it may also appear inside `\repeat` (see Section 4.1.5 [Al-fine repeats], page 195).

A section can optionally be named with `\sectionLabel` (see Section 8.1.4 [Section labels], page 313).

```
\fixed c' {
  f1
  \section
  \sectionLabel "Refrain"
  g1
  a1
  \fine
}
```



For details on interactions of `\fine` and `\section` bar lines with other types of bar lines, and options for changing their appearance, see Section 2.5.2 [Automatic bar lines], page 131.

See also

Music Glossary: Section “fine” in *Music Glossary*.

Notation Reference: Section 2.5.2 [Automatic bar lines], page 131, Section 17.4.4 [Divisiones], page 556, Section 4.1.5 [Al-fine repeats], page 195, Section 8.1.4 [Section labels], page 313.
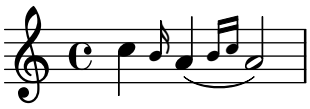
Internals Reference: Section “FineEvent” in *Internals Reference*, Section “SectionEvent” in *Internals Reference*.

2.6 Special rhythmic concerns

2.6.1 Grace notes
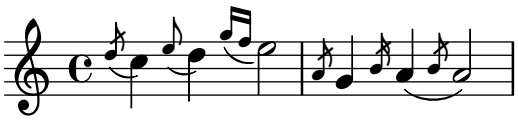
Grace notes are musical ornaments, printed in a smaller font, that take up no additional logical time in a measure.

```
\relative {
  c''4 \grace b16 a4(
  \grace { b16 c16 } a2)
}
```
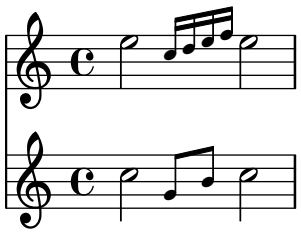


There are three other types of grace notes possible; the *acciaccatura* – an unmeasured grace note indicated by a slurred note with a slashed stem – and the *appoggiatura*, which takes a fixed fraction of the main note it is attached to and prints without the slash. It is also possible to write a grace note with a slashed stem, like the *acciaccatura* but without the slur, so as to place it between notes that are slurred themselves, using the `\slashedGrace` function.

```
\relative {
  \acciaccatura d''8 c4
  \appoggiatura e8 d4
  \acciaccatura { g16 f } e2
  \slashedGrace a,8 g4
  \slashedGrace b16 a4(
  \slashedGrace b8 a2)
}
```



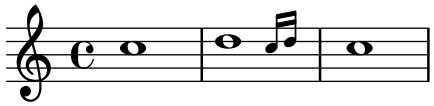
The placement of grace notes is synchronized between different staves. In the following example, there are two sixteenth grace notes for every eighth grace note

```
<<
  \new Staff \relative { e''2 \grace { c16 d e f } e2 }
  \new Staff \relative { c''2 \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.

```
\relative { c''1 \afterGrace d1 { c16[ d] } c1 }
```
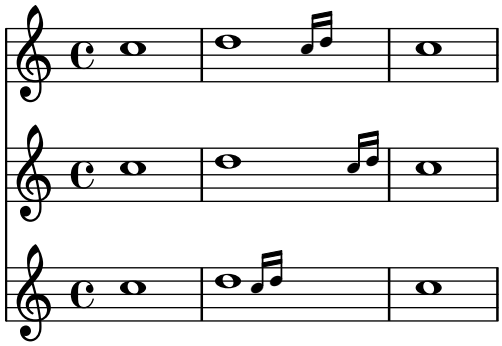


This will place the grace notes *after* the start of the main note. The point of time where the grace notes are placed is a given fraction of the main note's duration. The default setting of

```
afterGraceFraction = 3/4
```

may be redefined at top level. Individual \afterGrace commands may have the fraction specified right after the command itself instead.
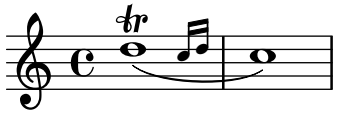
The following example shows the results from setting with the default space, setting it at 15/16, and finally at 1/2 of the main note.

```
<<
  \new Staff \relative {
    c''1 \afterGrace d1 { c16[ d] } c1
  }
  \new Staff \relative {
    c''1 \afterGrace 15/16 d1 { c16[ d] } c1
  }
  \new Staff \relative {
    c''1 \afterGrace 1/2 d1 { c16[ d] } c1
  }
>>
```
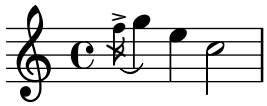


The effect of \afterGrace can also be achieved using spacers. The following example places the grace note after a space lasting 7/8 of the main note.

```
\new Voice \relative {
  <<
    { d''1^\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.
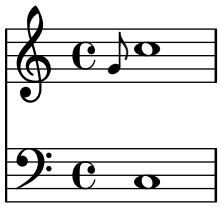
```
\new Voice \relative {
  \acciaccatura {
    \stemDown
    f''16->
    \stemNeutral
  }
  g4 e c2
}
```
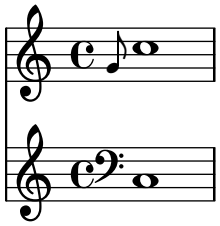


Since grace notes are optional material preceding main note material, often grace notes occur in one Voice or Staff but not another. Non-note material like clefs, key changes, page breaks, bars, property setting commands, section marks and so on is generally expected to be placed *before* grace notes occuring in other voices, so that is what LilyPond does by default. If you want LilyPond to refrain from moving such material before grace notes in parallel tracks, placing the command \rigid before them will prevent such loose timing.

```
<<
  \new Staff { \grace g'8 c''1 }
  \new Staff { \clef "bass" c1 }
>>

<<
  \new Staff { \grace g'8 c''1 }
  \new Staff { \rigid \clef "bass" c1 }
>>
```
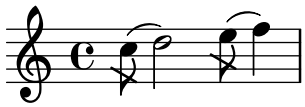




Selected snippets

Using grace note slashes with normal heads

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

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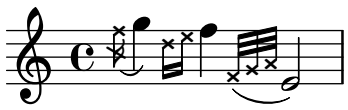


Tweaking grace layout within music

The appearance of grace expressions can be changed by using the functions `add-grace-property` and `remove-grace-property`.

The following example undefines the `direction` property of `Stem` grobs 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 following identifiers.
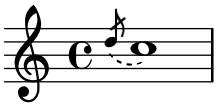
```
startGraceMusic
stopGraceMusic
startAcciaccaturaMusic
stopAcciaccaturaMusic
startAppoggiaturaMusic
stopAppoggiaturaMusic
```

They are defined in file `ly/grace-init.ly`. By redefining them other effects may be obtained.

```
startAcciaccaturaMusic = {
  <>(
  \override Flag.stroke-style = "grace"
  \slurDashed
}

stopAcciaccaturaMusic = {
  \revert Flag.stroke-style
  \slurSolid
  <>)
}

\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.

Due to Issue #6876 (`https://gitlab.com/lilypond/lilypond/-/issues/6876`), however, accidentals are ignored if this property is set. This snippet gives a workaround to circumvent the problem.

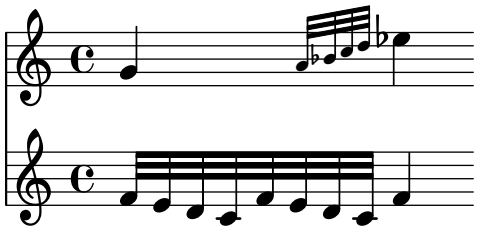
Another unfortunate side effect of this property is that LilyPond does not check whether there is enough horizontal space for grace notes (this is tracked as Issue #2630 (`https://gitlab.com/lilypond/lilypond/-/issues/2630`)). You have to make sure that enough space is available, for example, by using `\newSpacingSection` together with a proper value for the `base-shortest-duration` of the `SpacingSpanner` grob.

```
shiftedGrace =
#(define-music-function (offset music) (number? ly:music?)
   #{
     \override NoteHead.X-offset = #(- offset 0.85)
     \override Stem.X-offset = #offset
     \grace { $music }
     \revert NoteHead.X-offset
     \revert Stem.X-offset
   #})

\relative c'' <<
  { g4 \shiftedGrace #-1.3 a32 \shiftedGrace #-0.5 { bes c d } es4 }
  { f,32 e d c f e d c f4 }
>>

\layout {
  \context {
    \Score
    \override SpacingSpanner.strict-grace-spacing = ##t
```

```
  }
}
```



See also

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

Notation Reference: Section 2.1.3 [Scaling durations], page 61, Section 2.4.3 [Manual beams], page 115.

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_auto_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 sections should only be used within sequential music expressions. Nesting grace sections is not supported.

Each grace note in MIDI output has a length of 1/4 of its actual duration. If the combined length of the grace notes is greater than the length of the preceding note a “`Going back in MIDI time`” error will be generated. Either make the grace notes shorter in duration, for example:

```
c'8 \acciaccatura { c'8[ d' e' f' g'] }
```

becomes:

```
c'8 \acciaccatura { c'16[ d' e' f' g'] }
```

Or explicitly change the musical duration:

```
c'8 \acciaccatura { \scaleDurations 1/2 { c'8[ d' e' f' g'] } }
```

See Section 2.1.3 [Scaling durations], page 61.

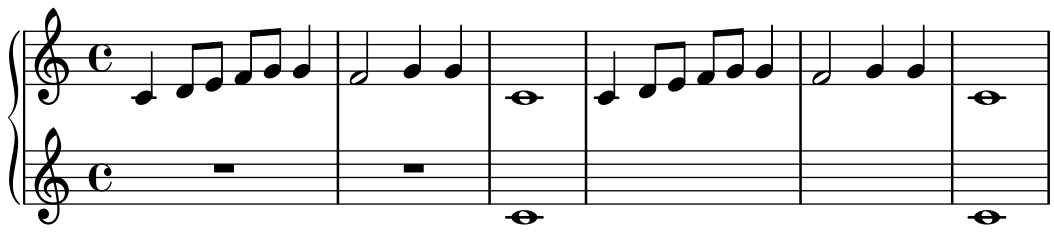
2.6.2 Aligning to cadenzas

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'4 d8 e f g g4
  f2 g4 g
}

\new GrandStaff <<
  \new Staff {
    \MyCadenza c'1
    \MyCadenza c'1
  }
  \new Staff {
    #(mmrest-of-length MyCadenza)
    c'1
    #(skip-of-length MyCadenza)
    c'1
  }
>>
```

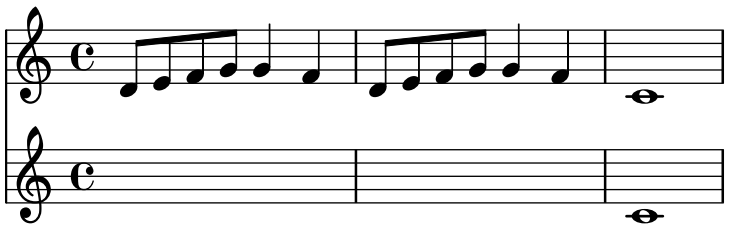


Another solution is to use the \skip command.

```
MyCadenza = \fixed c' {
  \repeat volta 2 {
    d8 e f g g4 f4
  }
}

music = <<
  \new Staff {
    \MyCadenza
    c'1
  }
  \new Staff {
    \skip \MyCadenza
    c'1
  }
>>

\unfoldRepeats \music
```
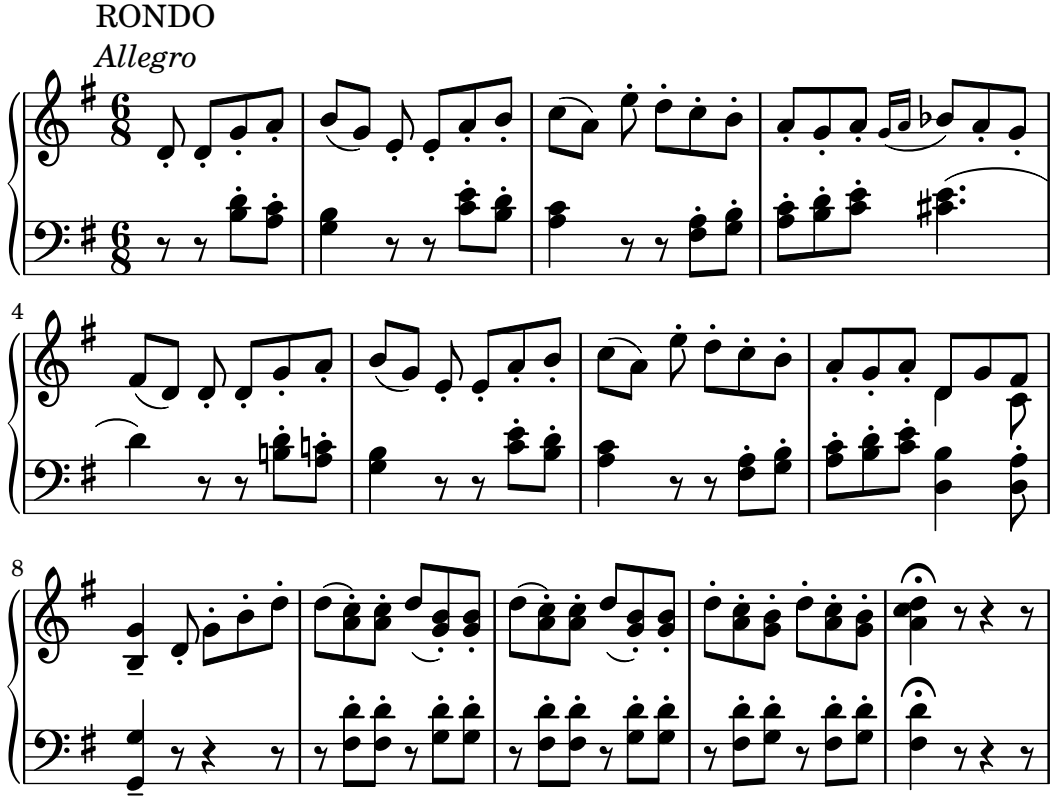


See also

Music Glossary: Section “cadenza” in *Music Glossary*.

Snippets: Section “Rhythms” in *Snippets*.

3 Expressive marks



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

3.1 Expressive marks 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.
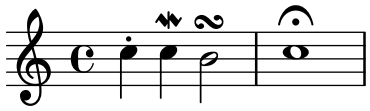
3.1.1 Articulations and ornamentations

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

*note**name*
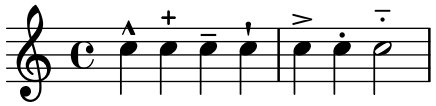
The possible values for *name* are listed in Section B.13 [List of articulations], page 928. For example:

```
\relative {
  c''4\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:
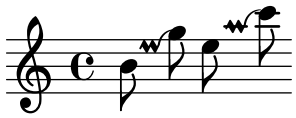
```
\relative {
  c''4-^ c-+ c-- c-!
  c4-> c-. c2-_
}
```



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 36.1 [Direction and placement], page 774.

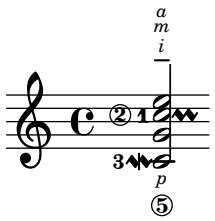
The `bachschleifer` is positioned to the left of the `NoteHead`. This ornament may have ledger lines.

```
{
  \autoBeamOff
  b'8 g''\bachschleifer
  e'' c'''\bachschleifer
}
```



It is also possible to position common articulations and ornamentations to the left or right of a note head by overriding their `side-axis` and, if necessary, the `direction` property. The convenience functions `\atLeft` or `\atRight` take care of this.

```
{
  \set fingeringOrientations = #'(left)
  \set stringNumberOrientations = #'(left down)
  <
   c'-3\5_\rightHandFinger #1 \atLeft \mordent
   g'^\rightHandFinger #2
   c''-1\2^\rightHandFinger #3 \atRight \prall
   e'' ^\rightHandFinger #4
  >2^\tenuto
}
```
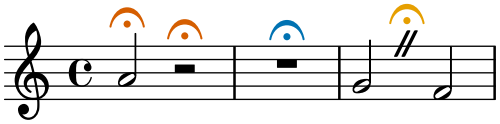


Predefined commands

`\atLeft`, `\atRight`.

The type of grob that an articulation creates depends on what it is attached to.

- On notes or ordinary rests, articulations create `Script` objects.
- On multi-measure rests, articulations create `MultiMeasureRestScript` objects.
- On `\caesura`, articulations create `CaesuraScript` objects.

```
\override Score.Script.color = #(universal-color 'vermillion)
\override Score.MultiMeasureRestScript.color = #(universal-color 'blue)
\override Score.CaesuraScript.color = #(universal-color 'orange)
a'2\fermata r\fermata
R1\fermata
g'2 \caesura \fermata f'2
```



In addition to articulations, text and markups can be attached to notes. See Section 8.1.2 [Text scripts], page 310.

For more information about the ordering of `Script` and `TextScript` grobs that are attached to notes, see Section "Placement of objects" in *Learning Manual*.

Selected snippets

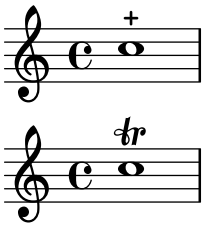
Modifying default values for articulation shorthand notation

The shorthands are defined in `ly/script-init.ly`, where the variables `dashHat`, `dashPlus`, `dashDash`, `dashBang`, `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.
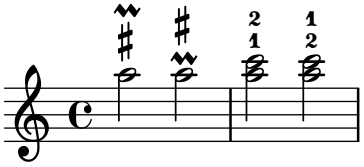
Note that for `Fingering`, `StringNumber`, and `StrokeFinger` grobs, if used within a chord, the vertical order is also determined by the vertical position of the associated note head, which is added to (or, depending on the direction, subtracted from) the grob's `script-priority` value. This ensures that for fingerings above a chord the lower note is associated with the lower fingering (and vice versa for the other direction); it doesn't matter whether you input the notes in the chord from top to bottom or from bottom to top.

By default, the least technical scripts are positioned closest to the note head; the rough order is articulation, flageolet, fingering, right-hand fingering, string number, fermata, bowing, and text script.

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

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

  \set fingeringOrientations = #'(up)
  <c-2 a-1>2
  <a-1 c\tweak script-priority -100 -2>2
}
```

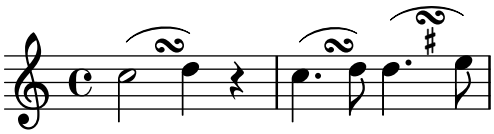


See Section B.18 [Default values for script-priority], page 938.

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. Changing the first argument of \after (which is a duration) adjusts the horizontal position.

```
\relative c'' {
  \after 2*2/3 \turn c2( d4) r |
  \after 4 \turn c4.( d8)
  \after 4
  {
    \once \set suggestAccidentals = ##t
    \once \override AccidentalSuggestion.outside-staff-priority = ##f
    \once \override AccidentalSuggestion.avoid-slur = #'inside
    \once \override AccidentalSuggestion.font-size = -3
    \once \override AccidentalSuggestion.script-priority = -1
    \once \hideNotes
    cis8\turn \noBeam
  }
  d4.( e8)
}
```



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: Section 8.1.2 [Text scripts], page 310, Section 36.1 [Direction and placement], page 774, Section B.13 [List of articulations], page 928, Section 3.3.3 [Trills], page 186.

Installed Files: `scm/script.scm`.

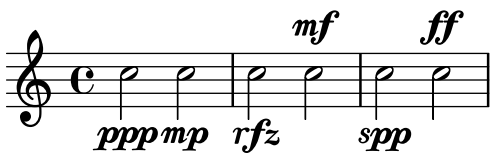
Snippets: Section "Expressive marks" in *Snippets*.

Internals Reference: Section "CaesuraScript" in *Internals Reference*, Section "MultiMeasureRestScript" in *Internals Reference*, Section "Script" in *Internals Reference*, Section "TextScript" in *Internals Reference*.

3.1.2 Dynamics

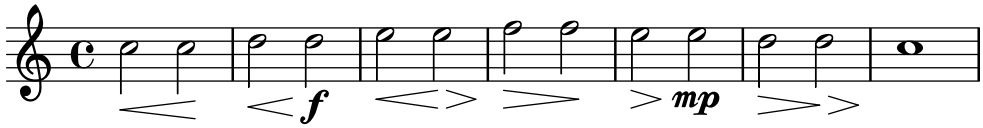
Absolute dynamic marks are specified using a command after a note, such as `c4\ff`. The available dynamic marks are `\ppppp`, `\pppp`, `\ppp`, `\pp`, `\p`, `\mp`, `\mf`, `\f`, `\ff`, `\fff`, `\ffff`, `\fffff`, `\fp`, `\sf`, `\sff`, `\sp`, `\spp`, `\sfz`, `\rfz`, and `\n`. Dynamic marks may be manually placed above or below the staff; see Section 36.1 [Direction and placement], page 774.

```
\relative c'' {
  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 `\>`; `\endcr` and `\enddecr` maybe used instead of `\!` to end a crescendo or decrescendo mark, respectively. *Hairpins* are engraved by default using this notation.

```
\relative c'' {
  c2\< c\!
  d2\< d\f
  e2\< e\>
  f2\> f\!
  e2\> e\mp
  d2\> d\>
  c1\!
}
```
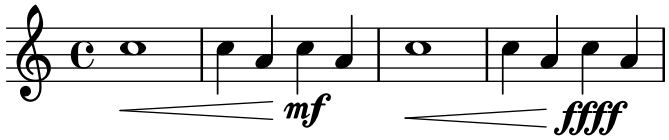


A hairpin that is terminated with `\!` will end at the right edge of the note that has the `\!` assigned to it. In the case where it is terminated with the start of another *crescendo* or *decrescendo* mark, it will end at the center of the note that has the next `\<` or `\>` assigned to it. The next hairpin will then start at the right edge of the same note instead of the usual left edge had it been terminated with `\!` before. A hairpin ending on a downbeat will stop at the preceding bar line.

```
\relative {
  c''1\< | c4 a c\< a | c4 a c\! a\< | c4 a c a\!
}
```



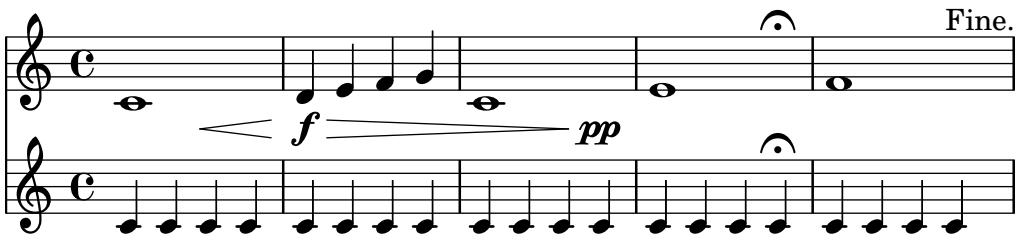
Hairpins that are terminated with absolute dynamic marks instead of `\!` will also be engraved in a similar way. However, the length of the absolute dynamic itself can alter where the preceding hairpin ends.

```
\relative {
  c''1\< | c4 a c\mf a | c1\< | c4 a c\ffff a
}
```



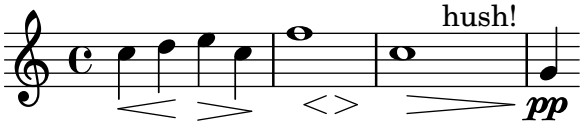
Often, marks like *crescendo* or *decrescendo* should begin or end at some point of time during a sustained note. This can be achieved with `\after`, which can also be used to create delayed articulations or text scripts:

```
<<
  \relative {
    \after 2 \< c'1
    d4\f\> e f g
    \after 2. \pp c,1
    \after 2. \fermata e
    \after 2. ^"Fine." f
  }
  \relative {
    \*12 c'4
    c c c c\fermata
    c c c c
  }
>>
```



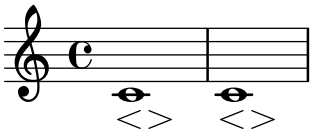
Multiple instances of `\after` can be used to engrave multiple marks on one note. This is particularly useful when adding a crescendo and decrescendo to the same note:

```
\relative {
  c''4\< d\! e\> c\!
  \after 4 \< \after 2\> \after 2. \! f1
  \textLengthOn
  \after 4 \> \after 2. ^"hush!" c1
  g4\pp
}
```
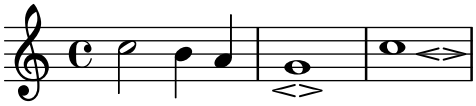


If the first in such a sequence of marks on a single note is supposed to coincide with the onset of the note, it is convenient to attach it to an empty chord <>. This way, all marks can be entered in their natural visual order:

```
{
  <>\< \after 4 \> \after 2 \! c'1
  % easier to write and read than:
  \after 4 \> \after 2 \! c'1\<
}
```
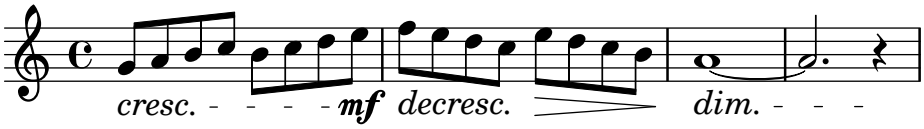


The \espressivo command can also be used to indicate a crescendo and decrescendo on the same note. However, be warned that this is implemented as an articulation, not a dynamic.

```
\relative {
  c''2 b4 a
  g1\espressivo
  \after 2. \espressivo c
}
```
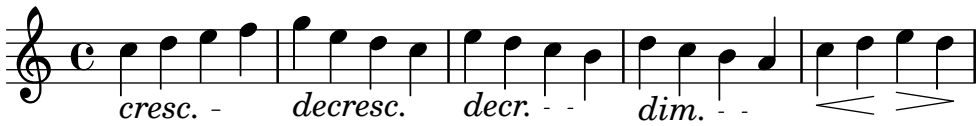


Textual crescendo marks begin with \cresc. Textual decrescendos begin with \decresc or \dim. Extender lines are engraved as required.

```
\relative {
  g'8\cresc a b c b c d e\mf |
  f8\decresc e d c e\> d c b |
  a1\dim ~ |
  a2. r4\! |
}
```



Textual marks for dynamic changes can also replace hairpins:

```
\relative c'' {
  \crescTextCresc
  c4\< d e f\! |
  \dimTextDecresc
  g4\> e d c\! |
  \dimTextDecr
  e4\> d c b\! |
  \dimTextDim
  d4\> c b a\! |
  \crescHairpin
  \dimHairpin
  c4\< d\! e\> d\! |
}
```
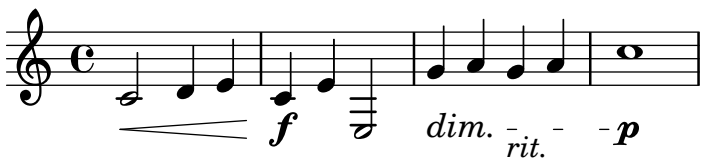


To create new absolute dynamic marks or text that should be aligned with dynamics, see Section 3.1.3 [New dynamic marks], page 168.

Vertical positioning of dynamics is handled by Section "DynamicLineSpanner" in *Internals Reference*.

A `Dynamics` context is available to engrave dynamics on their own horizontal line. Use spacer rests to indicate timing. (Notes in a `Dynamics` context will also take up musical time, but will not be engraved.) The `Dynamics` context can usefully contain some other items such as text scripts, text spanners, and piano pedal marks.

```
<<
  \new Staff \relative {
    c'2 d4 e |
    c4 e e,2 |
    g'4 a g a |
    c1 |
  }
  \new Dynamics {
    s1\< |
    s1\f |
    s2\dim s2-"rit." |
    s1\p |
  }
>>
```



Note: Even if there is only a single absolute dynamic mark like `\p` in a score, LilyPond always creates *two* objects for it, a `DynamicText` and a `DynamicLineSpanner` object, and the properties to control the dynamic mark are shared between these two objects. For example, the size can be changed with the `DynamicText.font-size` property, while the vertical position is controlled by `DynamicLineSpanner.Y-offset`.

As a consequence, code like

```
\tweak font-size 5 \p
```

works but

```
\tweak Y-offset 5 \p
```

does not. You have to say

```
\tweak DynamicLineSpanner.Y-offset 5 \p
```

instead.

Predefined commands

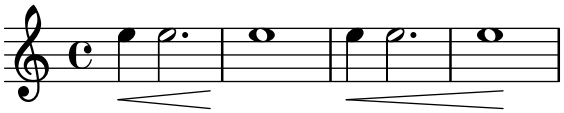
`\dynamicUp`, `\dynamicDown`, `\dynamicNeutral`, `\crescTextCresc`, `\dimTextDim`, `\dimTextDecr`, `\dimTextDecresc`, `\crescHairpin`, `\dimHairpin`.

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.

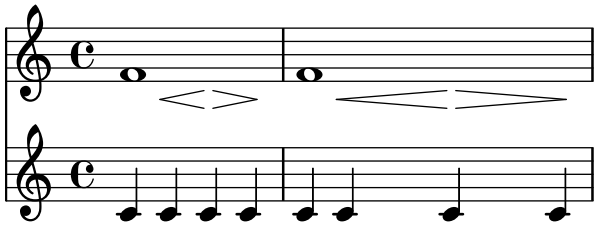
```
\relative c'' {
  e4\< e2.
  e1\!
  \override Hairpin.to-barline = ##f
  e4\< e2.
  e1\!
}
```



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.
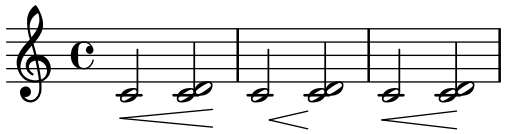
```
<<
  {
    \after 4 \< \after 2 \> \after 2. \! f'1
    \override Hairpin.minimum-length = 8
    \after 4 \< \after 2 \> \after 2. \! f'1
  }
  {
    \*8 c'4
  }
>>
```



Aligning the ends of hairpins to NoteColumn directions

The ends of hairpins may be aligned to the `LEFT`, `CENTER`, or `RIGHT` of `NoteColumn` grobs by overriding the property `endpoint-alignments`, which is a pair of numbers representing the left and right ends of the hairpin. `endpoint-alignments` are expected to be directions (either -1, 0 or 1). Other values will be transformed with a warning. The right end of a hairpin terminating at a rest is not affected, always ending at the left edge of the rest.

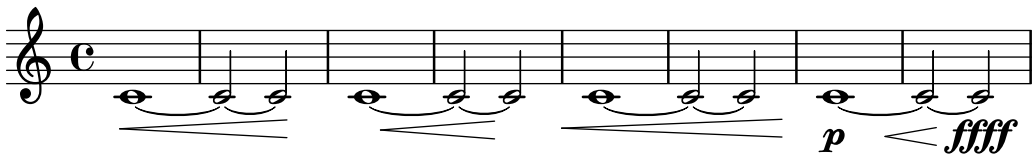
```
{
  c'2\< <c' d'>\! |
  \override Hairpin.endpoint-alignments = #'(1 . -1)
  c'2\< <c' d'>\! |
  \override Hairpin.endpoint-alignments = #`(,LEFT . ,CENTER)
  c'2\< <c' d'>\! |
}
```



Moving the ends of hairpins

The ends of hairpins may be offset by setting the `shorten-pair` property of the `Hairpin` object. Positive values move endpoints to the right, negative to the left. Unlike the `minimum-length` property, this property only affects the appearance of the hairpin; it does not adjust horizontal spacing (including the position of bounding dynamics). This method is thus suitable for fine-tuning a hairpin within its allotted space.
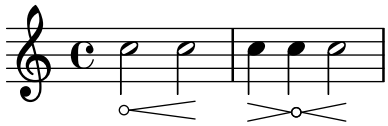
```
{
  c'1~\<
  c'2~ c'\!
  \once \override Hairpin.shorten-pair = #'(2 . 2)
  c'1~\<
  c'2~ c'\!
  \once \override Hairpin.shorten-pair = #'(-2 . -2)
  c'1~\<
  c'2~ c'\!
  c'1~\p-\tweak shorten-pair #'(2 . 0)\<
  c'2~ c'\ffff
}
```



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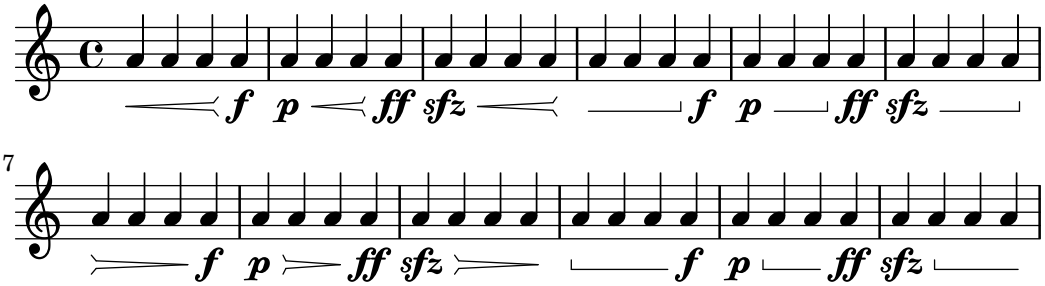
```
\relative c'' {
  \override Hairpin.circled-tip = ##t
  c2\< c\!
  c4\> c\< c2\!
}
```



Printing hairpins in various styles

Hairpin dynamics may be created in a variety of styles.

```
\relative c'' {
  \override Hairpin.stencil = #flared-hairpin
  a4\< a a a\f
  a4\p\< a a a\ff
  a4\sfz\< a a a\!
  \override Hairpin.stencil = #constante-hairpin
  a4\< a a a\f
  a4\p\< a a a\ff
  a4\sfz\< a a a\!
  \override Hairpin.stencil = #flared-hairpin
  a4\> a a a\f
  a4\p\> a a a\ff
  a4\sfz\> a a a\!
  \override Hairpin.stencil = #constante-hairpin
  a4\> a a a\f
  a4\p\> a a a\ff
  a4\sfz\> a a a\!
}
```



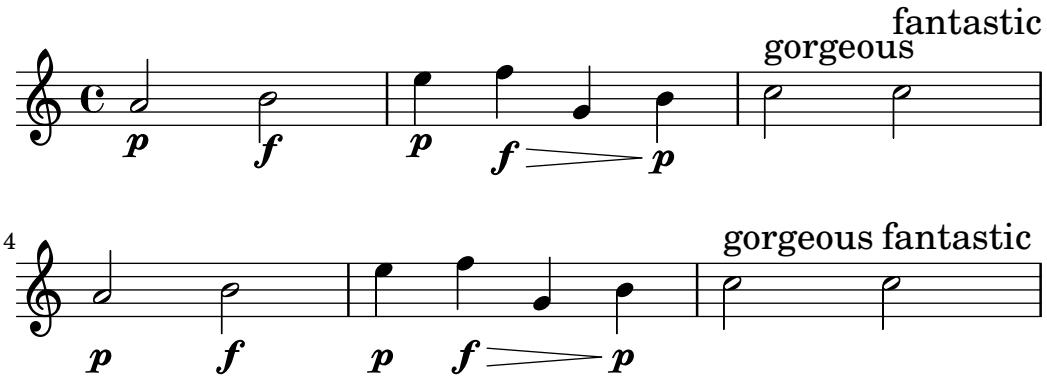
Vertically aligned dynamics and textscripts

For all `DynamicLineSpanner` objects (i.e., hairpins and dynamic texts), the vertical minimum distance between their reference line and the staff is given by the value in the `staff-padding` property, unless other notation elements forces them to be farther away. Setting this property to a sufficiently large value aligns the dynamics.

The same idea, together with `\textLengthOn`, is used to align text scripts along their baseline.

```
music = \relative c' {
  a'2\p b\f
  e4\p f\f\> g, b\p
  c2^\markup { \huge gorgeous } c^\markup { \huge fantastic }
}

{
  \music
  \break
  \override DynamicLineSpanner.staff-padding = 3
  \textLengthOn
  \override TextScript.staff-padding = 1
  \music
}
```



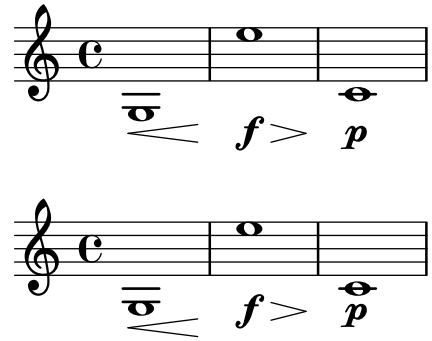
Breaking vertical alignment of dynamics and textscripts

By default, LilyPond uses `DynamicLineSpanner` grobs to vertically align successive dynamic objects like hairpins and dynamic text. However, this is not always wanted. By inserting `\breakDynamicSpan`, which ends the alignment spanner prematurely, this vertical alignment can be avoided.

See also snippet “Breaking horizontal alignment of dynamics and textscripts”.

```
{ g1\< |
  e''\f\> |
  c'\p }

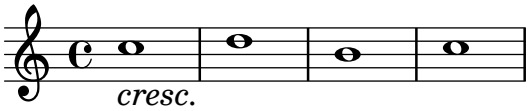
{ g1\< |
  e''\breakDynamicSpan\f\> |
  c'\p }
```



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 as follows.

```
\relative c'' {
  \override DynamicTextSpanner.style = #'none
  \crescTextCresc
  c1\< | d | b | c\!
}
```
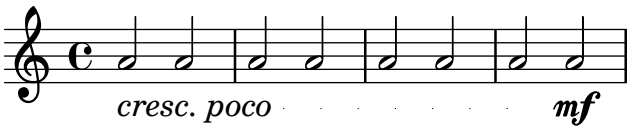


Changing text and spanner styles for text dynamics

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 `dashed-line`, and other possible values include `line`, `dotted-line`, and `none`.

```
\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 “Articulations and dynamics” in *Learning Manual*.

Notation Reference: Section 36.1 [Direction and placement], page 774, Section 3.1.3 [New dynamic marks], page 168, Section 24.9 [Enhancing MIDI output], page 665, Section 24.4 [Controlling MIDI dynamics], page 656.

Snippets: Section “Expressive marks” in *Snippets*.
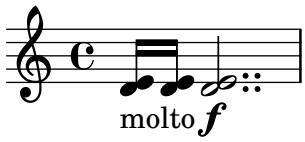
Internals Reference: Section “DynamicText” in *Internals Reference*, Section “Hairpin” in *Internals Reference*, Section “DynamicLineSpanner” in *Internals Reference*, Section “Dynamics” in *Internals Reference*.

3.1.3 New dynamic marks

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

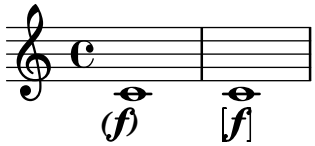
```
moltoF = \markup { molto \dynamic f }

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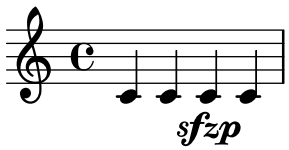


In markup mode, editorial dynamics (within parentheses or square brackets) can be created. The syntax for markup mode is described in Section 8.2 [Formatting text], page 320.

```
roundF = \markup {
    \center-align \concat { \bold { \italic ( }
          \dynamic f \bold { \italic ) } } }
boxF = \markup { \bracket { \dynamic f } }
\relative {
  c'1_\roundF
  c1_\boxF
}
```



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

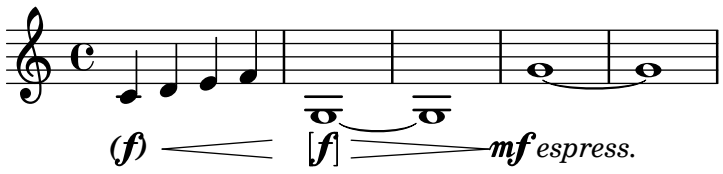
```
sfzp = #(make-dynamic-script "sfzp")
\relative {
  c'4 c c\sfzp c
}
```



In general, `make-dynamic-script` takes any markup object as its argument. The dynamic font only contains the characters `f`, `m`, `p`, `r`, `s`, `z`, and `n`; 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`. Using `make-dynamic-script` instead of an ordinary markup ensures vertical alignment of markup objects and hairpins that are attached to the same note head.

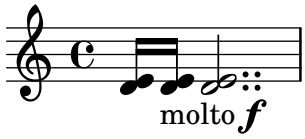
```
roundF = \markup { \center-align \concat {
           \normal-text { \bold { \italic ( } }
           \dynamic f
           \normal-text { \bold { \italic ) } } } }
boxF = \markup { \bracket { \dynamic f } }
mfEspress = \markup { \center-align \line {
              \hspace #3.7 mf \normal-text \italic espress. } }
roundFdynamic = #(make-dynamic-script roundF)
```

```
boxFdynamic = #(make-dynamic-script boxF)
mfEspressDynamic = #(make-dynamic-script mfEspress)
\relative {
  c'4_\roundFdynamic\< d e f
  g,1~_\boxFdynamic\>
  g1
  g'1~\mfEspressDynamic
  g1
}
```
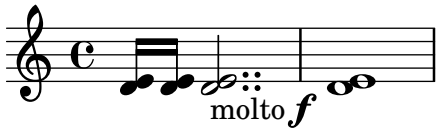


The Scheme form of markup mode may be used instead. Its syntax is explained in Section "Markup construction in Scheme" in *Extending*.

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



To left-align the dynamic text rather than centering it on a note use a `\tweak`:

```
moltoF = \tweak DynamicText.self-alignment-X #LEFT
         #(make-dynamic-script
            (markup #:normal-text "molto"
                    #:dynamic "f"))
\relative {
  <d' e>16 <d e>
  <d e>2..\moltoF <d e>1
}
```



Font settings in markup mode are described in Section 8.2.2 [Selecting font and font size], page 324.

See also

Notation Reference: Section 8.2 [Formatting text], page 320, Section 8.2.2 [Selecting font and font size], page 324, Section 24.9 [Enhancing MIDI output], page 665, Section 24.4 [Controlling MIDI dynamics], page 656.

Extending LilyPond: Section "Markup construction in Scheme" in *Extending*.

Snippets: Section "Expressive marks" in *Snippets*.

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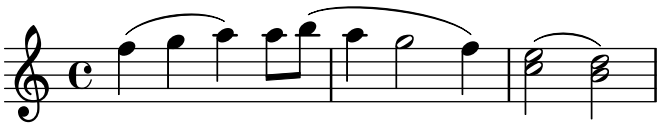
This section explains how to create various expressive marks that are curved: normal slurs, phrasing slurs, breath marks, falls, and doits.

3.2.1 Slurs

Slurs are entered using parentheses:

> **Note:** In polyphonic music, a slur must be terminated in the same voice it began.
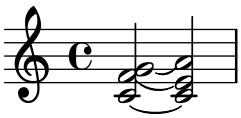
```
\relative {
  f''4( g a) a8 b(
  a4 g2 f4)
  <c e>2( <b d>2)
}
```



Slurs may be manually placed above or below the staff; see Section 36.1 [Direction and placement], page 774.
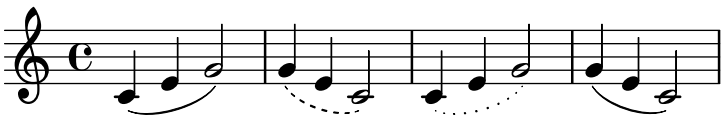
Simultaneous or overlapping slurs require special attention. Most occurrences of outer slurs actually indicate phrasing, and phrasing slurs may overlap a regular slur, see Section 3.2.2 [Phrasing slurs], page 174. When multiple regular slurs are needed in a single `Voice`, matching slur starts and ends need to be labeled by preceding them with `\=` followed by an identifying key (a symbol or non-negative integer).

```
\fixed c' {
  <c~ f\=1( g\=2( >2 <c e\=1) a\=2) >
}
```
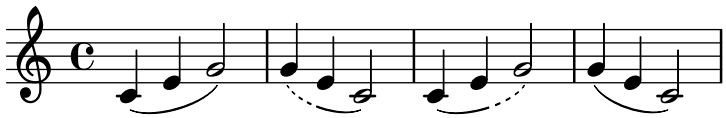


Slurs can be solid, dotted, or dashed. Solid is the default slur style:

```
\relative {
  c'4( e g2)
  \slurDashed
  g4( e c2)
  \slurDotted
  c4( e g2)
  \slurSolid
  g4( e c2)
}
```
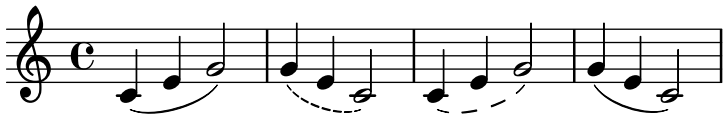


Slurs can also be made half-dashed (the first half dashed, the second half solid) or half-solid (the first half solid, the second half dashed):

```
\relative {
  c'4( e g2)
  \slurHalfDashed
  g4( e c2)
  \slurHalfSolid
  c4( e g2)
  \slurSolid
  g4( e c2)
}
```



Custom dash patterns for slurs can be defined:

```
\relative {
  c'4( 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

Adjusting slur positions vertically

Using \override Slur.positions it is possible to set the vertical position of the start and end points of a slur to absolute values (or rather, forcing LilyPond's slur algorithm to consider these values as desired). In many cases, this means a lot of trial and error until good values are found. You probably have tried the \offset command next just to find out that it doesn't work for slurs, emitting a warning instead.

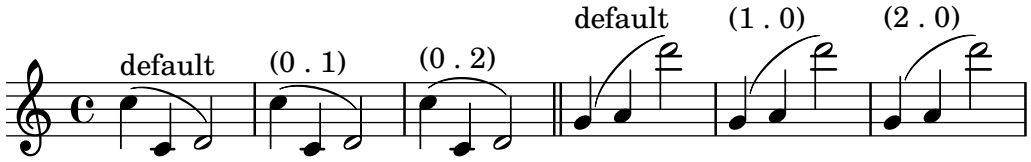
The code in this snippet allows you to tweak the vertical start and end positions by specifying *relative* changes, similar to \offset.

Syntax: \offsetPositions #'(*dy1 . dy2*)

```
offsetPositions =
#(define-music-function (offsets) (number-pair?)
  #{
     \once \override Slur.control-points =
       #(lambda (grob)
          (match-let ((((_ . y1) _ _ (_ . y2))
                       (ly:slur::calc-control-points grob))
```

```
                    ((off1 . off2) offsets))
              (set! (ly:grob-property grob 'positions)
                    (cons (+ y1 off1) (+ y2 off2)))
              (ly:slur::calc-control-points grob)))
  #})

\relative c'' {
  c4(^"default" c, d2)
  \offsetPositions #'(0 . 1)
  c'4(^"(0 . 1)" c, d2)
  \offsetPositions #'(0 . 2)
  c'4(^"(0 . 2)" c, d2)
  \bar "||"
  g4(^"default" a d'2)
  \offsetPositions #'(1 . 0)
  g,,4(^"(1 . 0)" a d'2)
  \offsetPositions #'(2 . 0)
  g,,4(^"(2 . 0)" a d'2)
}
```



Using double slurs for legato chords

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

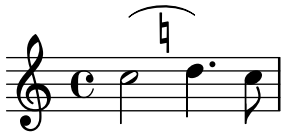
```
\relative c' {
  \set doubleSlurs = ##t
  <c e>4( <d f> <c e> <d f>)
}
```



Positioning text markups inside slurs

Text markups need to have the `outside-staff-priority` property set to `#f` 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
}
```
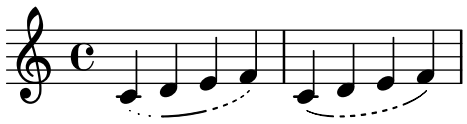


Making slurs with complex dash structure

Slurs can be composed of complex dash patterns by setting the `dash-definition` property, which is a list of slur segments, which in turn are lists of parameters setting up the dash behavior of the given segment.

Slur segments are defined in terms of the Bézier parameter *t*, which ranges from 0 at the left end of the slur to 1 at the right end of the slur. A slur segment has the form `(start-t stop-t dash-fraction dash-period)`. In the segment spanning the range *start-t* to *stop-t*, the dash pattern is defined by the values of *dash-fraction* and *dash-period*. *dash-fraction* specifies how much of a dash period is black; if set to 1 you get a solid slur segment. The unit for *dash-period* is staff spaces.

```
\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 36.1 [Direction and placement], page 774, Section 3.2.2 [Phrasing slurs], page 174.

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

Internals Reference: Section “Slur” in *Internals Reference*.

3.2.2 Phrasing slurs

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

```
\relative {
  c''4\( d( e) f(
  e2) d\)
}
```
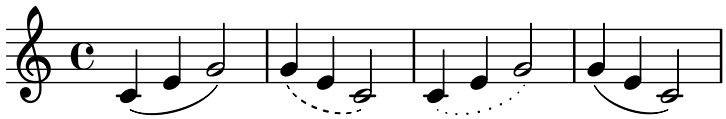


Typographically, a phrasing slur behaves almost exactly like a normal slur. However, they are treated as different objects; a `\slurUp` will have no effect on a phrasing slur. Phrasing

may be manually placed above or below the staff; see Section 36.1 [Direction and placement], page 774.

Simultaneous or overlapping phrasing slurs are entered using \= as with regular slurs, see Section 3.2.1 [Slurs], page 171.
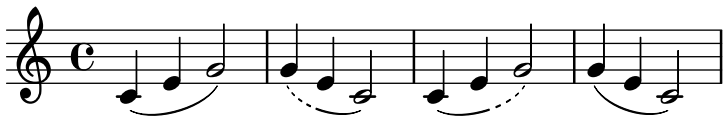
Phrasing slurs can be solid, dotted, or dashed. Solid is the default style for phrasing slurs:

```
\relative {
  c'4\( e g2\)
  \phrasingSlurDashed
  g4\( e c2\)
  \phrasingSlurDotted
  c4\( e g2\)
  \phrasingSlurSolid
  g4\( e c2\)
}
```
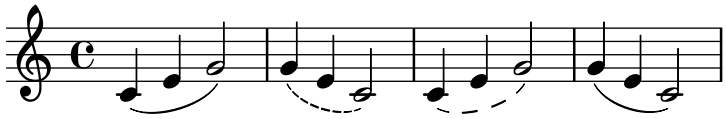


Phrasing slurs can also be made half-dashed (the first half dashed, the second half solid) or half-solid (the first half solid, the second half dashed):

```
\relative {
  c'4\( e g2\)
  \phrasingSlurHalfDashed
  g4\( e c2\)
  \phrasingSlurHalfSolid
  c4\( e g2\)
  \phrasingSlurSolid
  g4\( e c2\)
}
```



Custom dash patterns for phrasing slurs can be defined:

```
\relative {
  c'4\( e g2\)
  \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 Section 3.2.1 [Slurs], page 171.

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 36.1 [Direction and placement], page 774, Section 3.2.1 [Slurs], page 171.
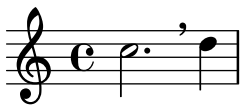
Snippets: Section "Expressive marks" in *Snippets*.

Internals Reference: Section "PhrasingSlur" in *Internals Reference*.

3.2.3 Breath marks

The `\breathe` command calls for the performer to shorten the previous note to take a breath.

```
\fixed c'' { c2. \breathe d4 }
```
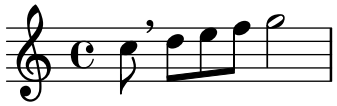


For a short break in sound that is not taken away from the previous note, see Section 2.2.4 [Caesuras], page 76.

Unlike other expressive marks, a breath mark is treated as a separate music event; therefore, any expressive marks pertaining to the preceding note, and any brackets indicating manual beams, slurs, or phrasing slurs, must be placed before `\breathe`. `\breathe` does not accept articulations itself, but see Section 2.2.4 [Caesuras], page 76.
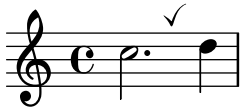
A breath mark ends an automatic beam; to override this, see Section 2.4.3 [Manual beams], page 115.

```
\fixed c'' { c8 \breathe d e f g2 }
```



The `breathMarkType` context property controls which of several predefined breath marks the `\breathe` command creates. See Section B.14 [List of breath marks], page 931.

```
\fixed c'' {
  \set breathMarkType = #'tickmark
  c2. \breathe d4
}
```



See also

Music Glossary: Section "breath mark" in *Music Glossary*.

Notation Reference: Section 2.2.4 [Caesuras], page 76, Section 17.4.4 [Divisiones], page 556.

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*.

3.2.4 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.

```
\relative c'' {
  c2\bendAfter 4
  c2\bendAfter -4
  c2\bendAfter 6.5
  c2\bendAfter -6.5
  c2\bendAfter 8
  c2\bendAfter -8
}
```



Selected snippets

Adjusting the shape of falls and doits

The `shortest-duration-space` property may 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*.

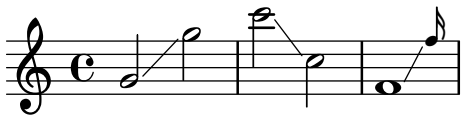
3.3 Expressive marks as lines

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

3.3.1 Glissando

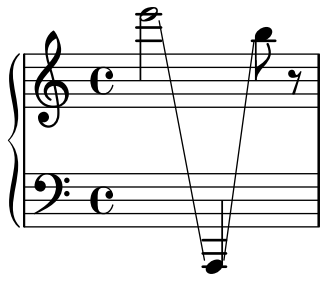
A *glissando* is created by appending `\glissando` to a note:

```
\relative {
  g'2\glissando g'
  c2\glissando c,
  \afterGrace f,1\glissando f'16
}
```
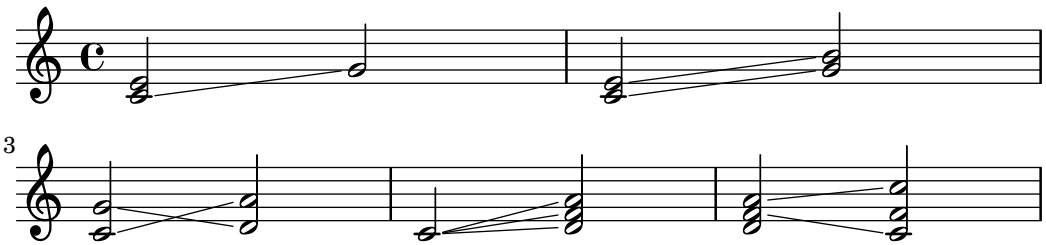


A glissando can connect notes across staves:

```
\new PianoStaff <<
  \new Staff = "right" {
    e'''2\glissando
    \change Staff = "left"
    a,,4\glissando
    \change Staff = "right"
    b''8 r |
  }
  \new Staff = "left" {
    \clef bass
    s1
  }
>>
```



A glissando can connect notes in chords. If anything other than a direct one-to-one pairing of the notes in the two chords is required, the connections between the notes are defined by setting `\glissandoMap` to a Scheme list. The elements are pairs of integers; each pair *(x . y)* creates a glissando line from the *x*-th note of the first chord to the *y*-th note of the second chord. Notes are numbered from zero in the order in which they appear in the input `.ly` file. Not all notes need be part in a glissando.

```
\relative {
  <c' e>2\glissando g'
  <c, e>\glissando <g' b>
  \break
  \set glissandoMap = #'((0 . 1) (1 . 0))
  <c, g'>\glissando <d a'>
  \set glissandoMap = #'((0 . 0) (0 . 1) (0 . 2))
  c\glissando <d f a>
  \set glissandoMap = #'((2 . 2) (0 . 0))
  <f d a'>\glissando <c f c'>
}
```
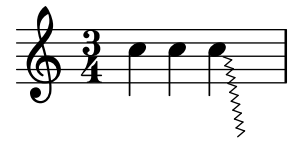


Different styles of glissandi can be created. For details, see Section 36.5 [Line styles], page 782.

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 "|"
}
```



Adding timing marks to long glissandi

Skipped beats in very long glissandi are sometimes indicated by timing marks consisting of stems without noteheads. Such stems can also be used to carry intermediate expression markings.
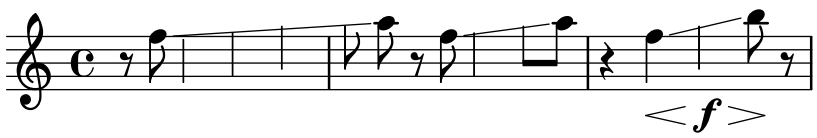
If the stems do not align well with the glissando, they may need to be repositioned slightly.

```
glissandoSkipOn = {
  \override NoteColumn.glissando-skip = ##t
  \hide NoteHead
  \override NoteHead.no-ledgers = ##t
}

glissandoSkipOff = {
  \revert NoteColumn.glissando-skip
  \undo \hide NoteHead
  \revert NoteHead.no-ledgers
}

\relative c'' {
  r8 f8\glissando \glissandoSkipOn f4 g a |
```

```
  a8\noBeam \glissandoSkipOff a8
    r8 f8\glissando \glissandoSkipOn g4 a8 \glissandoSkipOff a8 |
  r4 f\glissando\< \glissandoSkipOn a4\f\> \glissandoSkipOff b8\! r |
}
```



Making glissandi breakable

Normally, LilyPond refuses to automatically break a line at places where a glissando crosses a bar line. This behavior can be changed by setting the `Glissando.breakable` property to `#t`. Also setting the `after-line-breaking` property to `#t` makes the glissando line continue after the break.

The `breakable` property does not affect manual breaks inserted with commands like `\break`.
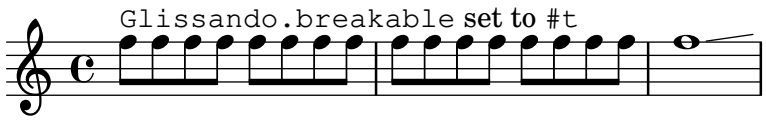
```
glissandoSkipOn = {
  \override NoteColumn.glissando-skip = ##t
  \hide NoteHead
  \override NoteHead.no-ledgers = ##t
}
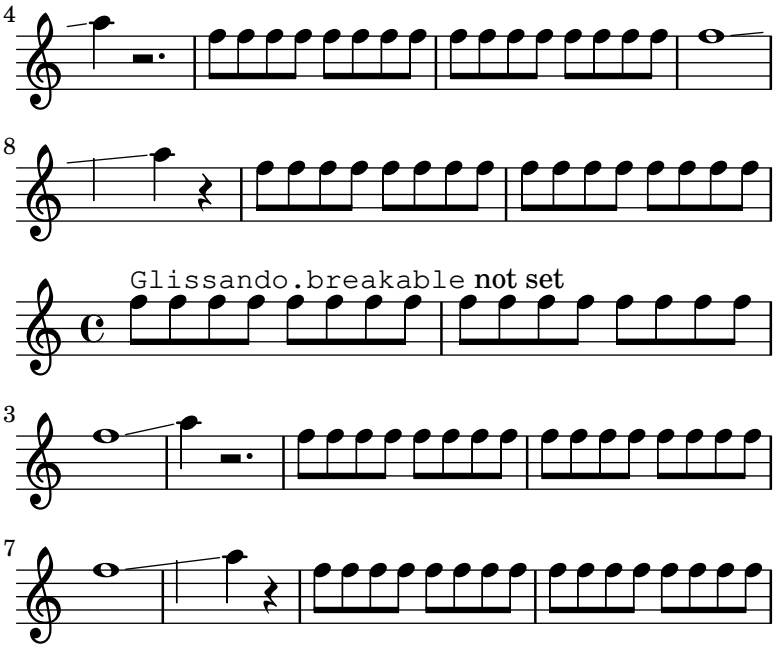
music = {
  \*16 f8 |
  f1\glissando |
  a4 r2. |
  \*16 f8 |
  f1\glissando \once\glissandoSkipOn |
  a2 a4 r4 |
  \*16 f8
}

\relative c'' {
  <>^\markup { \typewriter Glissando.breakable
               set to \typewriter "#t" }
  \override Glissando.breakable = ##t
  \override Glissando.after-line-breaking = ##t
  \music
}

\relative c'' {
  <>^\markup { \typewriter Glissando.breakable not set }
  \music
}

\paper {
  line-width = 100\mm
}
```





Extending glissandi across repeats

A glissando that extends into several `\alternative` blocks can be simulated by adding a hidden grace note with a glissando at the start of each `\alternative` block. The grace note should be at the same pitch as the note which starts the initial glissando. This is implemented here with a music function that takes the pitch of the grace note as its argument.

Note that in polyphonic music the grace note must be matched with corresponding grace notes in all other voices.

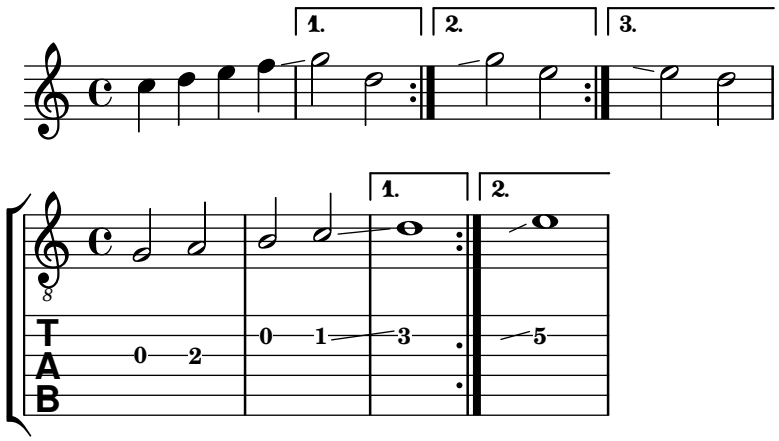
```
repeatGliss = #(define-music-function (grace)
  (ly:pitch?)
  #{
    % the next two lines ensure the glissando is long enough
    % to be visible
    \once \override Glissando.springs-and-rods
      = #ly:spanner::set-spacing-rods
    \once \override Glissando.minimum-length = 3.5
    \once \hideNotes
    \grace $grace \glissando
  #})

\score {
  \relative c'' {
    \repeat volta 3 { c4 d e f\glissando }
    \alternative {
      { g2 d }
      { \repeatGliss f g2 e }
      { \repeatGliss f e2 d }
    }
  }
}

music =  \relative c' {
  \voiceOne
```

```
  \repeat volta 2 {
    g a b c\glissando
  }
  \alternative {
    { d1 }
    { \repeatGliss c \once \omit StringNumber e1\2 }
  }
}

\score {
  \new StaffGroup <<
    \new Staff <<
      \new Voice { \clef "G_8" \music }
    >>
    \new TabStaff  <<
      \new TabVoice { \clef "moderntab" \music }
    >>
  >>
}
```



See also

Music Glossary: Section “glissando” in *Music Glossary*.

Notation Reference: Section 36.5 [Line styles], page 782.

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

Internals Reference: Section “Glissando” in *Internals Reference*.
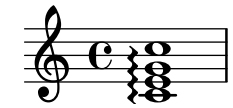
Known issues and warnings

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

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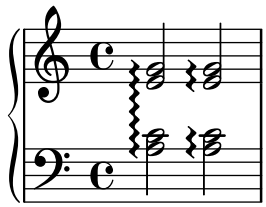
An *arpeggio* on a chord (also known as a broken chord) is denoted by appending `\arpeggio` to the chord construct:

```
\relative { <c' e g c>1\arpeggio }
```



It is possible to connect arpeggios across voices or staves by setting the property `connectArpeggios` in a staff-grouping context. This connects all arpeggios in that context:

```
\new PianoStaff \fixed c' <<
  \new Staff {
    \once \set PianoStaff.connectArpeggios = ##t
    <e g>2\arpeggio q\arpeggio
  }
  \new Staff {
    \clef "bass"
    <a, c>2\arpeggio q\arpeggio
  }
>>
```



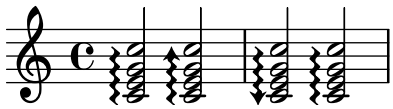
There are commands to specify the performance direction. `\arpeggioNormal` reverts to a normal arpeggio:

```
\relative {
  <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
}
```



These predefined commands internally modify the `arpeggio-direction` property; see their full definition in the `ly/property-init.ly` file.

The symbol that the `\arpeggio` command creates can be changed:

```
\relative {
  <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
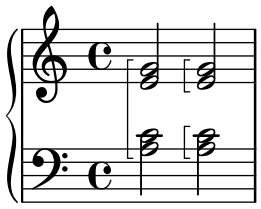
  \arpeggioNormal
  <c e g c>2\arpeggio
}
```



These predefined commands internally override the `Arpeggio` object's `stencil` property, and may also adapt its `X-extent` (that is, the horizontal dimension it takes not to collide with other objects).

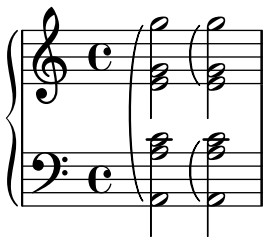
For a bracket designating a non-arpeggiated chord, it is better to use `\nonArpeggiato` than to use `\arpeggio` and alter the appearance. This creates a `ChordBracket` object.

```
\new PianoStaff \fixed c' <<
  \new Staff {
    \once \set PianoStaff.connectChordBrackets = ##t
    <e g>2\nonArpeggiato q\nonArpeggiato
  }
  \new Staff {
    \clef "bass"
    <a, c>2\nonArpeggiato q\nonArpeggiato
  }
>>
```



For a vertical slur designating a quasi-non-arpeggiated chord, it is better to use `\chordSlur` than to use `\arpeggio` and alter the appearance. This creates a `ChordSlur` object.

```
\new PianoStaff \fixed c <<
  \new Staff {
    \once \set PianoStaff.connectChordSlurs = ##t
    <e' g' g''>2\chordSlur q\chordSlur
  }
  \new Staff {
    \clef "bass"
    <a, a c'>2\chordSlur q\chordSlur
  }
>>
```



The dash properties of ChordSlur and the parenthesis Arpeggio are controlled with the dash-definition property (see Section 3.2.1 [Slurs], page 171).

Arpeggios can be explicitly written out with ties. For more information, see Section 2.1.4 [Ties], page 63.

Predefined commands
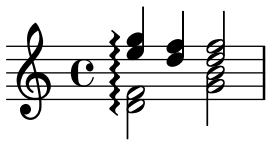
\arpeggio, \arpeggioArrowUp, \arpeggioArrowDown, \arpeggioNormal, \arpeggioBracket, \arpeggioParenthesis, \arpeggioParenthesisDashed, \chordSlur, \nonArpeggiato.

Selected snippets

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: Section 3.2.1 [Slurs], page 171, Section 2.1.4 [Ties], page 63.

Installed Files: ly/property-init.ly.

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

Internals Reference: Section “Arpeggio” in *Internals Reference*, Section “ChordBracket” in *Internals Reference*, Section “ChordSlur” in *Internals Reference*, Section “Slur” in *Internals Reference*, Section “PianoStaff” in *Internals Reference*.

Known issues and warnings

Predefined commands such as \arpeggioArrowUp only apply to the current context, and thus will not affect arpeggios spanning several voices or staves. In such cases, these commands need to be used in a \context block within \layout, or in a \with block, as explained in Section 33.5 [Changing context default settings], page 746. Alternatively, rather than using predefined shortcuts, it may be advisable to directly override the relevant properties for the Arpeggio object in the appropriate context; for example:

```
\override Staff.Arpeggio.stencil = #ly:chord-bracket::print
\override Staff.Arpeggio.X-extent = #ly:chord-bracket::width
```

to print cross-voice arpeggio brackets at the Staff level, or

```
\override PianoStaff.Arpeggio.arpeggio-direction = #UP
```

to print cross-staff arrowed arpeggios (pointing upwards) in a PianoStaff context.

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

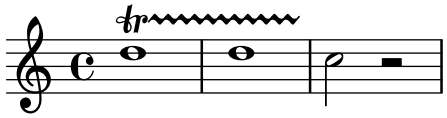
The simple way of setting parenthesis-style arpeggio brackets does not work for cross-staff arpeggios; see [Cross-staff chord brackets], page 420.

3.3.3 Trills

Short trills without an extender line are printed with \trill; see Section 3.1.1 [Articulations and ornamentations], page 156.
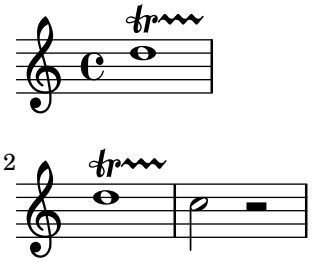
Longer trills with an extender line are made with \startTrillSpan and \stopTrillSpan:

```
\relative {
  d''1\startTrillSpan
  d1
  c2\stopTrillSpan
  r2
}
```



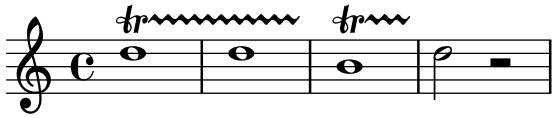
A trill spanner crossing a line break will restart exactly above the first note on the new line.

```
\relative {
  d''1\startTrillSpan
  \break
  d1
  c2\stopTrillSpan
  r2
}
```
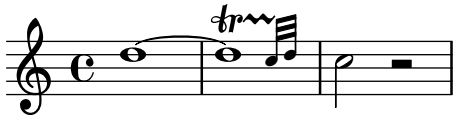


Consecutive trill spans will work without explicit \stopTrillSpan commands, since successive trill spanners will automatically become the right bound of the previous trill.

```
\relative {
  d''1\startTrillSpan
  d1
  b1\startTrillSpan
  d2\stopTrillSpan
  r2
}
```
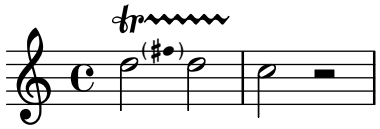


Trills can also be combined with grace notes. The syntax of this construct and the method to precisely position the grace notes are described in Section 2.6.1 [Grace notes], page 148.

```
\relative {
  d''1~\afterGrace
  d1\startTrillSpan { c32[ d]\stopTrillSpan }
  c2 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.

```
\relative {
  \pitchedTrill
  d''2\startTrillSpan fis
  d2
  c2\stopTrillSpan
  r2
}
```



The Accidental of the first pitched trill in a measure is always printed, even for naturals.

```
{
  \key d \major
  \pitchedTrill
  d'2\startTrillSpan cis d\stopTrillSpan
  \pitchedTrill
  d2\startTrillSpan c d\stopTrillSpan
  \pitchedTrill
  d2\startTrillSpan e d\stopTrillSpan
}
```



Subsequent accidentals (of the same note in the same measure) will need to be added manually.

```
\relative {
  \pitchedTrill
  eis''4\startTrillSpan fis
  eis4\stopTrillSpan
  \pitchedTrill
  eis4\startTrillSpan cis
  eis4\stopTrillSpan
  \pitchedTrill
  eis4\startTrillSpan fis
  eis4\stopTrillSpan
  \pitchedTrill
  eis4\startTrillSpan fis!
  eis4\stopTrillSpan
}
```



Predefined commands

`\startTrillSpan`, `\stopTrillSpan`.

See also
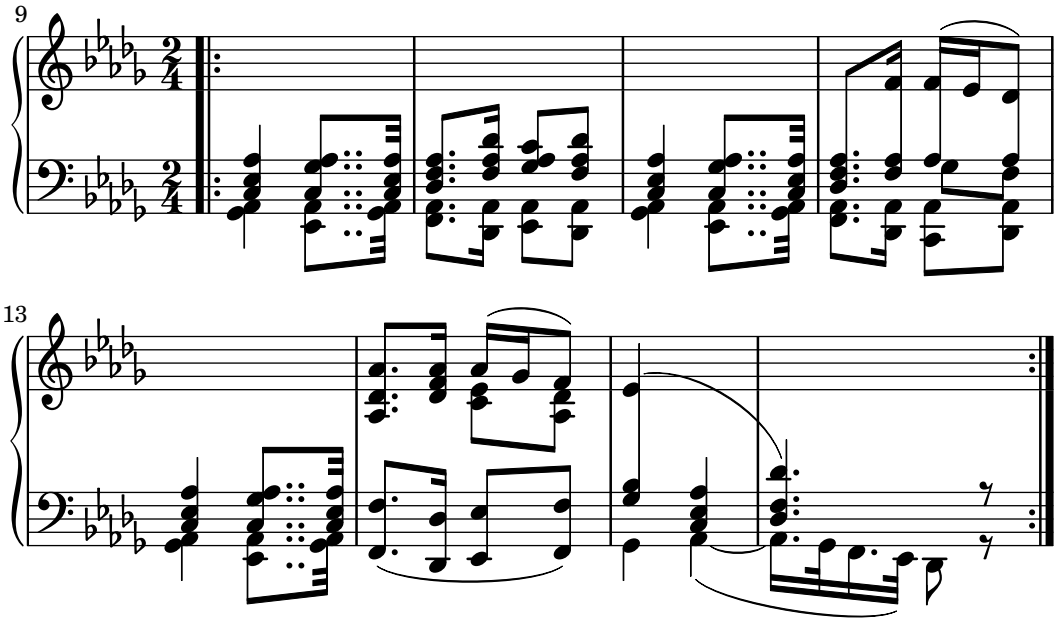
Music Glossary: Section “trill” in *Music Glossary*.

Notation Reference: Section 3.1.1 [Articulations and ornamentations], page 156, Section 2.6.1 [Grace notes], page 148.

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

Internals Reference: Section “TrillSpanner” in *Internals Reference*, Section “TrillPitchHead” in *Internals Reference*, Section “TrillPitchAccidental” in *Internals Reference*, Section “TrillPitchParentheses” in *Internals Reference*, Section “TrillPitchGroup” in *Internals Reference*, Section “Pitched_trill_engraver” in *Internals Reference*.

4 Repeats



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

`volta` This is the standard notation for repeats with or without alternative endings. The repeated section is framed between repeat bar lines, but the starting bar line is omitted when the repeated section begins the piece. Alternative endings are printed in sequence, bracketed, and numbered with the volte to which they apply.

`segno` This supports various *da capo* and *dal segno* cases. The repeated section begins with a segno mark, except at the start of the piece. Alternative endings are printed in sequence and marked with coda marks, and a section label such as 'Coda' can optionally be applied to the final alternative. The repeated section ends with an instruction such as *D.S.*

`unfold` The repeated music is written out in full a specified number of times.

`percent` These are beat or measure repeats. They look like single slashes or percent signs.

`tremolo` This is used to write tremolo beams.

Chord constructs can be repeated using the chord repetition symbol, `q`. See Section 5.1.2 [Chord repetition], page 218.

4.1 Long repeats
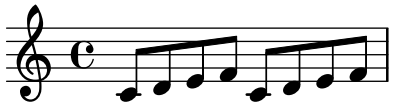
This section discusses how to input long (usually multi-measure) repeats.

4.1.1 Written-out repeats

The `\repeat unfold` command repeats music by writing it out a number of times. The syntax is the same as the `\repeat volta` and `\repeat segno` commands, which are documented in following sections.

There is also a short form, `\*`, which can be convenient in a work that uses unfolded repetition heavily:

```
\*2 { c'8 d' e' f' }
```



To avoid redundancy, unfolding is not demonstrated in detail here; however, some of the examples in following sections illustrate repeats in multiple forms using the `\unfoldRepeats` command to convert the `volta` or `segno` form to the `unfold` form. For another important use of the `\unfoldRepeats` command, see Section 24.6 [Using repeats with MIDI], page 661.

There are some points of interest specific to the `\repeat unfold` command.

In some cases, especially in a `\relative` context, the outcome of unfolding is not the same as of writing the input music expression multiple times, e.g.,

```
\repeat unfold 2 { a'4 b c d | }
```

differs from the following by an octave change:

```
a'4 b c d |
a'4 b c d |
```

Also, nesting `\repeat unfold` can be practical in ways that nesting `\repeat volta` or `\repeat segno` would not be.

> **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 "UnfoldedRepeatedMusic" in *Internals Reference*.
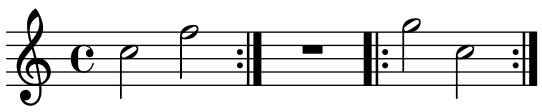
4.1.2 Simple repeats

This is the syntax for a repeat without variation:

```
\repeat volta repeatcount musicexpr
```
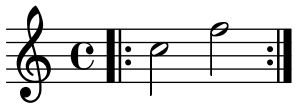
where *musicexpr* is the music expression to be repeated.

```
\fixed c'' {
  \repeat volta 2 { c2 f }
  R1
  \repeat volta 2 { g2 c }
}
```
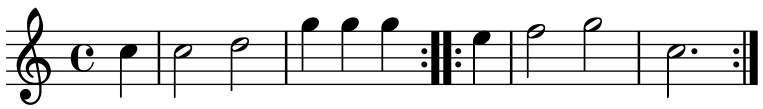


By default, a starting bar line is not automatically printed at the beginning of a piece, in accordance with classical engraving conventions. However, in some contexts, these bar lines are traditionally added, such as in lead sheets for jazz standards. This can be achieved by setting the `printInitialRepeatBar` property.

```
\fixed c'' {
  \set Score.printInitialRepeatBar = ##t
  \repeat volta 2 { c2 f }
}
```



A repeated section that starts in the middle of a measure usually ends at the same position in a later measure so that the two ends make a complete measure. The repeat bar lines are not measure boundaries in such cases, so no bar checks should be placed there. Likewise, no `\partial` command should be placed within the repeated music, because the measures are complete; however, a `\partial` command should be placed before the repeat when there is a truly incomplete measure the first time through.

```
\fixed c'' {
  \partial 4
  \repeat volta 2 {
    c4
    c2 d
    g4 g g
  }
  \repeat volta 2 {
    e4
    f2 g
    c2.
  }
}
```



4.1.3 Alternative endings
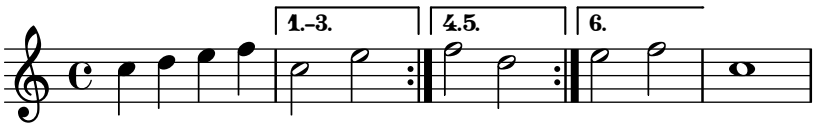
Repeats with alternative endings can be written two ways. This is the preferred syntax:

```
\repeat volta repeatcount {
  musicexpr...
  \alternative {
    \volta numberlist musicexpr
    \volta numberlist musicexpr
    ...
  }
}
```

where *musicexpr* is a music expression, *musicexpr...* is any number of them, and *numberlist* is a comma-separated list of volta numbers chosen from the range 1 to *repeatcount*.

```
\fixed c'' {
  \repeat volta 6 {
    c4 d e f
    \alternative {
      \volta 1,2,3 { c2 e }
      \volta 4,5 { f2 d }
      \volta 6 { e2 f }
```

```
    }
  }
  c1
}
```

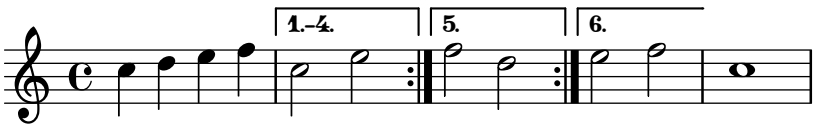


An older syntax where the \alternative block follows outside the repeated music expression is still supported and has the same effect. This syntax requires \repeat to appear literally: a trailing \alternative does not attach to a music function or variable containing \repeat.

```
\repeat volta repeatcount musicexpr
\alternative {
  \volta numberlist musicexpr
  \volta numberlist musicexpr
  ...
}
```

\volta specifications within an \alternative block are optional on an all-or-none basis. If they are omitted, alternatives are used once each, but the first is repeated as needed to satisfy the repeat count.

```
\fixed c'' {
  \repeat volta 6 {
    c4 d e f
    \alternative {
      { c2 e }
      { f2 d }
      { e2 f }
    }
  }
  c1
}
```

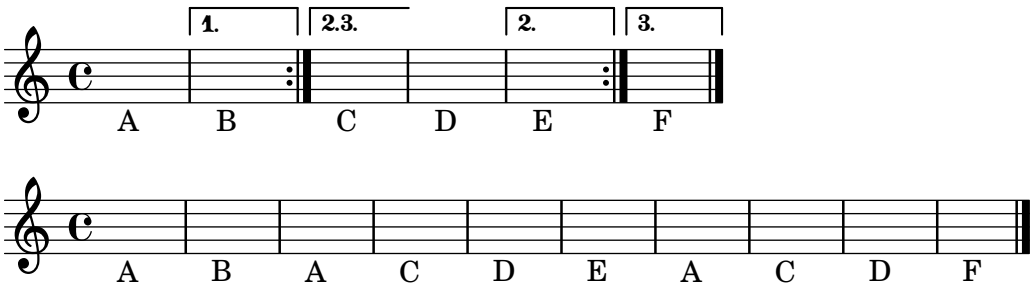


\alternative blocks can be nested.

```
music = \fixed c' {
  \repeat volta 3 {
    s1_"A"
    \alternative {
      \volta 1 { s1_"B" }
      \volta 2,3 {
        \once \override Score.VoltaBracket.musical-length =
          \musicLength 1
        s1_"C"
        s1_"D"
        \alternative {
          \volta 2 { s1_"E" }
          \volta 3 { s1_"F" }
```

```
        }
      }
    }
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```



Note: Every element in an `\alternative` block is treated as an alternative ending. Something as simple as a bar check on the wrong side of a bracket can produce unexpected results.

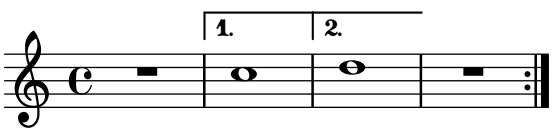
Note: If you include `\relative` inside a `\repeat` without explicitly instantiating the `Voice` context, extra (unwanted) staves will appear. See Section "An extra staff appears" in *Application Usage*.

Note: When alternative bar numbering is enabled, it is applied to the outermost bracketed alternatives.

4.1.4 Other variation in repeated sections

An `\alternative` block can be used within a `\repeat` block to produce notation similar to alternative endings (see Section 4.1.3 [Alternative endings], page 191).

```
\fixed c'' {
  \repeat volta 2 {
    R1
    \alternative {
      \volta 1 { c1 }
      \volta 2 { d1 }
    }
    R1
  }
}
```

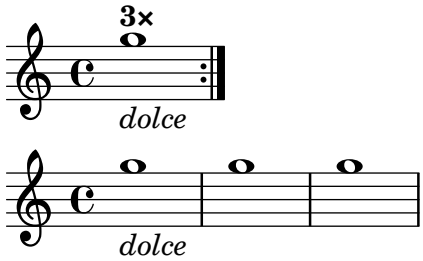


The \volta command is not limited to use on the elements of an \alternative block. It can be used anywhere within a \repeat to designate music for particular volte, though it does not create brackets in other cases.

When a \repeat is unfolded, volta-specific music is omitted from every volta to which it does not apply. Providing an empty Scheme list in place of volta numbers removes the music entirely.

```
music = \repeat volta 3 {
  \volta #'() { <>^\markup { \bold "3×" } }
  \volta 1 { <>_\markup { \italic dolce } }
  g''1
}

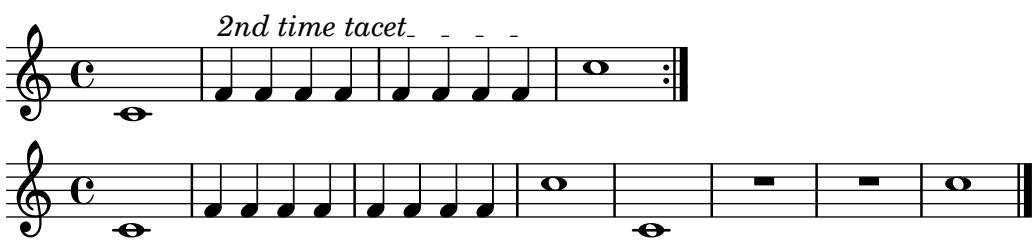
\score { \music }
\score { \unfoldRepeats \music }
```



When a \repeat is unfolded, it may be desirable not only to filter out volta-specific music, but also to add music that was not present in the folded form. The \unfolded command designates music to be ignored until the enclosing \repeat is unfolded.

```
music = \fixed c' {
  \repeat volta 2 {
    c1
    <<
      \volta #'() {
        \once \override TextSpanner.bound-details.left.text =
          "2nd time tacet"
        s4*7\startTextSpan s4\stopTextSpan
      }
      \volta 1 { f4 f f f | f f f f }
      \volta 2 { \unfolded { R1*2 } }
    >>
    c'1
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```



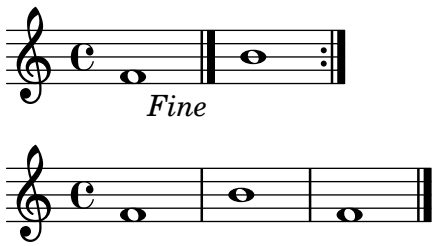
> **Note:** The `\volta` and `\unfolded` commands function with respect to the innermost repeat enclosing them.

4.1.5 Al-fine repeats

The `\fine` command marks the end of the music but does not enforce it. When a repeat containing `\fine` is unfolded, the `\fine` command is unfolded like any other music. For correct unfolding, it is necessary to specify the volta in which the *Fine* should be performed and the volte in which any following music should be performed (see Section 4.1.4 [Other variation in repeated sections], page 193).
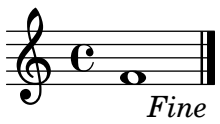
```
music = \fixed c' {
  \repeat volta 2 {
    f1
    \volta 2 \fine
    \volta 1 b1
  }
}

\score { \music }
\score { \unfoldRepeats \music }
```
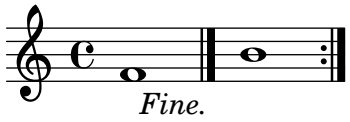


As shown immediately above, at the written end of the music, `\fine` creates a final bar line without a *Fine* instruction. To force *Fine* to appear in such cases, set the `finalFineTextVisibility` context property.

```
\fixed c' {
  \set Score.finalFineTextVisibility = ##t
  f1
  \fine
}
```



To change the text that `\fine` prints, set the `fineText` context property.

```
\fixed c' {
  \set Score.fineText = "Fine."
  \repeat volta 2 {
    f1
    \volta 2 \fine
    \volta 1 b1
  }
}
```



For details on interactions with other types of bar lines and options for changing their appearance, see Section 2.5.2 [Automatic bar lines], page 131.

See also

Music Glossary: Section “fine” in *Music Glossary*.

Notation Reference: Section 2.5.2 [Automatic bar lines], page 131, Section 4.1.4 [Other variation in repeated sections], page 193, Section 2.5.7 [Section divisions], page 147.

Snippets: Section “Repeats” in *Snippets*.

Internals Reference: Section “FineEvent” in *Internals Reference*, Section “Jump_engraver” in *Internals Reference*, Section “JumpScript” in *Internals Reference*.

4.1.6 Segno repeat structure

`\repeat segno` differs from `\repeat volta` only in the resulting notation. Refer to the preceding sections for general information on entering music with repetition, alternatives, and variation. This section covers particulars of segno notation without fully reiterating the input syntax.

`\repeat segno` notates repetition with *D.C.* or *D.S.* instructions. It marks the beginning of the repeated section with a segno mark when it is not the beginning of the piece. It also marks alternative endings with coda marks in lieu of volta brackets, provided that the endings are intended to be performed in order, e.g., not `\volta 1,3` then `\volta 2,4`.

When alternative bar numbering is enabled, it is applied to alternatives notated with volta brackets whether they are created by `\repeat segno` or `\repeat volta`, but it is not applied to alternative endings notated with coda marks.

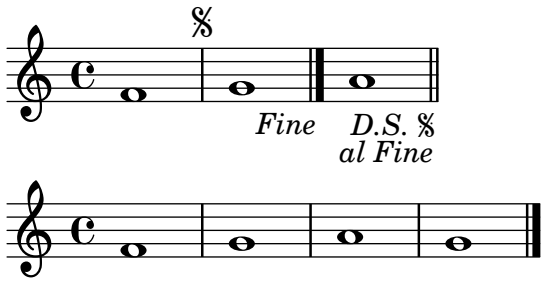
Demonstrations of common uses follow.

al fine

Repeat instructions include *al Fine* if a `\fine` command appears at any prior point (see Section 4.1.5 [Al-fine repeats], page 195).

```
music = \fixed c' {
  f1
  \repeat segno 2 {
    g1
    \volta 2 \fine
    \volta 1 a1
  }
  \section
}

\score { \music }
\score { \unfoldRepeats \music }
```
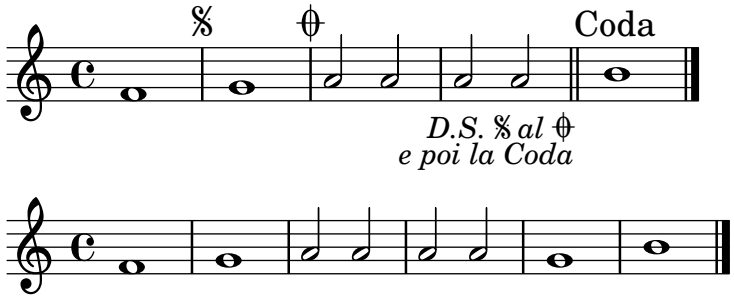


alla coda

The beginning of each alternative ending is marked with an implied `\codaMark \default`. Repeat instructions in alternatives include '*al ... e poi la ...*' referring to the mark at the first alternative and the mark to skip to. Provided that the duration of the final alternative is zero, the automatic mark is suppressed, allowing a section label to be set instead.

```
music = \fixed c' {
  f1
  \repeat segno 2 {
    g1
    \alternative {
      \volta 1 { \*4 a2 }
      \volta 2 \volta #'() {
        \section
        \sectionLabel "Coda"
      }
    }
  }
  b1
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```



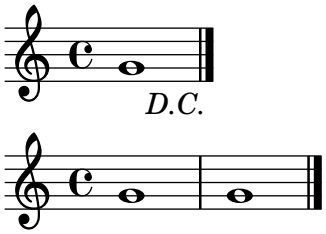
The return instruction can be abbreviated by setting an alternative formatting procedure (see Section 4.1.7 [Segno repeat appearance], page 199).

da capo

Repeat instructions include *D.C.* when the repeated section begins at the beginning of the score. The supported *da capo* cases parallel the supported *dal segno* cases.

```
music = \fixed c' {
  \repeat segno 2 {
    g1
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```

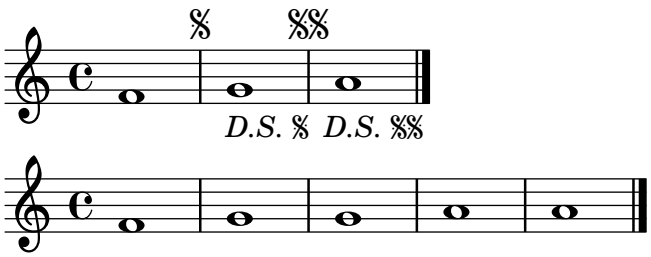


dal segno

Repeat instructions include *D.S.* when the repeated section begins after the beginning of the score. The beginning of the repeated section is marked with an implied `\segnoMark \default`.

```
music = \fixed c' {
  f1
  \repeat segno 2 {
    g1
  }
  \repeat segno 2 {
    a1
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```
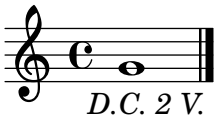


A *dal-segno* repeat starting at the beginning of the score can be forced (see Section 4.1.7 [Segno repeat appearance], page 199).

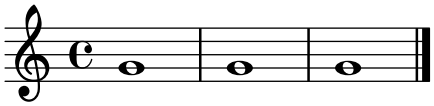
multiple return

A repeat instruction to be performed more than once includes '*. . . V.*'. The number of times the instruction is performed is one less than the number of times the passage is performed.

```
music = \fixed c' {
  \repeat segno 3 {
    g1
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```





See also

Music Glossary: Section “da capo” in *Music Glossary*, Section “dal segno” in *Music Glossary*, Section “fine” in *Music Glossary*.

Notation Reference: Section 4.1.5 [Al-fine repeats], page 195, Section 4.1.3 [Alternative endings], page 191, Section 2.5.2 [Automatic bar lines], page 131, Section 4.1.4 [Other variation in repeated sections], page 193, Section 2.5.7 [Section divisions], page 147, Section 8.1.4 [Section labels], page 313, Section 4.1.2 [Simple repeats], page 190.

Snippets: Section “Repeats” in *Snippets*.
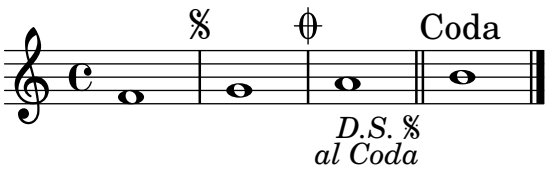
Internals Reference: Section “CodaMark” in *Internals Reference*, Section “JumpScript” in *Internals Reference*, Section “SectionLabel” in *Internals Reference*, Section “SegnoMark” in *Internals Reference*.

4.1.7 Segno repeat appearance

The marks and return instructions that `\repeat segno` creates are adjustable in some respects.
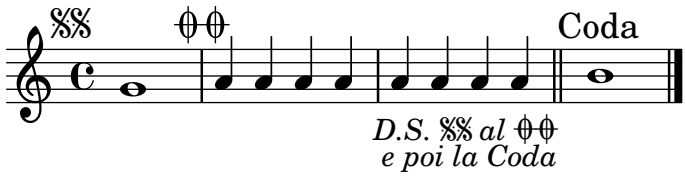
Return instructions are formatted with the Scheme procedure specified in the `dalSegnoTextFormatter` property. There is a predefined alternative formatter that creates shorter instructions.

```
\fixed c' {
  \set Score.dalSegnoTextFormatter = #format-dal-segno-text-brief
  f1
  \repeat segno 2 {
    g1
    \alternative {
      \volta 1 { a1 }
      \volta 2 \volta #'() {
        \section
        \sectionLabel "Coda"
      }
    }
  }
  b1
  \fine
}
```



The sequence numbers of the marks at the beginning of the repeated section and the beginning of the first alternative may be set explicitly without interfering with automatic return instructions (see Section 4.1.8 [Manual repeat marks], page 205).

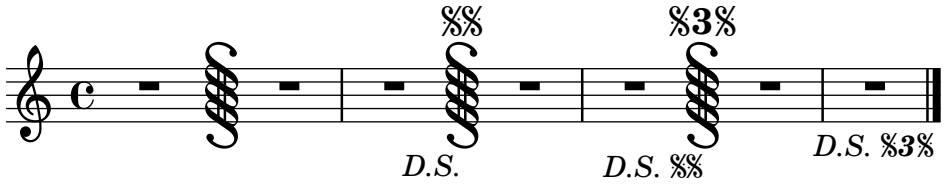
```
\fixed c' {
  \repeat segno 2 {
    \volta #'() { \segnoMark 2 }
    g1
    \alternative {
      \volta 1 {
        \volta #'() { \codaMark 2 }
        \*8 a4
      }
      \volta 2 \volta #'() {
        \section
        \sectionLabel "Coda"
      }
    }
  }
  b1
  \fine
}
```



Without the explicit `\segnoMark 2`, the above would have been rendered as a *da-capo* repeat.

As an alternative to printing a segno as a mark above the staff, it is possible to print it as a bar line by setting the `segnoStyle` property to `bar-line`. To avoid ambiguity, only the first segno bar remains unmarked.

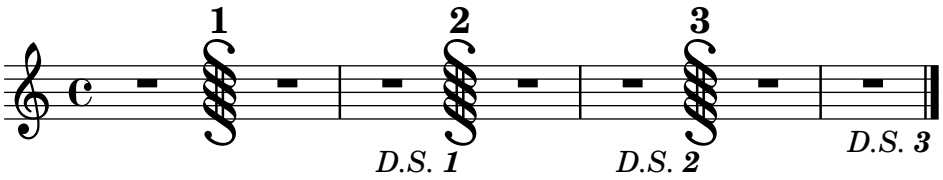
```
\fixed c' {
  \set Score.segnoStyle = #'bar-line
  R1
  \repeat unfold 3 {
    \repeat segno 2 {
      R1*2
    }
  }
  \fine
}
```



Where a segno bar coincides with other special bar lines, a combination bar line is chosen automatically from a predetermined set. For each supported combination, the bar line can be customized by setting a context property (see Section 2.5.2 [Automatic bar lines], page 131).
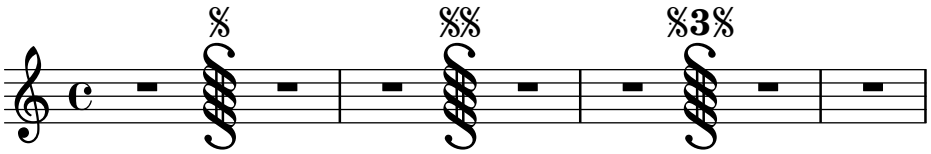
Segno and coda marks are formatted with procedures specified in the `segnoMarkFormatter` and `codaMarkFormatter` properties. These are interchangeable with procedures used to format rehearsal marks (see Section 2.5.5 [Rehearsal marks], page 142).

```
\fixed c' {
  \set Score.segnoMarkFormatter = #format-mark-numbers
  \set Score.segnoStyle = #'bar-line
  R1
  \repeat unfold 3 {
    \repeat segno 2 {
      R1*2
    }
  }
  \fine
}
```



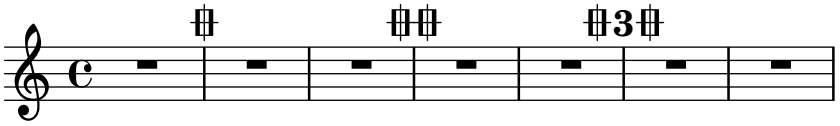
There is a predefined alternative segno formatter that prints a mark even over the first bar line:

```
\fixed c' {
  \set Score.segnoMarkFormatter = #format-segno-mark
  \set Score.segnoStyle = #'bar-line
  R1
  \repeat unfold 3 {
    \segnoMark \default
    R1*2
  }
}
```



There is a predefined alternative coda mark formatter that uses \varcoda signs.

```
\fixed c' {
  \set Score.codaMarkFormatter = #format-varcoda-mark
  R1
  \repeat unfold 3 {
    \codaMark \default
    R1*2
  }
}
```
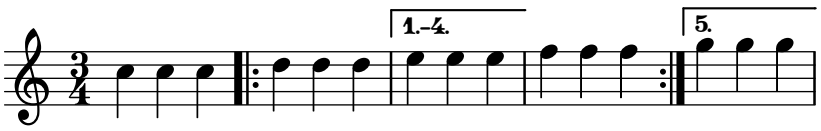


Selected snippets

Shortening volta brackets

By default, volta brackets are drawn over all of the alternative music, but it is possible to shorten them by overriding `VoltaBracket.musical-length`. In the next example, the bracket only spans one measure, which has a duration of 3/4.

```
\fixed c'' {
  \time 3/4
  c4 c c
  \repeat volta 5 {
    d4 d d
    \alternative {
      \volta 1,2,3,4 {
        \once \override Score.VoltaBracket.musical-length =
          \musicLength 2.
        e4 e e
        f4 f f
      }
      \volta 5 {
        g4 g g } } }
}
```



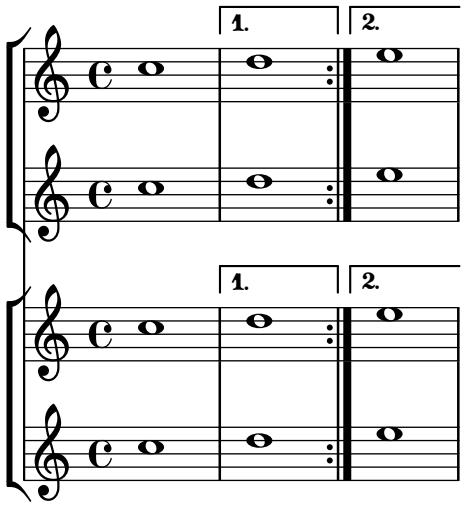
Volta brackets in multiple staves

By adding the `Volta_engraver` to the relevant staff, volte can be put over staves other than the topmost one in a score.

`\repeat` and related commands should be present in all staves.

```
voltaMusic = \relative c'' {
  \repeat volta 2 {
    c1
    \alternative {
      \volta 1 { d1 }
      \volta 2 { e1 }
    }
  }
}

<<
  \new StaffGroup <<
    \new Staff \voltaMusic
    \new Staff \voltaMusic
  >>
  \new StaffGroup <<
    \new Staff \with { \consists "Volta_engraver" }
      \voltaMusic
    \new Staff \voltaMusic
  >>
>>
```



Alternative bar numbering

Setting the `alternativeNumberingStyle` context property, two additional methods are available for enumerating bar numbers in repeats.
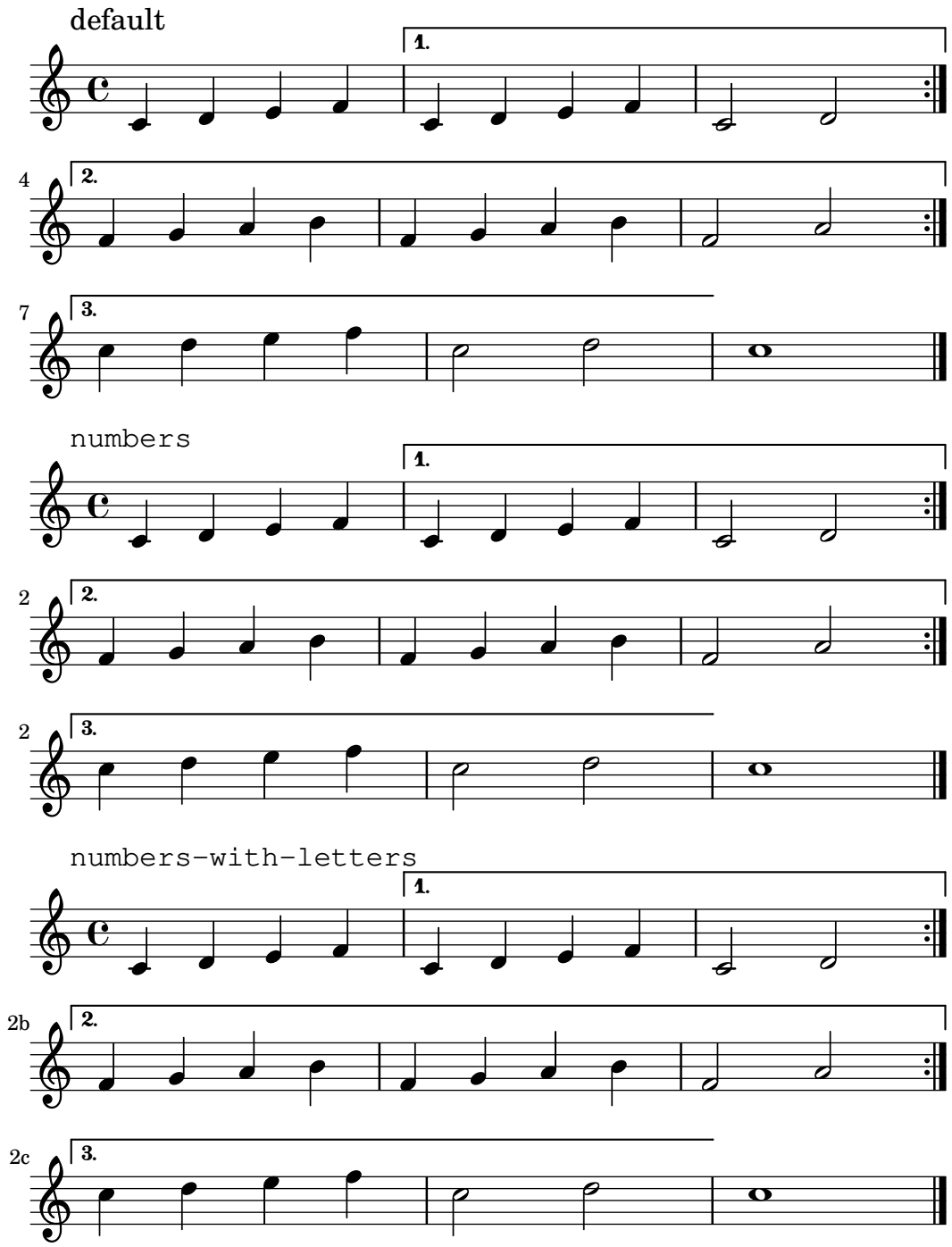
```
music = \relative c' {
  \repeat volta 3 {
    c4 d e f |
    \alternative {
      \volta 1 { c4 d e f | c2 d \break }
      \volta 2 { f4 g a b | f4 g a b | f2 a | \break }
      \volta 3 { c4 d e f | c2 d } } }
  c1 \bar "|."
}

{
  \textMark \markup \large "default"
  \music
}

{
  \textMark \markup \large \typewriter "numbers"
  \set Score.alternativeNumberingStyle = #'numbers
  \music
}

{
  \textMark \markup \large \typewriter "numbers-with-letters"
  \set Score.alternativeNumberingStyle = #'numbers-with-letters
  \music
}

\layout {
  \context {
    \Score
    \override TextMark.Y-offset = #5
  }
}
```



See also

Music Glossary: Section “repeat” in *Music Glossary*, Section “volta” in *Music Glossary*.

Notation Reference: Section 2.5.2 [Automatic bar lines], page 131, Section 2.5.1 [Bar lines], page 121, Section 2.3.1 [Measures], page 78, Section 33.4 [Modifying context plug-ins], page 745, Section 36.11.1 [Modifying ties and slurs], page 801.

Installed Files: `ly/engraver-init.ly`.

Snippets: Section “Repeats” in *Snippets*.

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

Known issues and warnings

For repeats in `volta` form, spanners (slurs, etc.) that cross into alternatives work for the first alternative only. They likewise cannot wrap around from the end of an alternative back to the beginning of the repeated section.

The visual appearance of a continuing slur or tie in subsequent alternatives can be achieved with `\repeatTie` if the slur extends into only one note in the alternative block, although this method does not work in `TabStaff`; see [Repeat tie], page 64. Other methods which may be tailored to indicate continuing slurs over several notes in alternative blocks, and which also work in `TabStaff` contexts, are shown in Section 36.11.1 [Modifying ties and slurs], page 801.

The visual appearance of a continuing glissando in subsequent alternatives can be achieved by coding a glissando starting on a hidden grace note. See [Extending glissandi across repeats], page 181.

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 using the old `\alternative` syntax is ambiguous:

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

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 advised to add braces and place the `\alternative` block inside the body of its repeat.
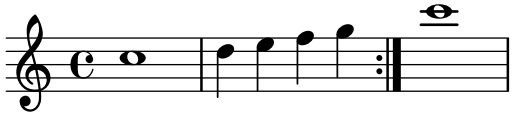
4.1.8 Manual repeat marks

> **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 Section 2.5.1 [Bar lines], page 121.

The property `repeatCommands` can be used to control the layout of volta-style repeats. Its value is a Scheme list. In general, each element is itself a list, `'(`*command args...*`)`, but a command with no arguments may be abbreviated to a symbol; e.g., `'((start-repeat))` may be given as `'(start-repeat)`.
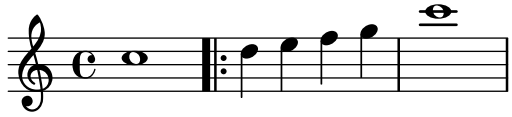
`end-repeat`
: End a repeated section.

```
\relative {
  c''1
  d4 e f g
  \set Score.repeatCommands = #'(end-repeat)
  c1
}
```



`start-repeat`
: Start a repeated section.

```
\relative {
  c''1
  \set Score.repeatCommands = #'(start-repeat)
  d4 e f g
  c1
}
```
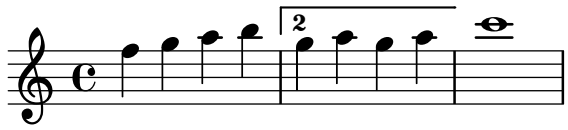


As per standard engraving practice, repeat signs are not printed at the beginning of a piece.

volta *text*

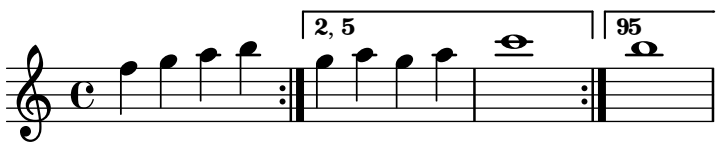
If *text* is markup, start a volta bracket with that label; if *text* is `#f`, end a volta bracket. A volta bracket which is not ended explicitly will not be printed.

```
\relative {
  f''4 g a b
  \set Score.repeatCommands =
    #`((volta ,#{ \markup \volta-number "2" #}))
  g4 a g a
  \set Score.repeatCommands = #'((volta #f))
  c1
}
```
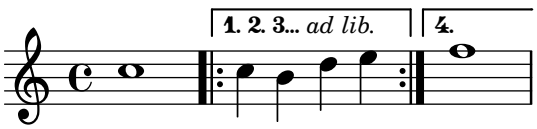


Multiple repeat commands may occur at the same point:

```
\relative {
  f''4 g a b
  \set Score.repeatCommands =
    #`((volta ,#{ \markup { \concat { \volta-number 2 , }
                            \volta-number 5 } #})
       end-repeat)
  g4 a g a
  c1
  \set Score.repeatCommands =
    #`((volta #f)
       (volta ,#{ \markup \volta-number 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 8.2 [Formatting text], page 320. The simplest way to use markup text is to define the markup first, then include the markup in a Scheme list.

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

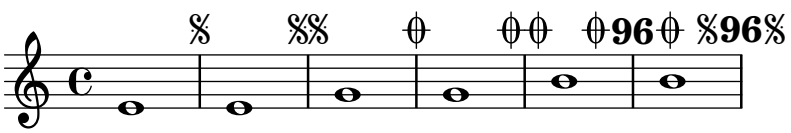


Segno and coda marks can be created with the \segnoMark, \inStaffSegno, and \codaMark commands. This is the syntax for the mark commands:

```
\codaMark n
\segnoMark n
```

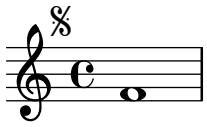
where *n* is a sequence number, or \default to use the next automatically. Rehearsal, segno, and coda marks are counted independently.

```
\fixed c' {
  e1
  \segnoMark \default
  e1
  \segnoMark \default
  g1
  \codaMark \default
  g1
  \codaMark \default
  b1
  \codaMark 96
  b1
  \segnoMark 96
}
```



At the beginning of a piece, \segnoMark \default and \codaMark \default create no mark. Specify '1' to force a mark.

```
\fixed c' {
  \segnoMark 1
  f1
}
```



The \inStaffSegno command is equivalent to \segnoMark \default with the extra effect of temporarily setting the segnoStyle property to bar-line to force printing it as a bar line.
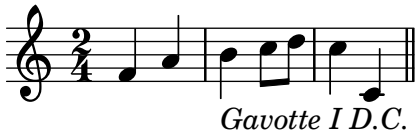
```
\fixed c' {
  e1
  \inStaffSegno
  g1
  \segnoMark \default
  b1
}
```



For more information on changing the appearance of segno and coda marks, see Section 4.1.7 [Segno repeat appearance], page 199.

To create arbitrary jump instructions, use the \jump command.

```
\fixed c' {
  \time 2/4
  f4 a
  b4 c'8 d'
  c'4 c
  \jump "Gavotte I D.C."
  \section
}
```



See also

Notation Reference: Section 2.5.1 [Bar lines], page 121, Section 8.2 [Formatting text], page 320, Section 2.5.5 [Rehearsal marks], page 142, Section 4.1.7 [Segno repeat appearance], page 199.

Snippets: Section “Repeats” in *Snippets*.

Internals Reference: Section “CodaMark” in *Internals Reference*, Section “Jump_engraver” in *Internals Reference*, Section “JumpScript” in *Internals Reference*, Section “Mark_engraver” in *Internals Reference*, Section “SegnoMark” in *Internals Reference*, Section “SegnoRepeatedMusic” in *Internals Reference*, Section “VoltaBracket” in *Internals Reference*, Section “VoltaRepeatedMusic” in *Internals Reference*.

4.2 Short repeats

This section discusses how to input short repeats. Short repeats can take two forms: slashes or percent signs to represent repeats of a single note, a single measure or two measures, and tremolos otherwise.

4.2.1 Percent repeats

Repeated short patterns are printed once, and the repeated pattern is replaced with a special sign.
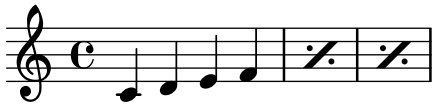
The syntax is

```
\repeat percent number musicexpr
```

where *musicexpr* is a music expression.

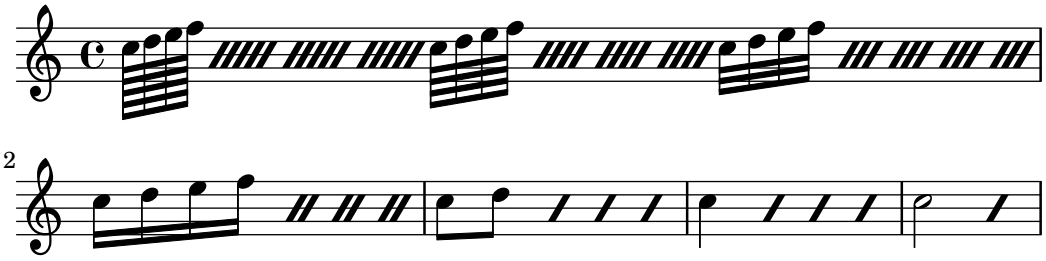
There is also a short form, \%, which can be convenient in a work that uses percent repetition heavily:

```
\%3 { c'4 d' e' f' }
```
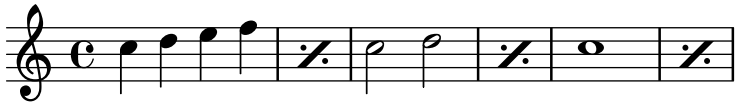


Patterns that are shorter than one measure are replaced by slashes.

```
\relative c'' {
  \%4 { c128 d e f }
  \%4 { c64 d e f }
  \%5 { c32 d e f }
  \%4 { c16 d e f }
  \%4 { c8 d }
  \%4 { c4 }
  \%2 { c2 }
}
```
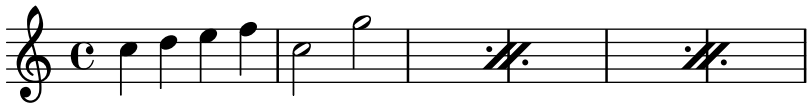


Patterns of one or two measures are replaced by percent-like symbols.

```
\relative c'' {
  \%2 { c4 d e f }
  \%2 { c2 d }
  \%2 { c1 }
}
```

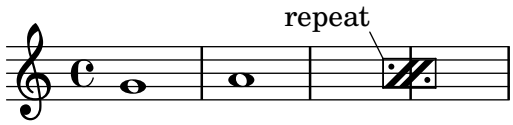


```
\relative {
  \%3 { c''4 d e f | c2 g' }
}
```



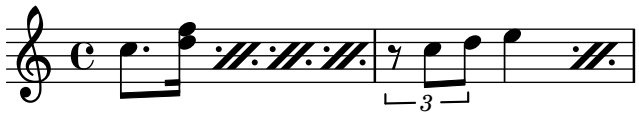
Note that the correct time offset to access the DoublePercentRepeat grob is the beginning of the repeat's second bar, which can be easily achieved with \after.

```
\new Voice \with { \consists Balloon_engraver }
  { \after 1*3
      \balloonGrobText DoublePercentRepeat #'(-1 . 2) "repeat"
    \%2 { g'1 | a'1 } }
```



Patterns that are shorter than one measure but contain mixed durations use a double-percent symbol.

```
\relative {
  \%4 { c''8. <d f>16 }
  \%2 { \tuplet 3/2 { r8 c d } e4 }
}
```

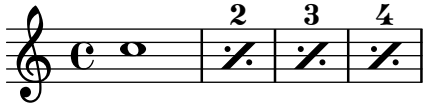


Selected snippets

Percent repeat counter

Measure repeats of more than two repeats are printed with a counter if the countPercentRepeats context property is set.

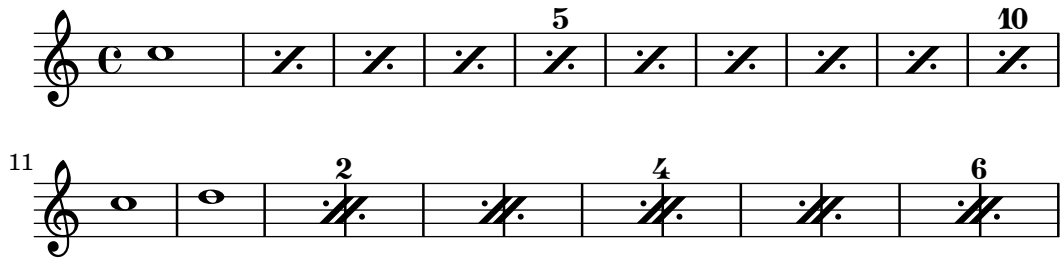
```
\relative c'' {
  \set countPercentRepeats = ##t
  \%4 c1
}
```



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)
  \%10 c1 \break
  \set repeatCountVisibility = #(every-nth-repeat-count-visible 2)
  \%6 { c1 d1 }
}
```
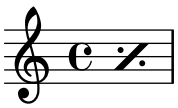


Isolated percent repeats

Isolated percents can also be printed.

```
makePercent =
#(define-music-function (note) (ly:music?)
   "Make a percent repeat the same length as NOTE."
   (make-music 'PercentEvent
               'length (ly:music-length note)))

\relative c'' {
  \makePercent s1
}
```



See also

Music Glossary: Section “percent repeat” in *Music Glossary*, Section “simile” in *Music Glossary*.
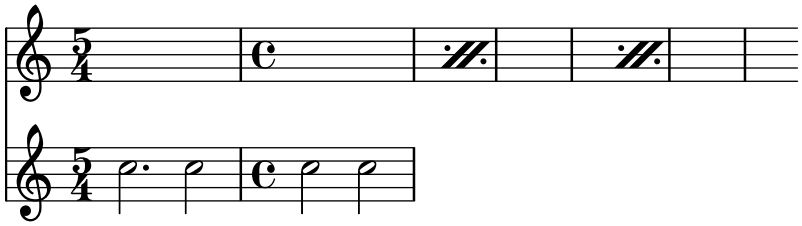
Snippets: Section “Repeats” in *Snippets*.

Internals Reference: Section “RepeatSlash” in *Internals Reference*, Section “RepeatSlashEvent” in *Internals Reference*, Section “DoubleRepeatSlash” in *Internals Reference*, Section “PercentRepeat” in *Internals Reference*, Section “PercentRepeatCounter” in *Internals Reference*, Section “PercentRepeatedMusic” in *Internals Reference*, Section “Percent_repeat_engraver” in *Internals Reference*, Section “DoublePercentEvent” in *Internals Reference*, Section “DoublePercentRepeat” in *Internals Reference*, Section “DoublePercentRepeatCounter” in *Internals Reference*, Section “Double_percent_repeat_engraver” in *Internals Reference*, Section “Slash_repeat_engraver” in *Internals Reference*.

Known issues and warnings
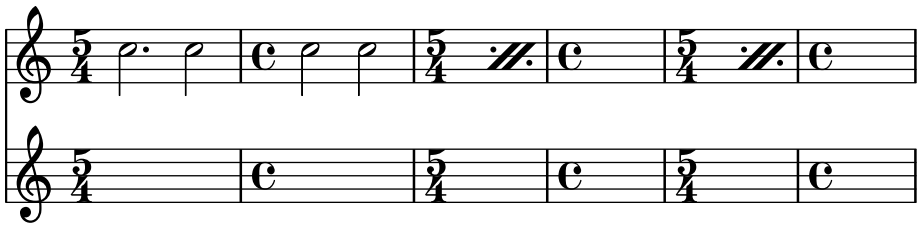
Percent repeats will not contain anything else apart from the percent sign itself; in particular, timing changes will not be repeated.

```
\%3 { \time 5/4 c2. 2 \time 4/4 2 2 }
```



Any meter changes or \partial commands need to occur in parallel passages *outside* of any percent repeat, e.g in a separate timing track.

```
<<
  \%3 { c2. 2 2 2 }
  \*3 { \time 5/4 s4*5 \time 4/4 s1 }
>>
```
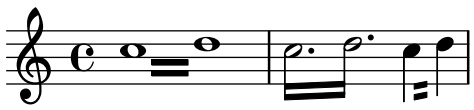


4.2.2 Tremolo repeats

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 with tremolo style:

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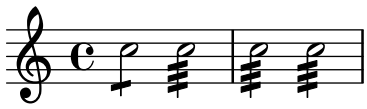
There are two ways to put tremolo marks on a single note. The \repeat tremolo syntax is also used here, in which case the note should not be surrounded by braces:

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



The same output can be obtained by adding :*N* after the note, where *N* indicates the duration of the subdivision (it must be at least 8). If *N* is 8, one beam is added to the note's stem. If *N* is omitted, the last value is used:

```
\relative {
  c''2:8 c:32
  c: c:
}
```
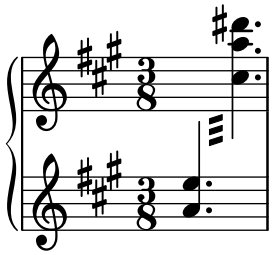


Selected snippets

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
      }
    }
  }
>>
```



Controlling the appearance of tremolo slashes

Using various properties of the `StemTremolo` grob it is possible to control the appearance of tremolo slashes.

- Property `slope` sets the slope for tremolo slashes.

- Property `shape` determines whether tremolo slashes look like rectangles (value `rectangle`) or like very small beams (value `beam-like`).
- Property `style` sets both the slope and the shape depending on whether the note has flags, beams, or only a plain stem. This is in contrast to the previous two properties, which change the slope and shape unconditionally. There are two styles defined.
  - `default`: slashes for down-stem flags are longer and more sloped than slashes for up-stem flags; slashes on beamed notes have a rectangular shape and are parallel to the beam.
  - `constant`: all slashes are beam-like and have the same slope except for down-stem flags.

```
music = {
  a''4:32 a':
  e''8: \noBeam e':
  a'':[ a':]
  f':[ g':]
  d':[ d':]
}
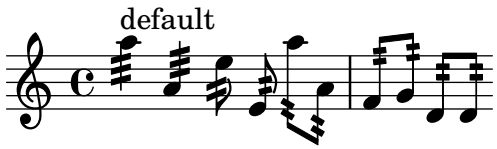
\new Staff {
  <>^\markup "default"
  \music
}

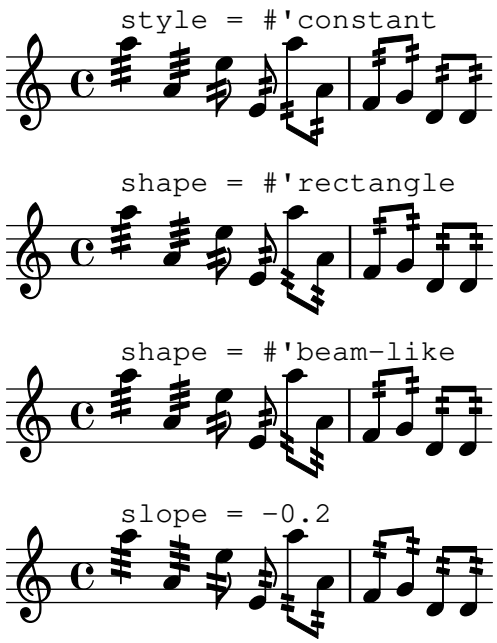
\new Staff {
  <>^\markup \typewriter "style = #'constant"
  \override StemTremolo.style = #'constant
  \music
}

\new Staff {
  <>^\markup \typewriter "shape = #'rectangle"
  \override StemTremolo.shape = #'rectangle
  \music
}

\new Staff {
  <>^\markup \typewriter "shape = #'beam-like"
  \override StemTremolo.shape = #'beam-like
  \music
}

\new Staff {
  <>^\markup \typewriter "slope = -0.2"
  \override StemTremolo.slope = -0.2
  \music
}
```
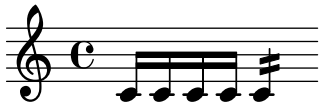




See also

Snippets: Section “Repeats” in *Snippets*.
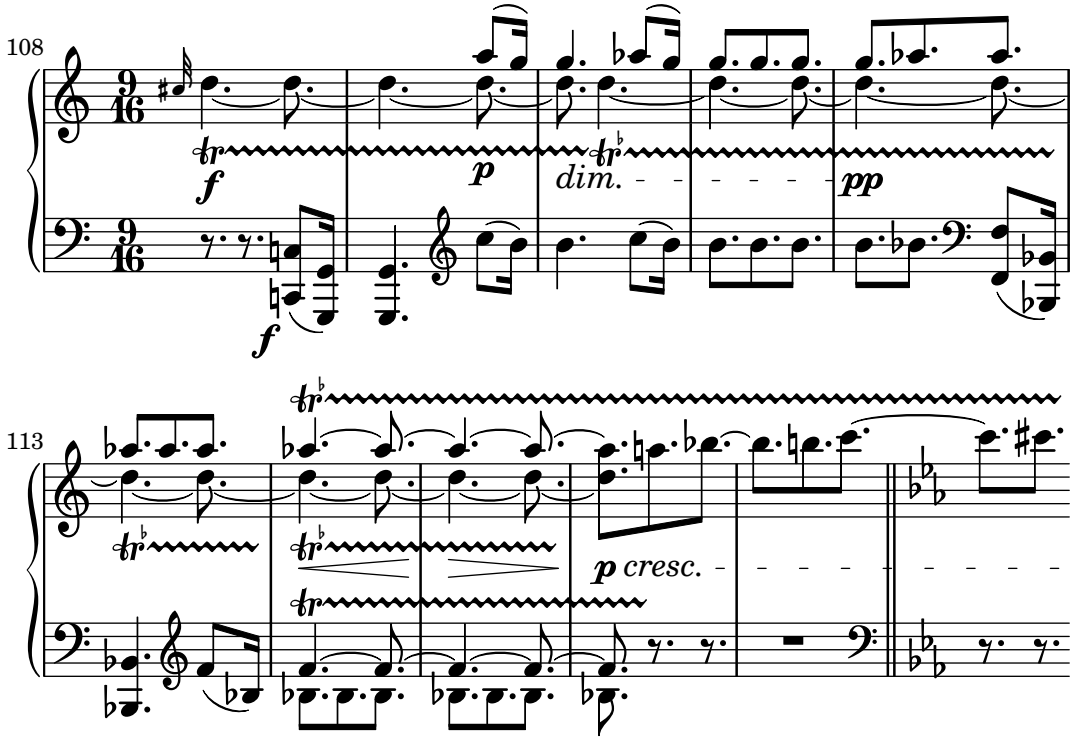
Known issues and warnings

While the `\repeat tremolo` syntax is unfolded by `\unfoldRepeats`, its shorthand is not (this is tracked in issue #6145 (`https://gitlab.com/lilypond/lilypond/-/issues/6145`)):

```
{
  \unfoldRepeats \repeat tremolo 4 c'16
  \unfoldRepeats c'4:16
}
```



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.

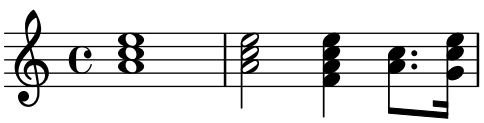
5.1 Single voice

This section discusses simultaneous notes inside the same voice.

5.1.1 Chorded notes

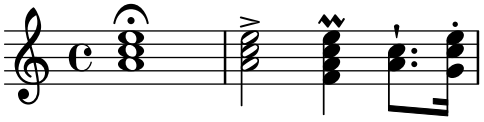
A chord is formed by enclosing a set of pitches between < and >. A chord may be followed by a duration just like simple notes.

```
\relative {
  <a' c e>1 <a c e>2 <f a c e>4 <a c>8. <g c e>16
}
```



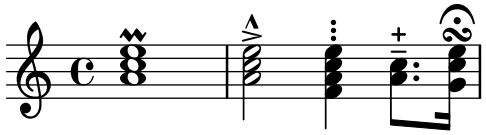
Chords may also be followed by articulations, again just like simple notes.

```
\relative {
  <a' c e>1\fermata <a c e>2-> <f a c e>4\prall <a c>8.^! <g c e>16-.
}
```



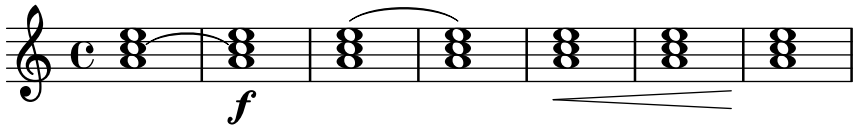
The notes within the chord themselves can also be followed by articulation and ornamentation.

```
\relative {
  <a' c\prall e>1 <a-> c-^ e>2 <f-. a c-. e-.>4
  <a-+ c-->8. <g\fermata c e\turn>16
}
```
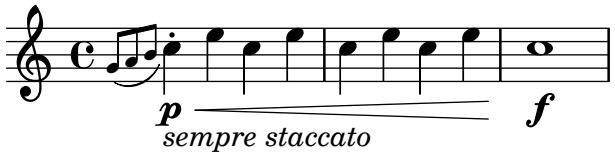


However some notation, such as dynamics and hairpins must be attached to the chord rather than to notes within the chord, otherwise they will not print. Other notation like fingerings and slurs will get placed markedly different when attached to notes within a chord rather than to whole chords or single notes.

```
\relative {
  <a'\f c( e>1 <a c) e>\f <a\< c e>( <a\! c e>)
  <a c e>\< <a c e> <a c e>\!
}
```



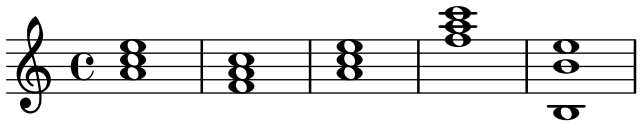
A chord acts merely as a container for its notes, its articulations and other attached elements. Consequently, a chord without notes inside does not actually have a duration. Any attached articulations will happen at the same musical time as the next following note or chord and be combined with them (for more complex possibilities of combining such elements, see Section 5.1.3 [Simultaneous expressions], page 220):

```
\relative {
  \grace { g'8( a b }
  <>) \p \< -. -\markup \italic "sempre staccato"
  \*4 { c4 e }  c1\f
}
```



Relative mode can be used for pitches in chords. The first note of each chord is always relative to the first note of the chord that came before it, or in the case where no preceding chord exists, the pitch of the last note that came before the chord. All remaining notes in the chord are relative to the note that came before it *within the same chord*.

```
\relative {
  <a' c e>1 <f a c> <a c e> <f' a c> <b, e b,>
}
```



For more information about chords, see Chapter 15 [Chord notation], page 508.

See also

Music Glossary: Section “chord” in *Music Glossary*.

Learning Manual: Section “Combining notes into chords” in *Learning Manual*.

Notation Reference: Chapter 15 [Chord notation], page 508, Section 3.1.1 [Articulations and ornamentations], page 156, Section 1.1.2 [Relative octave entry], page 4, Section 5.2 [Multiple voices], page 222.

Snippets: Section “Simultaneous notes” in *Snippets*.

Known issues and warnings

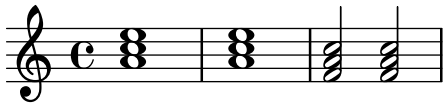
Chords containing more than two pitches within a staff space, such as ‘`<e f! fis!>`’, create overlapping note heads. Depending on the situation, better representations might involve

- temporary use of Section 5.2 [Multiple voices], page 222, ‘`<< f! \\ <e fis!> >>`’,
- enharmonic transcription of one or more pitches, ‘`<e f ges>`’, or
- Section 5.1.4 [Clusters], page 221.

5.1.2 Chord repetition

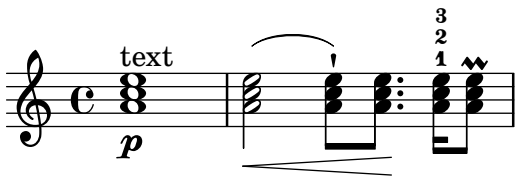
In order to save typing, a shortcut can be used to repeat the preceding chord. The chord repetition symbol is `q`:

```
\relative {
  <a' c e>1 q <f a c>2 q
}
```



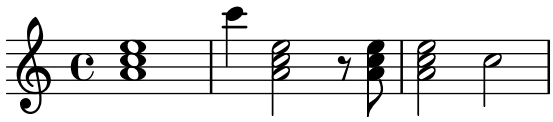
As with regular chords, the chord repetition symbol can be used with durations, articulations, markups, slurs, beams, etc., as only the pitches of the previous chord are duplicated.

```
\relative {
  <a' c e>1\p^"text" q2\<( q8)[-! q8.]\! q16-1-2-3 q8\prall
}
```
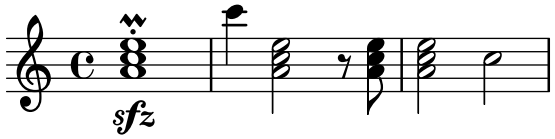


The chord repetition symbol always remembers the last instance of a chord so it is possible to repeat the most recent chord even if other non-chorded notes or rests have been added since.

```
\relative {
  <a' c e>1 c'4 q2 r8 q8 |
  q2 c, |
}
```



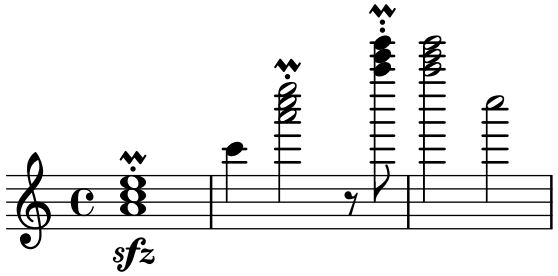
However, the chord repetition symbol does not retain any dynamics, articulation or ornamentation within, or attached to, the previous chord.

```
\relative {
  <a'-. c\prall e>1\sfz c'4 q2 r8 q8 |
  q2 c, |
}
```



To have some of them retained, the \chordRepeats function can be be called explicitly with an extra argument specifying a list of *event types* to keep unless events of that type are already present on the q chord itself.
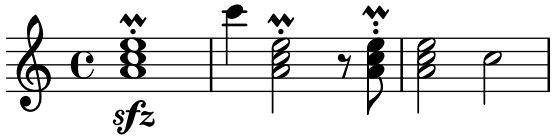
```
\relative {
  \chordRepeats #'(articulation-event)
  { <a'-. c\prall e>1\sfz c'4 q2 r8 q8-. } |
  q2 c, |
}
```



Here using \chordRepeats inside of a \relative construction produces unexpected results: once chord events have been expanded, they are indistinguishable from having been entered as regular chords, making \relative assign an octave based on their current context.

Since nested instances of \relative don't affect one another, another \relative inside of \chordRepeats can be used for establishing the octave relations before expanding the repeat chords. In that case, the whole content of the inner \relative does not affect the outer one; hence the different octave entry of the final note in this example.

```
\relative {
  \chordRepeats #'(articulation-event)
  \relative
  { <a'-. c\prall e>1\sfz c'4 q2 r8 q8-. } |
  q2 c'' |
}
```



Interactions with \relative occur only with explicit calls of \chordRepeats: the implicit expansion at the start of typesetting is done at a time where all instances of \relative have already been processed.

See also

Notation Reference: Chapter 15 [Chord notation], page 508, Section 3.1.1 [Articulations and ornamentations], page 156.
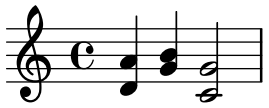
Installed Files: ly/chord-repetition-init.ly.

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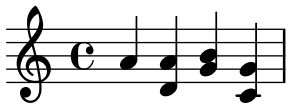
One or more music expressions enclosed in double angle brackets are taken to be simultaneous. If the first expression begins with a single note or if the whole simultaneous expression appears explicitly within a single voice, the whole expression is placed on a single staff; otherwise the elements of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

The following examples show simultaneous expressions on one staff:

```
\new Voice {  % explicit single voice
  << \relative { a'4 b g2 }
     \relative { d'4 g c,2 } >>
}
```
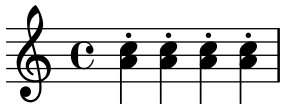


```
\relative {
  % single first note
  a' << \relative { a'4 b g }
        \relative { d'4 g c, } >>
}
```



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. Notes, articulations, and property changes in a *single* 'Voice' are collected and engraved in musical order:
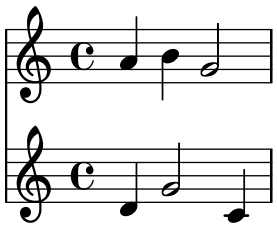
```
\relative {
  <a' c>4-.  <>-. << c a >>  << { c-. <c a> } { a s-. } >>
}
```



Multiple stems or beams or different note durations or properties at the same musical time require the use of multiple voices.

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

```
% no single first note
<< \relative { a'4 b g2 }
   \relative { d'4 g2 c,4 } >>
```



Here different rhythms cause no problems because they are interpreted in different voices.

Known issues and warnings

If notes from two or more voices, with no shifts specified, have stems in the same direction, the message

```
warning: This voice needs a \voiceXx or \shiftXx setting
```

will appear during compilation. This message can be suppressed by:
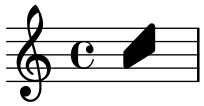
```
\override NoteColumn.ignore-collision = ##t
```

However, this not only suppresses the warning but will prevent any collision resolution whatsoever and may have other unintended effects (also see *Known Issues* in Section 5.2.3 [Collision resolution], page 227).

5.1.4 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.,

```
\relative \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.

The following cluster styles are supported: `ramp`, `leftsided-stairs`, `rightsided-stairs`, and `centered-stairs`.

```
fragment = { <e' d''>4 <g' a'> <e' a'> r }

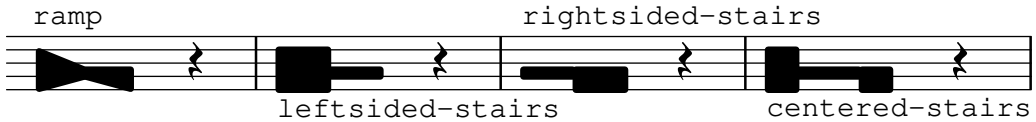
{
  \omit Staff.Clef
  \omit Staff.TimeSignature

  <>^\markup \typewriter "ramp"
  \override ClusterSpanner.style = #'ramp
  \makeClusters \fragment

  <>_\markup \typewriter "leftsided-stairs"
  \override ClusterSpanner.style = #'leftsided-stairs
  \makeClusters \fragment

  <>^\markup \typewriter "rightsided-stairs"
  \override ClusterSpanner.style = #'rightsided-stairs
  \makeClusters \fragment

  <>_\markup \typewriter "centered-stairs"
  \override ClusterSpanner.style = #'centered-stairs
  \makeClusters \fragment
}
```



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.

5.2 Multiple voices
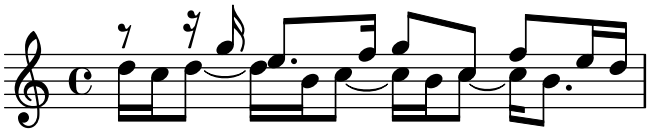
This section discusses simultaneous notes in multiple voices or multiple staves.

5.2.1 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"
    \relative { \voiceOne r8 r16 g'' e8. f16 g8[ c,] f e16 d }
  \new Voice= "second"
    \relative { \voiceTwo d''16 c d8~ 16 b c8~ 16 b c8~ 16 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.

Note that `Voice` is a bottom-level context (see Section 33.1.5 [Bottom-level contexts – voices], page 738). In `TabStaff` one would use `TabVoice` instead.

Temporary polyphonic passages
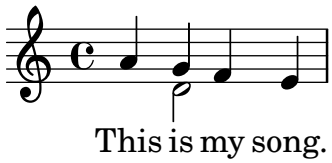
A temporary polyphonic passage can be created with the following construct:

```
<< { \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:

```
\relative <<
  \new Voice = "melody" {
    a'4
    <<
      {
        \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. In `TabStaff` one needs to use `TabVoice`.
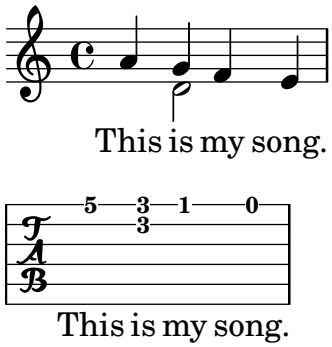
If the same music should appear in `Staff` *and* `TabStaff` the general `Bottom` context may be used (see Section 33.1.5 [Bottom-level contexts – voices], page 738).

```
mus =
\relative
  \new Bottom = "melody" {
    a'4
    <<
      {
        \voiceOne
        g f
      }
      \new Bottom {
        \voiceTwo
        d2
      }
    >>
    \oneVoice
    e4
  }

<<
```

```
  \new Staff \mus
  \new Lyrics \lyricsto "melody" {
    This is my song.
  }
>>

<<
  \new TabStaff \mus
  \new Lyrics \lyricsto "melody" {
    This is my song.
  }
>>
```
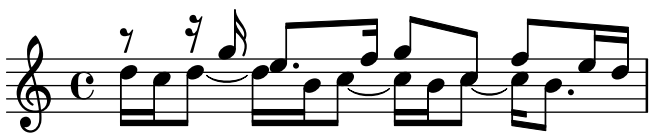


The double backslash construct

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 `Bottom` contexts of the current type, typically `Voice` or `TabVoice` (see Section 33.1.5 [Bottom-level contexts – voices], page 738). These new `Bottom` contexts are created implicitly and are given the fixed names `"1"`, `"2"`, etc.

The first example could be typeset as follows:
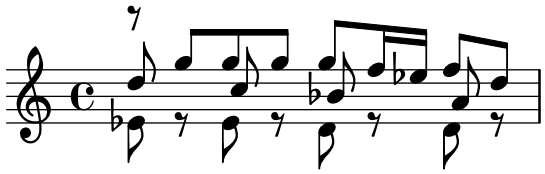
```
<<
  \relative { r8 r16 g'' e8. f16 g8[ c,] f e16 d }
  \\
  \relative { d''16 c d8~ 16 b c8~ 16 b c8~ 16 b8. }
>>
```



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.

```
<<
  \relative { r8 g'' g g g f16 ees f8 d }
  \\
  \relative { ees'8 r ees r d r d r }
  \\
  \relative { 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*.
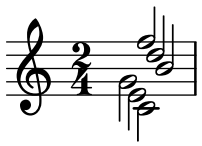
Voice order

When entering multiple voices in the input file, use the following order:

```
Voice 1: highest
Voice 2: lowest
Voice 3: second highest
Voice 4: second lowest
Voice 5: third highest
Voice 6: third lowest
etc.
```
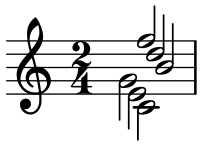
Though this may seem counterintuitive, it simplifies the automatic layout process. Note that the odd-numbered voices are given up-stems, and the even-numbered voices are given down-stems:

```
\new Staff <<
  \time 2/4
  { f''2 }  % 1: highest
  \\
  { c'2  }  % 2: lowest
  \\
  { d''2 }  % 3: second-highest
  \\
  { e'2  }  % 4: second-lowest
  \\
  { b'2  }  % 5: third-highest
  \\
  { g'2  }  % 6: third-lowest
>>
```



When a different voice entry order is desired, the command `\voices` may be convenient:

```
\new Staff \voices 1,3,5,6,4,2 <<
  \time 2/4
  { f''2 }  % 1: highest
  \\
  { d''2 }  % 3: second-highest
  \\
  { b'2  }  % 5: third-highest
  \\
  { g'2  }  % 6: third-lowest
  \\
  { e'2  }  % 4: second-lowest
  \\
  { c'2  }  % 2: lowest
>>
```
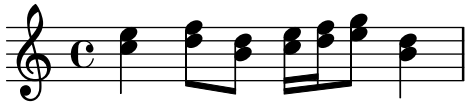


Note: Lyrics and spanners (such as slurs, ties, hairpins, etc.) cannot be created 'across' voices.

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 <<
  \relative { e''4 f8 d e16 f g8 d4 }
  \relative { c''4 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*.

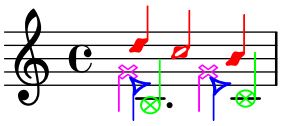
Notation Reference: Section 13.1.5 [Percussion staves], page 489, Section 2.2.2 [Invisible rests], page 69, Section 7.1.9 [Stems], page 297.

Snippets: Section "Simultaneous notes" in *Snippets*.

5.2.2 Voice styles

Voices may be given distinct colors and shapes, allowing them to be easily identified:

```
<<
  \relative { \voiceOneStyle d''4 c2 b4 }
  \\
  \relative { \voiceTwoStyle e'2 e }
  \\
  \relative { \voiceThreeStyle b2. c4 }
  \\
  \relative { \voiceFourStyle g'2 g }
>>
```



The `\voiceNeutralStyle` command is used to revert to the standard presentation.

Predefined commands

`\voiceOneStyle`, `\voiceTwoStyle`, `\voiceThreeStyle`, `\voiceFourStyle`, `\voiceNeutralStyle`.

See also
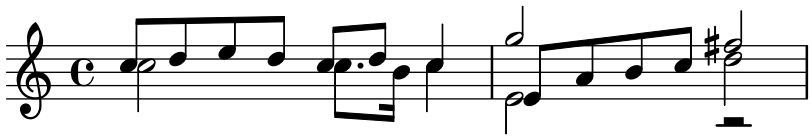
Learning Manual: Section “I’m hearing voices” in *Learning Manual*, Section “Other sources of information” in *Learning Manual*.

Snippets: Section “Simultaneous notes” in *Snippets*.

5.2.3 Collision resolution
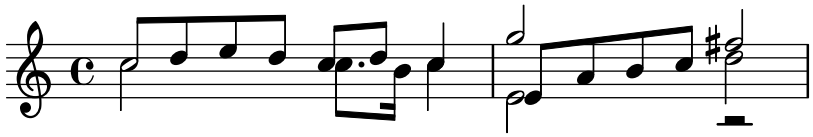
The note heads of notes in different voices with the same pitch, same note head and opposite stem direction are automatically merged, but notes with different note heads or the same stem direction are not. Rests opposite a stem in a different voice are shifted vertically. The following example shows three different circumstances, on beats 1 and 3 in bar 1 and beat 1 in bar 2, where the automatic merging fails.

```
<<
  \relative {
    c''8 d e d c d c4
    g'2 fis
  } \\
  \relative {
    c''2 c8. b16 c4
    e,2 r
  } \\
  \relative {
    \oneVoice
    s1
    e'8 a b c d2
  }
>>
```



Notes with different note heads may be merged as shown below. In this example the note heads on beat 1 of bar 1 are now merged:

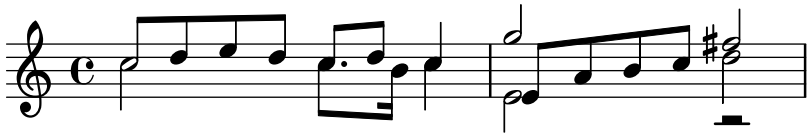
```
<<
  \relative {
    \mergeDifferentlyHeadedOn
    c''8 d e d c d c4
    g'2 fis
  } \\
  \relative {
    c''2 c8. b16 c4
    e,2 r
  } \\
  \relative {
    \oneVoice
    s1
    e'8 a b c d2
  }
>>
```



Quarter and half notes are not merged in this way, since it would be difficult to tell them apart.
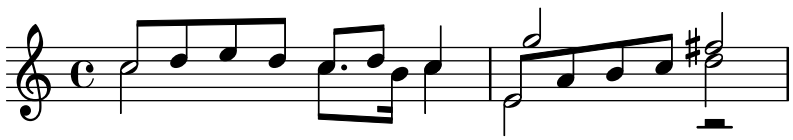
Note heads with different dots as shown in beat 3 of bar 1 may be also be merged:

```
<<
  \relative {
    \mergeDifferentlyHeadedOn
    \mergeDifferentlyDottedOn
    c''8 d e d c d c4
    g'2 fis
  } \\
  \relative {
    c''2 c8. b16 c4
    e,2 r
  } \\
  \relative {
    \oneVoice
    s1
    e'8 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.

```
<<
  \relative {
    \mergeDifferentlyHeadedOn
    \mergeDifferentlyDottedOn
    c''8 d e d c d c4
    \shiftOn
    g'2 fis
  } \\
  \relative {
    c''2 c8. b16 c4
    e,2 r
  } \\
  \relative {
    \oneVoice
    s1
    e'8 a b c d2
  }
>>
```



The `\shiftOn` command allows (but does not force) the notes in a voice to be shifted. When `\shiftOn` is applied to a voice, a note or chord in that voice is shifted only if its stem would otherwise collide with a stem from another voice, and only if the colliding stems point in the same direction. The `\shiftOff` command prevents this type of shifting from occurring.
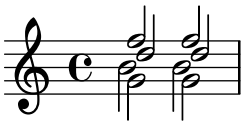
By default, the outer voices (normally voices one and two) have `\shiftOff` specified, while the inner voices (three and above) have `\shiftOn` specified. When a shift is applied, voices with up-stems (odd-numbered voices) are shifted to the right, and voices with down-stems (even-numbered voices) are shifted to the left.

Here is an example to help you visualize how an abbreviated polyphonic expression would be expanded internally.

> **Note:** Note that with three or more voices, the vertical order of voices in your input file should not be the same as the vertical order of voices on the staff!

```
\new Staff \relative {
  %% abbreviated entry
  <<
    { f''2  }  % 1: highest
    \\
    { g,2 }  % 2: lowest
    \\
```

```
    { d'2 }  % 3: upper middle
    \\
    { b2  }  % 4: lower middle
  >>
  %% internal expansion of the above
  <<
    \new Voice = "1" { \voiceOne   \shiftOff f'2 }
    \new Voice = "2" { \voiceTwo   \shiftOff g,2 }
    \new Voice = "3" { \voiceThree \shiftOn  d'2 } % shifts right
    \new Voice = "4" { \voiceFour  \shiftOn  b2  } % shifts left
  >>
}
```



Two additional commands, `\shiftOnn` and `\shiftOnnn` provide further shift levels which may be specified temporarily to resolve collisions in complex situations – see Section "Real music example" in *Learning Manual*.

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
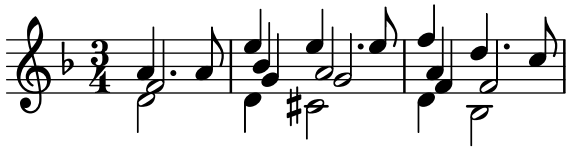
Additional voices to avoid collisions

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
  <<
    \new Voice  {
      \voiceOne
      a4. a8
      e'4 e4. e8
      f4 d4. c8
    }
    \new Voice {
      \voiceTwo
      d,2
      d4 cis2
```
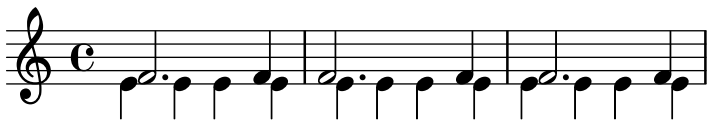
```
      d4 bes2
    }
    \new Voice {
      \voiceThree
      f'2
      bes4 a2
      a4 s2
    }
    \new Voice {
      \voiceFive
      s2
      g4 g2
      f4 f2
    }
  >>
}
```



Moving dotted notes in polyphony

When a dotted note in the upper voice is moved to avoid a collision with a note in another voice, the default is to move the upper note to the right. This behaviour can be changed setting the `prefer-dotted-right` property of the `NoteCollision` grob.

```
\new Staff \relative c' <<
  {
    f2. f4
    \override Staff.NoteCollision.prefer-dotted-right = ##f
    f2. f4
    \override Staff.NoteCollision.prefer-dotted-right = ##t
    f2. f4
  }
  \\
  { e4 e e e e e e e e e e }
>>
```

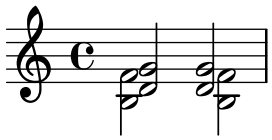


Forcing horizontal shift of notes

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*.

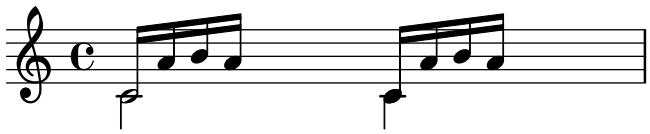
Snippets: Section “Simultaneous notes” in *Snippets*.

Internals Reference: Section “NoteColumn” in *Internals Reference*, Section “NoteCollision” in *Internals Reference*, Section “RestCollision” in *Internals Reference*.

Known issues and warnings

Using `\override NoteColumn.ignore-collision = ##t` will cause differently headed notes in different voices to merge incorrectly.

```
\mergeDifferentlyHeadedOn
<< \relative { c'16 a' b a } \\ \relative { c'2 } >>
\override NoteColumn.ignore-collision = ##t
<< \relative { c'16 a' b a } \\ \relative { c'2 } >>
```
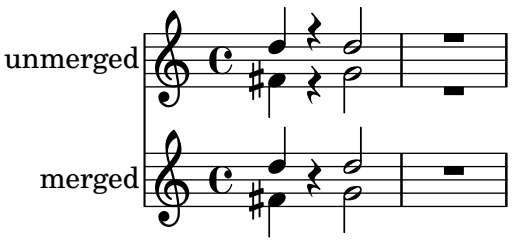


5.2.4 Merging rests

When using multiple voices it is common to merge rests which occur in both parts. This can be accomplished using `Merge_rests_engraver`.

```
voiceA = \relative { d''4 r d2 | R1 | }
voiceB = \relative { fis'4 r g2 | R1 | }
\score {
  <<
    \new Staff \with {
      instrumentName = "unmerged"
    }
    <<
      \new Voice { \voiceOne \voiceA }
      \new Voice { \voiceTwo \voiceB }
    >>
    \new Staff \with {
      instrumentName = "merged"
      \consists Merge_rests_engraver
    }
    <<
```

```
      \new Voice { \voiceOne \voiceA }
      \new Voice { \voiceTwo \voiceB }
    >>
  >>
}
```



Setting the context property `suspendRestMerging` to `#t` allows for turning off rest merging temporarily.

5.2.5 Automatic part combining

Automatic part combining is used to merge two separate parts of music onto a single staff. This can be especially helpful when typesetting orchestral scores. A single `Voice` is printed while the two parts of music are the same, but in places where they differ, a second `Voice` is printed. Stem directions are set up & down accordingly while Solo and *a due* parts are also identified and marked appropriately.

The syntax for automatic part combining is:
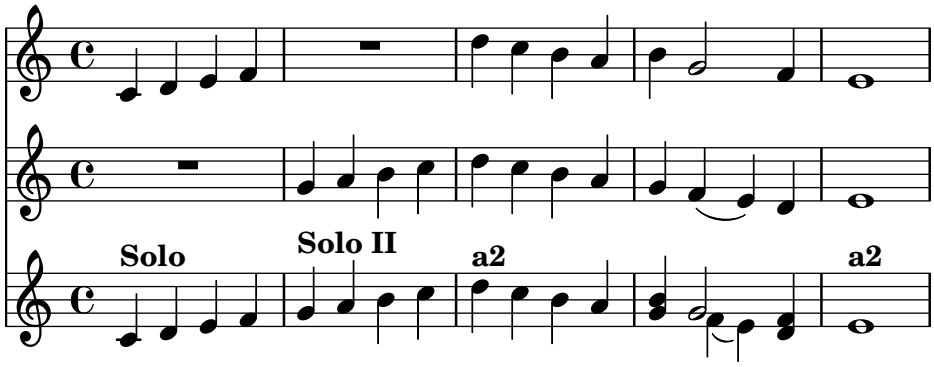
`\partCombine` *musicexpr1 musicexpr2*

The following example demonstrates the basic functionality, putting parts on a single staff as polyphony and setting stem directions accordingly. The same variables are used for the independent parts and the combined staff.

```
instrumentOne = \relative {
  c'4 d e f |
  R1 |
  d'4 c b a |
  b4 g2 f4 |
  e1 |
}

instrumentTwo = \relative {
  R1 |
  g'4 a b c |
  d4 c b a |
  g4 f( e) d |
  e1 |
}

<<
  \new Staff \instrumentOne
  \new Staff \instrumentTwo
  \new Staff \partCombine \instrumentOne \instrumentTwo
>>
```



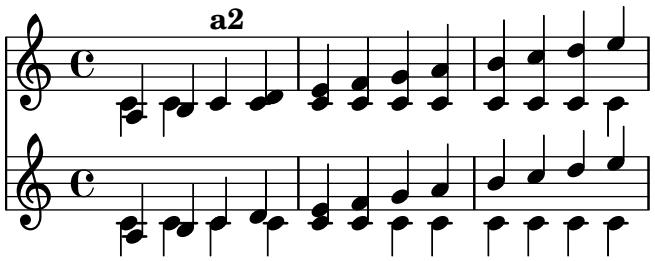
Both parts have identical notes in the third measure, so only one instance of the notes is printed. Stem, slur, and tie directions are set automatically, depending on whether the parts are playing solo or in unison. When needed in polyphony situations, the first part (with context called `one`) gets "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 unison (*a due*) parts are marked with the text "a2".

By default, the part combiner merges two notes of the same pitch as an *a due* note, combines notes with the same rhythm less than a ninth apart as chords and separates notes more than a ninth apart (or when the voices cross) into separate voices. This can be overridden with an optional argument of a pair of numbers after the `\partCombine` command: the first specifies the interval where notes start to be combined (the default is zero) and the second where the notes are split into separate voices. Setting the second argument to zero means that the part combiner splits notes with an interval of a second or more, setting it to one splits notes of a third or more, and so on.

```
instrumentOne = \relative {
  a4 b c d |
  e f g a |
  b c d e |
}

instrumentTwo = \relative {
  c'4 c c c |
  c c c c |
  c c c c |
}

<<
  \new Staff \partCombine \instrumentOne \instrumentTwo
  \new Staff \partCombine #'(2 . 3) \instrumentOne \instrumentTwo
>>
```



Both arguments to \partCombine will be interpreted as separate Voice contexts, so if the music is being specified in relative mode then *both* parts must contain a \relative function, i.e.,

```
\partCombine
  \relative ... musicexpr1
  \relative ... musicexpr2
```

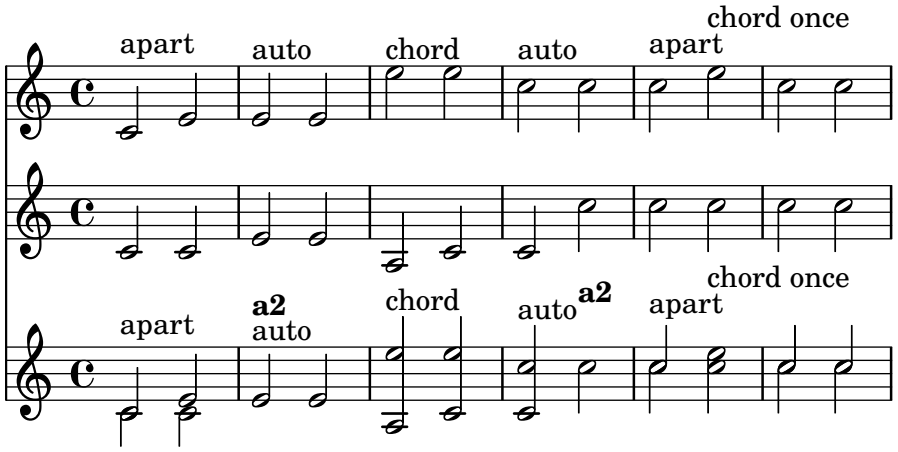
A \relative section that encloses a \partCombine has no effect on the pitches of *musicexpr1* or *musicexpr2*.

In professional scores, voices are often kept apart from each other for long passages of music even if some of the notes are the same in both voices, and could just as easily be printed as unison. Combining notes into a chord, or showing one voice as solo is, therefore, not ideal as the \partCombine function considers each note separately. In this case the \partCombine function can be overridden with one of the following commands. All of the commands may be preceded with \once in order to have them only apply to the next note in the music expression.

- \partCombineApart keeps the notes as two separate voices, even if they can be combined into a chord or unison.
- \partCombineChords combines the notes into a chord.
- \partCombineUnisono combines both voices as "unison".
- \partCombineSoloI prints only voice one, and marks it as a "Solo".
- \partCombineSoloII prints only voice two and marks it as a "Solo".
- \partCombineAutomatic ends the functions of the commands above, and reverts back to the standard \partCombine functionality.

```
instrumentOne = \relative c' {
  \partCombineApart c2^"apart" e |
  \partCombineAutomatic e2^"auto" e |
  \partCombineChords e'2^"chord" e |
  \partCombineAutomatic c2^"auto" c |
  \partCombineApart c2^"apart"
    \once \partCombineChords e^"chord once" |
  c2 c |
}
instrumentTwo = \relative {
  c'2 c |
  e2 e |
  a,2 c |
  c2 c' |
  c2 c |
  c2 c |
}

<<
  \new Staff { \instrumentOne }
  \new Staff { \instrumentTwo }
  \new Staff { \partCombine \instrumentOne \instrumentTwo }
>>
```



Using \partCombine with lyrics

The \partCombine command is not designed to work with lyrics; if one of the voices is explicitly named in order to attach lyrics to it, the part combiner will stop working. However, this effect can be achieved using a NullVoice context. See Section 9.2.6 [Polyphony with shared lyrics], page 376.

Selected snippets

Combining two parts on the same staff

The part combiner tool (i.e., the \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 them 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
}

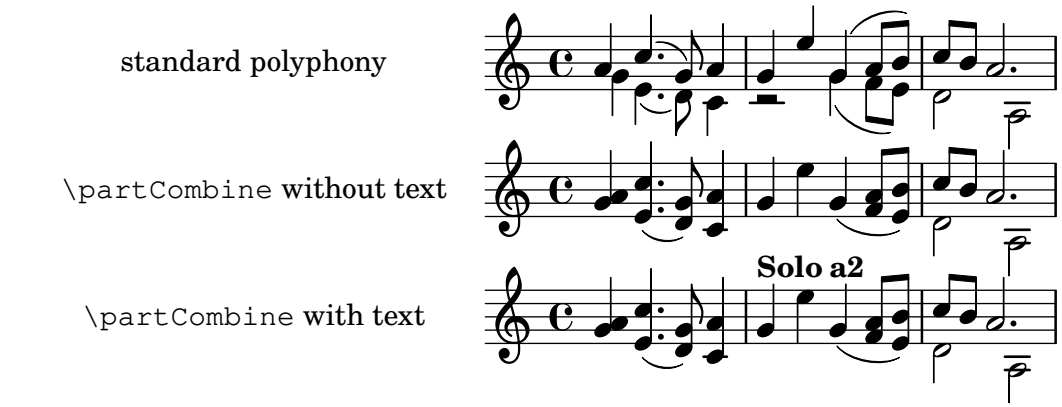
\score {
  <<
    \new Staff \with {
      instrumentName = "standard polyphony"
    } << \musicUp \\ \musicDown >>

    \new Staff \with {
```

```
      instrumentName =
        \markup { \typewriter "\\partCombine" without text}
      printPartCombineTexts = ##f
    } \partCombine \musicUp \musicDown

    \new Staff \with {
      instrumentName =
        \markup { \typewriter "\\partCombine" with text}
    } \partCombine \musicUp \musicDown
  >>
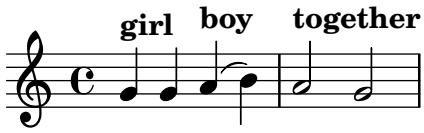
  \layout {
    indent = 6.0\cm
    \context {
      \Score
      % Setting this to a large value avoids a bar line at the
      % beginning that would connect the three staves otherwise.
      \override SystemStartBar.collapse-height = 30
    }
  }
}
```



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 6.3 [Writing parts], page 262.

Snippets: Section "Simultaneous notes" in *Snippets*.

Internals Reference: Section "PartCombineMusic" in *Internals Reference*, Section "Voice" in *Internals Reference*.

Known issues and warnings

All `\partCombine...` functions can only accept two voices.

`\partCombine...` functions cannot be placed inside a `\tuplet` or `\relative` block.

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

`\partCombine` only knows when a note starts in a `Voice`; it cannot, for example, remember if a note in one `Voice` has already started when combining notes that have just started in the other `Voice`. This can lead to a number of unexpected issues including "Solo" or "Unison" marks being printed incorrectly.

`\partCombine` keeps all spanners (slurs, ties, hairpins, etc.) in the same `Voice` so that if any such spanners start or end in a different `Voice`, they may not be printed properly or at all.

If the `\partCombine` function cannot combine both music expressions (i.e., when both voices have different durations), it will give the voices, internally, its own custom names: `one` and `two` respectively. This means if there is any "switch" to a differently named `Voice` context, the events in that differently named `Voice` will be ignored.

Because `\partCombine` is a two-pass feature, care must be taken to not mix up the two phases. For example, this code

```
one = { e''2 \tag #'score f''
            \tag #'part fis'' g''1 }
two = { e''2 d'' g'1 }

\removeWithTag #'score \partCombine \one \two
```

fails because the first pass does not know that you are removing tagged music, so the information it records is inconsistent with the music when it comes time for the second pass. If you want to use `\partCombine` with filtering, unfolded repeats, or other transformations, you must transform first so that the music is in its final form before the first pass, for example,

```
...
\partCombine
  \removeWithTag #'score \one
  \removeWithTag #'score \two
```

Refer also to *Known issues and warnings* when using `\partCombine` with tablature in Section 12.1.3 [Default tablatures], page 433, and the *Note* in Section 2.4.1 [Automatic beams], page 102, when using automatic beaming.

5.2.6 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

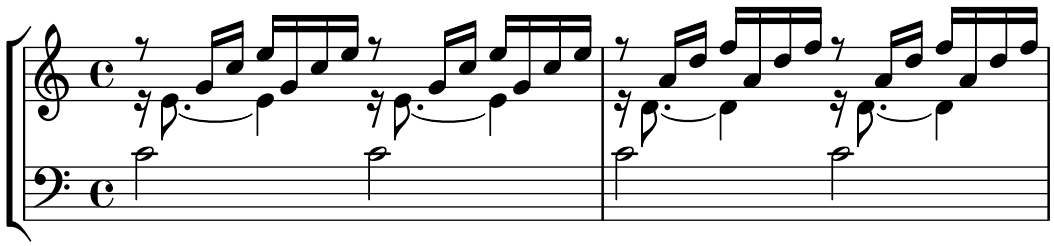
content of alternate measures from the expression become the value of the respective variables, so you can use them afterwards to print the music.

> **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.~   4             r16 e'8.~   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.~   4             r16 d'8.~   4               |
  c'2                       c'2                         |

}
\new StaffGroup <<
  \new Staff << \voiceA \\ \voiceB >>
  \new Staff { \clef bass \voiceC }
>>
```

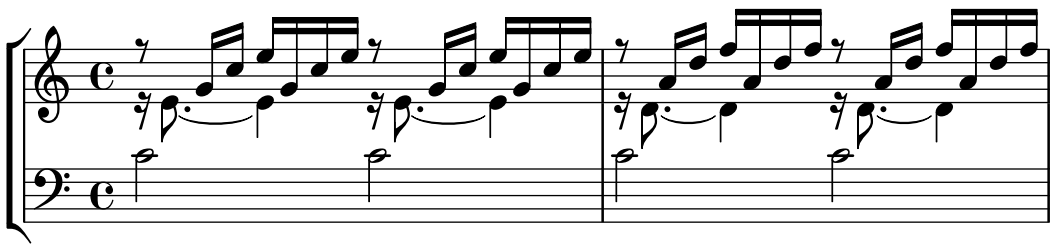


Relative mode may be used. Note that the \relative command is not used inside \parallelMusic itself. The notes are relative to the preceding note in the voice, not to the previous note in the input – in other words, relative notes for voiceA ignore the notes in voiceB.

```
\parallelMusic voiceA,voiceB,voiceC {
  % Bar 1
  r8 g16 c e g, c e r8 g,16 c e g, c e  |
  r16 e8.~ 4        r16 e8.~  4         |
  c2                c                   |

  % Bar 2
  r8 a,16 d f a, d f r8 a,16 d f a, d f |
  r16 d8.~  4        r16 d8.~  4        |
  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:
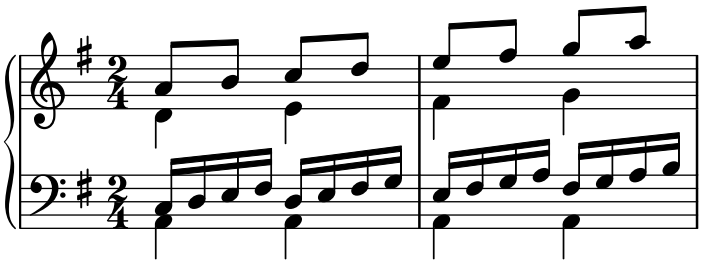
```
global = {
  \key g \major
  \time 2/4
}

\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
       >>
     }
  >>
}
```

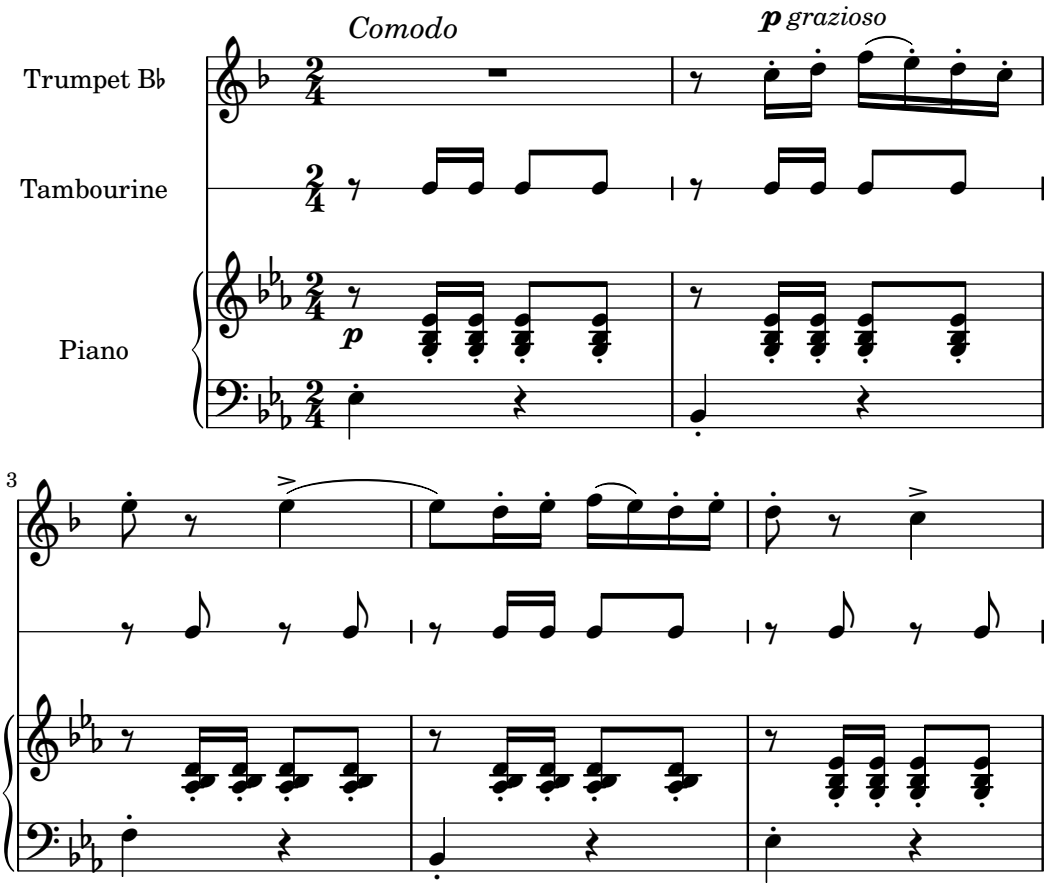


See also

Learning Manual: Section “Organizing pieces with variables” in *Learning Manual*.

Snippets: Section “Simultaneous notes” in *Snippets*.

6 Staff notation



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.

6.1 Displaying staves
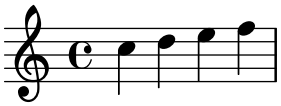
This section describes the different methods of creating and grouping staves.

6.1.1 Instantiating new staves

Staves (singular: *staff*) are created with the `\new` or `\context` commands. For details, see Section 33.2 [Creating and referencing contexts], page 739.

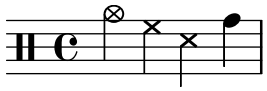
The basic staff context is `Staff`:

```
\new Staff \relative { c''4 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 Section 13.1.5 [Percussion staves], page 489.

```
\new DrumStaff {
  \drummode { cymc hh ss tomh }
}
```



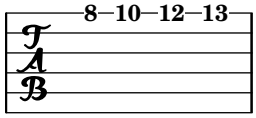
RhythmicStaff creates a single-line staff that only displays the rhythmic values of the input. Real durations are preserved. For details, see Section 2.3.8 [Showing melody rhythms], page 100.

```
\new RhythmicStaff { c4 d e f }
```



TabStaff creates a tablature with six strings in standard guitar tuning. For details, see Section 12.1.3 [Default tablatures], page 433.

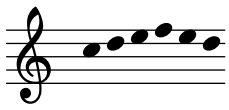
```
\new TabStaff \relative { c''4 d e f }
```



There are staff contexts specific for the notation of ancient music, for example, MensuralStaff and VaticanaStaff. They are described in Section 17.2.1 [Predefined contexts], page 540.

The GregorianTranscriptionStaff context creates a staff to notate modern Gregorian chant. It engraves *divisiones* as bar lines, but it does not show measure bar lines.

```
\new GregorianTranscriptionStaff \relative { c''4 d e f e d }
```



New single staff contexts may be defined. For details, see Section 33.6 [Defining new contexts], page 751.

See also

Music Glossary: Section “staff” in *Music Glossary*, Section “staves” in *Music Glossary*.

Notation Reference: Section 33.2 [Creating and referencing contexts], page 739, Section 13.1.5 [Percussion staves], page 489, Section 2.3.8 [Showing melody rhythms], page 100, Section 12.1.3 [Default tablatures], page 433, Section 17.2.1 [Predefined contexts], page 540, Section 6.2.1 [Staff symbol], page 250, Section 17.4.1 [Gregorian chant contexts], page 555, Section 17.3.1 [Mensural contexts], page 542, Section 33.6 [Defining new contexts], page 751.

Snippets: Section “Staff notation” in *Snippets*.

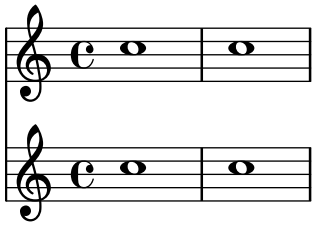
Internals Reference: Section “Staff” in *Internals Reference*, Section “DrumStaff” in *Internals Reference*, Section “GregorianTranscriptionStaff” in *Internals Reference*, Section “RhythmicStaff” in *Internals Reference*, Section “TabStaff” in *Internals Reference*, Section “MensuralStaff” in *Internals Reference*, Section “VaticanaStaff” in *Internals Reference*, Section “StaffSymbol” in *Internals Reference*.

6.1.2 Grouping staves

Various contexts exist to group single staves together in order to form multi-staff systems. Each grouping context sets the style of the system start delimiter and the behavior of bar lines.

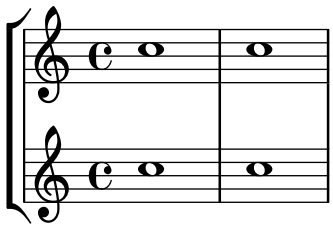
If no context is specified, the default properties will be used: the group is started with a vertical line, and the bar lines are not connected.

```
<<
  \new Staff \relative { c''1 c }
  \new Staff \relative { c''1 c }
>>
```



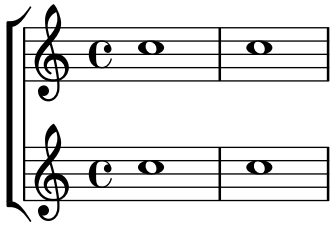
In the StaffGroup context, the group is started with a bracket and bar lines are drawn through all the staves.

```
\new StaffGroup <<
  \new Staff \relative { c''1 c }
  \new Staff \relative { c''1 c }
>>
```



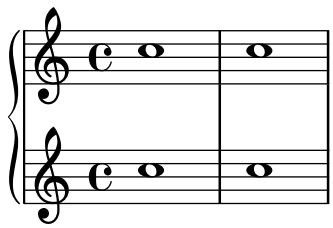
In a ChoirStaff, the group starts with a bracket, but bar lines are not connected.

```
\new ChoirStaff <<
  \new Staff \relative { c''1 c }
  \new Staff \relative { c''1 c }
>>
```



In a GrandStaff, the group begins with a brace, and bar lines are connected between the staves.

```
\new GrandStaff <<
  \new Staff \relative { c''1 c }
  \new Staff \relative { c''1 c }
>>
```
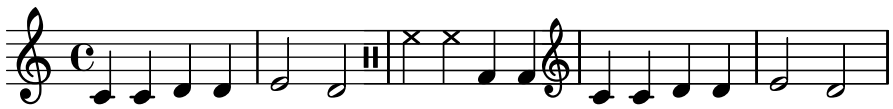


The PianoStaff is identical to a GrandStaff, except that its staves are only removed together, never separately; see Section 6.2.3 [Hiding staves], page 257.

The OneStaff is a staff group that places separate contexts in the same vertical alignment. This example shows three staves sharing the same space. Here, the Time_signature_engraver has been moved from the staves to the OneStaff context to prevent it from being repeated for each staff.

```
\layout {
  \context {
    \OneStaff
    \consists Time_signature_engraver
  }
  \context {
    \Staff
    \remove Time_signature_engraver
  }
  \context {
    \DrumStaff
    \remove Time_signature_engraver
  }
}

\new OneStaff {
  \new Staff {
    c'4 4 d'4 4 e'2 d'
  }
  \drums {
    hihat4 hh bassdrum bd
  }
  \new Staff {
    c'4 4 d'4 4 e'2 d'
  }
}
```



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.

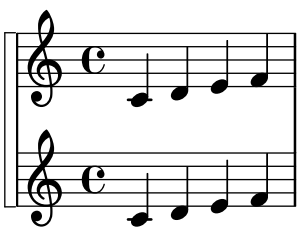
New staff group contexts may be defined. For details, see Section 33.6 [Defining new contexts], page 751.

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 `ChoirStaff` context.

```
\score {
  \new StaffGroup { <<
  \set StaffGroup.systemStartDelimiter = #'SystemStartSquare
    \new Staff { c'4 d' e' f' }
    \new Staff { c'4 d' e' f' }
  >> }
}
```

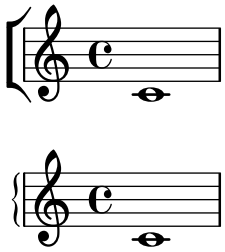


Display bracket with only one staff in a system

If there is only one staff in a `ChoirStaff` or `StaffGroup` context, the bracket and the starting bar line will not be displayed by default. This can be changed by setting the `collapse-height` property to a value less than the number of staff lines in the staff.

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.

```
\score {
  \new StaffGroup <<
    % Must be lower than the actual number of staff lines
    \override StaffGroup.SystemStartBracket.collapse-height = 4
    \override Score.SystemStartBar.collapse-height = 4
    \new Staff {
      c'1
    }
  >>
}
\score {
  \new PianoStaff <<
    \override PianoStaff.SystemStartBrace.collapse-height = 4
    \override Score.SystemStartBar.collapse-height = 4
    \new Staff {
      c'1
    }
  >>
}
```



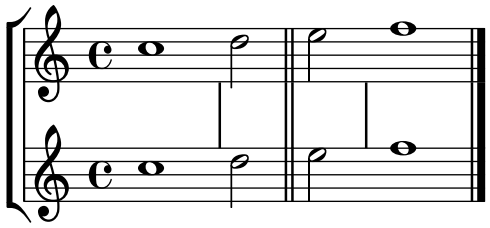
Mensurstriche layout (bar lines between the staves)

Mensurstriche, bar lines between but not through staves, can be printed by setting `measureBarType` to `"-span|"` and using a grouping context that allows span bars, such as `StaffGroup`.

```
\layout {
  \context {
    \Staff
    measureBarType = "-span|"
  }
}

music = \fixed c'' {
  c1
  d2 \section e2
  f1 \fine
}

\new StaffGroup <<
  \new Staff \music
  \new Staff \music
>>
```



See also

Music Glossary: Section “brace” in *Music Glossary*, Section “bracket” in *Music Glossary*, Section “grand staff” in *Music Glossary*, Section “mensurstrich” in *Music Glossary*.

Notation Reference: Section 6.3.1 [Instrument names], page 262, Section 33.6 [Defining new contexts], page 751.
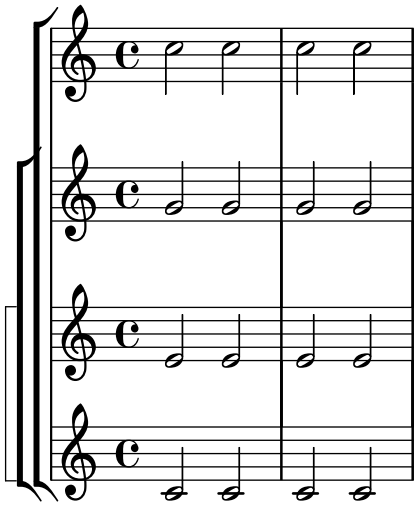
Snippets: Section “Staff notation” in *Snippets*.

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 “OneStaff” in *Internals Reference*, Section “SystemStartBar” in *Internals Reference*, Section “SystemStartBrace” in *Internals Reference*, Section “SystemStartBracket” in *Internals Reference*, Section “SystemStartSquare” in *Internals Reference*.

6.1.3 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 \relative { c''2 c | c2 c }
  \new StaffGroup <<
    \new Staff \relative { g'2 g | g2 g }
    \new StaffGroup \with {
      systemStartDelimiter = #'SystemStartSquare
    }
    <<
      \new Staff \relative { e'2 e | e2 e }
      \new Staff \relative { c'2 c | c2 c }
    >>
  >>
>>
```



New nested staff group contexts can be defined. For details, see Section 33.6 [Defining new contexts], page 751.

Selected snippets

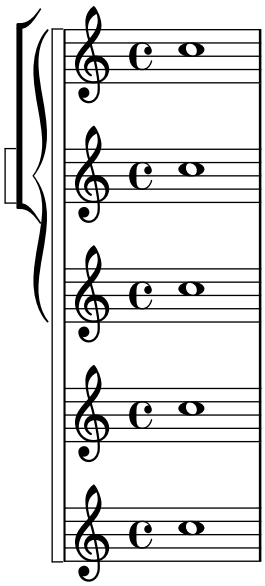
Nesting staves

The property systemStartDelimiterHierarchy can be used to make more complex nested staff groups. The systemStartDelimiterHierarchy property of the StaffGroup context 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'' <<
  \override StaffGroup.SystemStartSquare.collapse-height = 4
  \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: Section 6.1.2 [Grouping staves], page 244, Section 6.3.1 [Instrument names], page 262, Section 33.6 [Defining new contexts], page 751.

Snippets: Section "Staff notation" in *Snippets*.

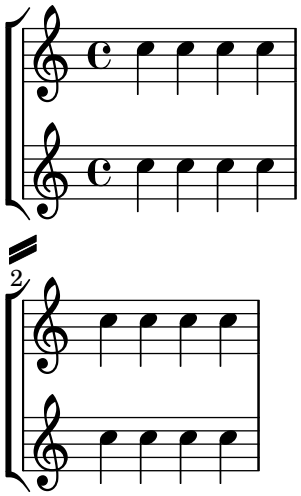
Internals Reference: Section "StaffGroup" in *Internals Reference*, Section "ChoirStaff" in *Internals Reference*, Section "SystemStartBar" in *Internals Reference*, Section "SystemStartBrace" in *Internals Reference*, Section "SystemStartBracket" in *Internals Reference*, Section "SystemStartSquare" in *Internals Reference*.

6.1.4 Separating systems

If the number of systems per page changes from page to page it is customary to separate the systems by placing a system separator mark between them. By default the system separator is blank, but can be turned on with a \paper option.

```
\book {
  \score {
    \new StaffGroup <<
      \new Staff {
        \relative {
          c''4 c c c
          \break
          c4 c c c
        }
```

```
      }
      \new Staff {
        \relative {
          c''4 c c c
          \break
          c4 c c c
        }
      }
    >>
  }
  \paper {
    system-separator-markup = \slashSeparator
    tagline = ##f
  }
}
```



See also

Notation Reference: Chapter 26 [Page layout], page 671.

Snippets: Section “Staff notation” in *Snippets*.
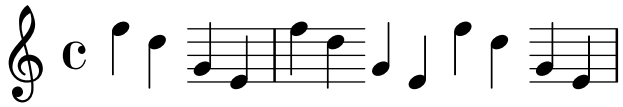
6.2 Modifying single staves

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.

6.2.1 Staff symbol

The `\stopStaff` and `\startStaff` commands can be used to stop or (re)start the staff lines respectively, from being printed at any point within a score.

```
\relative {
  \stopStaff f''4 d \startStaff g, e
  f'4 d \stopStaff g, e
  f'4 d \startStaff g, e
}
```



Predefined commands
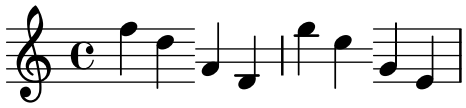
`\startStaff`, `\stopStaff`.

The lines of a staff belong to the `StaffSymbol` grob (including ledger lines) and can be modified using `StaffSymbol` properties, but these modifications must be made before the staff is (re)started.

The number of staff lines can be altered:
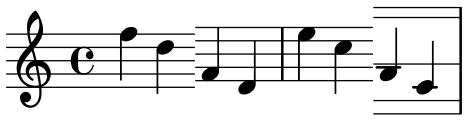
```
\relative {
  f''4 d \stopStaff
  \override Staff.StaffSymbol.line-count = 2
  \startStaff g, e |

  f'4 d \stopStaff
  \revert Staff.StaffSymbol.line-count
  \startStaff g, e |
}
```



The position of each staff line can also be altered. A list of numbers sets each line's position. `0` corresponds to the normal center line, and the normal line positions are `(-4 -2 0 2 4)`. A single staff line is printed for every value entered so that the number of staff lines, as well as their position, can be changed with a single override (thus, the `line-count` property is disregarded if `line-positions` is set).
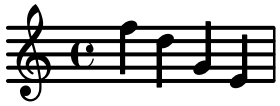
```
\relative {
  f''4 d \stopStaff
  \override Staff.StaffSymbol.line-positions = #'(1 3 5 -1 -3)
  \startStaff g, e |
  f'4 d \stopStaff
  \override Staff.StaffSymbol.line-positions = #'(8 6.5 -6 -8 -0.5)
  \startStaff g, e |
}
```



To preserve typical stem directions (in the bottom half of the staff stems point up, in the top half they point down), align the center line (or space) of the customized staff with the position of the normal center line (0). The clef position and the position of middle C may need to be adjusted accordingly to fit the new lines. See Section 1.3.1 [Clef], page 19.
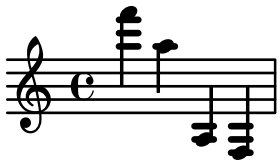
Staff line thickness can be altered. Ledger lines and note stems, by default, are also affected.

```
\new Staff \with {
  \override StaffSymbol.thickness = 3
} \relative {
  f''4 d g, e
}
```



It is also possible to set ledger line thickness independently of staff lines.

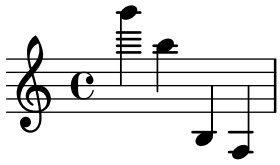
```
\new Staff \with {
  \override StaffSymbol.thickness = 2
  \override StaffSymbol.ledger-line-thickness = #'(0.5 . 0.4)
} \relative {
  f'''4 a, a,, f
}
```



The first value is multiplied by the staff line thickness, the second by the staff space and then the two values are added together to give the new thickness of the ledger line.
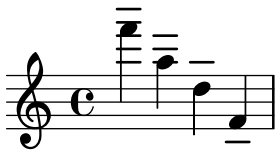
The vertical positions of ledger lines can be altered,

```
\new Staff \with {
  \override StaffSymbol.ledger-positions = #'(-3 -2 -1 2 5 6)
} \relative {
  f'''4 a, a,, f
}
```



Additional ledger lines can be made to appear above or below note heads depending on the current position relative to other note heads that also have their own ledger lines.
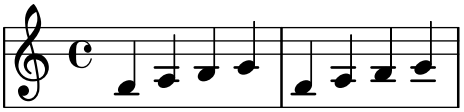
```
\new Staff \with {
  \override StaffSymbol.ledger-extra = 4
} \relative {
  f'''4 a, d, f,
}
```



Ledger lines can also be made to appear inside the staff where custom staff lines are required. The example shows the default position of ledger lines when the explicit `ledger-position` is and is not set. The `\stopStaff` is needed in the example to revert the `\override` for the whole `StaffSymbol`.

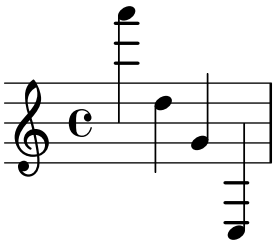
```
\relative d' {
  \override Staff.StaffSymbol.line-positions = #'(-8 0 2 4)
  d4 e f g
  \stopStaff
  \startStaff
  \override Staff.StaffSymbol.ledger-positions = #'(-8 -6 (-4 -2) 0)
  d4 e f g
```
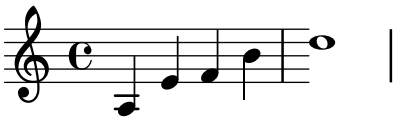
```
}
```



The distance between staff lines can be altered. This affects ledger line spacing as well.

```
\new Staff \with {
  \override StaffSymbol.staff-space = 1.5
} \relative {
  f'''4 d, g, e,
}
```



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
}
\relative { a4 e' f b | d1 }
```
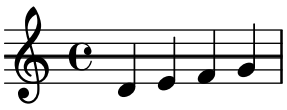


Selected snippets

Making some staff lines thicker than the others

For educational 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: Section 1.3.1 [Clef], page 19.
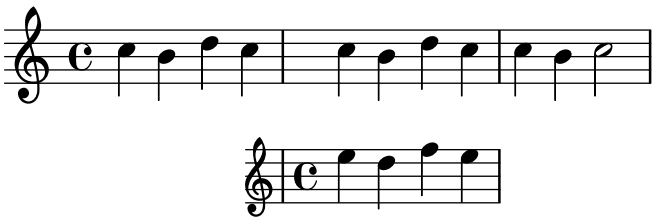
Snippets: Section “Staff notation” in *Snippets*.

Internals Reference: Section “StaffSymbol” in *Internals Reference*, Section “staff-symbol-interface” in *Internals Reference*.

6.2.2 Ossia staves

Ossia staves can be set by creating a new simultaneous staff in the appropriate location:

```
\new Staff \relative {
  c''4 b d c
  <<
    { c4 b d c }
    \new Staff { e4 d f e }
  >>
  c4 b c2
}
```

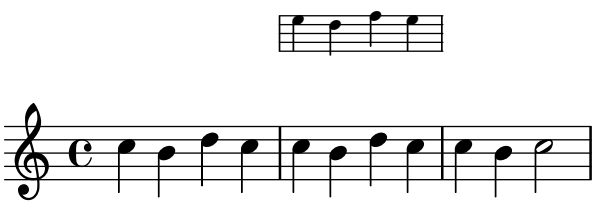


However, the above example is not what is usually desired. To create ossia staves that are above the original staff, have no time signature or clef, and have a smaller font size, tweaks must be used. The Learning Manual describes a specific technique to achieve this goal, beginning with Section “Nesting 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''4 b d c
  <<
    { c4 b d c }

    \new Staff \with {
      \remove Time_signature_engraver
      alignAboveContext = "main"
      \magnifyStaff #2/3
      firstClef = ##f
    }
    { e4 d f e }
  >>
  c4 b c2
}
```
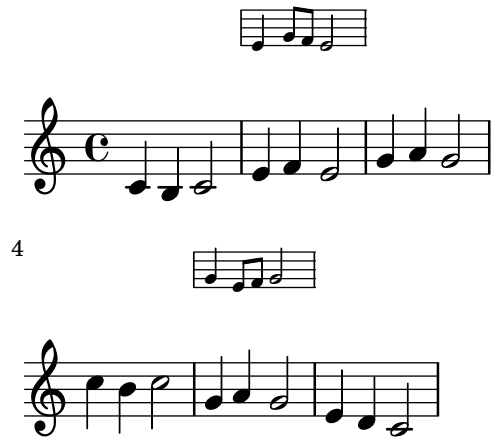


If many isolated ossia staves are needed, creating an empty `Staff` context with a specific *context id* may be more appropriate; the ossia staves may then be created by *calling* this context

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.
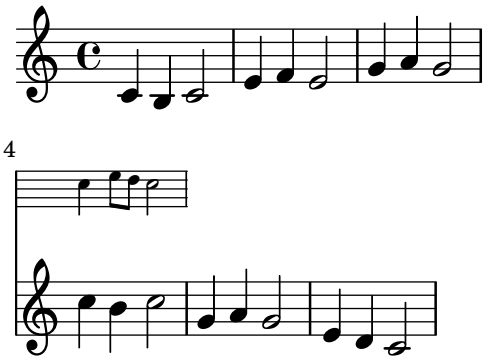
```
<<
  \new Staff = "ossia" \with {
    \remove Time_signature_engraver
    \hide Clef
    \magnifyStaff #2/3
  }
  { \stopStaff s1*6 }

  \new Staff \relative {
    c'4 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
  }
>>
```



Using the \RemoveAllEmptyStaves 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 \RemoveAllEmptyStaves, see Section 6.2.3 [Hiding staves], page 257.

```
<<
  \new Staff = "ossia" \with {
    \remove Time_signature_engraver
    \hide Clef
    \magnifyStaff #2/3
    \RemoveAllEmptyStaves
  } \relative {
    R1*3
    c''4 e8 d c2
  }
  \new Staff \relative {
    c'4 b c2
    e4 f e2
    g4 a g2 \break
    c4 b c2
    g4 a g2
    e4 d c2
  }
>>
```



Selected snippets

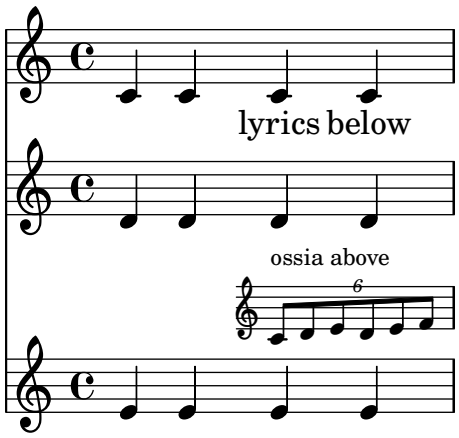
Vertically aligning ossias and lyrics

This snippet demonstrates the use of the context properties `alignBelowContext` and `alignAboveContext` to control the positioning of lyrics and ossias.

```
\relative c' <<
  \new Staff = "1" { c4 c c c }
  \new Staff = "2" { d4 d d d }
  \new Staff = "3" { e4 e e e }

  { \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"
        \override VerticalAxisGroup.staff-staff-spacing =
          #'((minimum-distance . 0)
             (basic-distance . 0)
             (padding . 1))
      } {
        \tuplet 6/4 {
          \override TextScript.padding = 2
          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: Section 6.2.3 [Hiding staves], page 257.

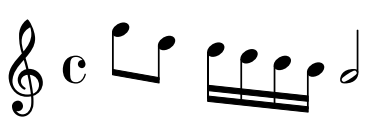
Snippets: Section “Staff notation” in *Snippets*.

Internals Reference: Section “StaffSymbol” in *Internals Reference*.

6.2.3 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 { a''8 f e16 d c b a2 }
```

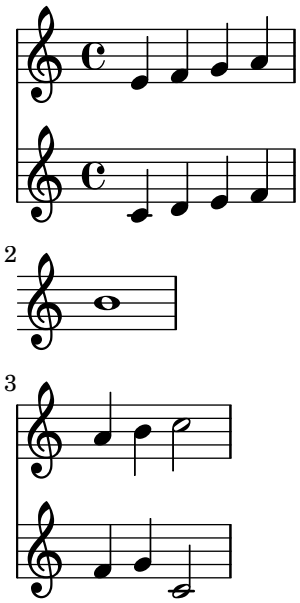


Empty staves can be hidden (for a so-called ‘Frenched Score’) by applying the `\RemoveEmptyStaves` command on a context, which can be done globally (in a `\layout` block)

as well as for specific staves only (in a \with block). This command removes all empty staves in a score except for those in the first system. If you want those in the first system to be hidden also, use \RemoveAllEmptyStaves.

```
\layout {
  \context {
    \Staff
    \RemoveEmptyStaves
  }
}

\relative <<
  \new Staff {
    e'4 f g a \break
    b1 \break
    a4 b c2
  }
  \new Staff {
    c,4 d e f \break
    R1 \break
    f4 g c,2
  }
>>
```



A staff is considered empty when it contains only multi-measure rests, rests, skips, or a combination of these elements. All *other* musical objects (that cause a staff not to be considered as empty) are listed in the keepAliveInterfaces context property, as initially set in the ly/engraver-init.ly file.
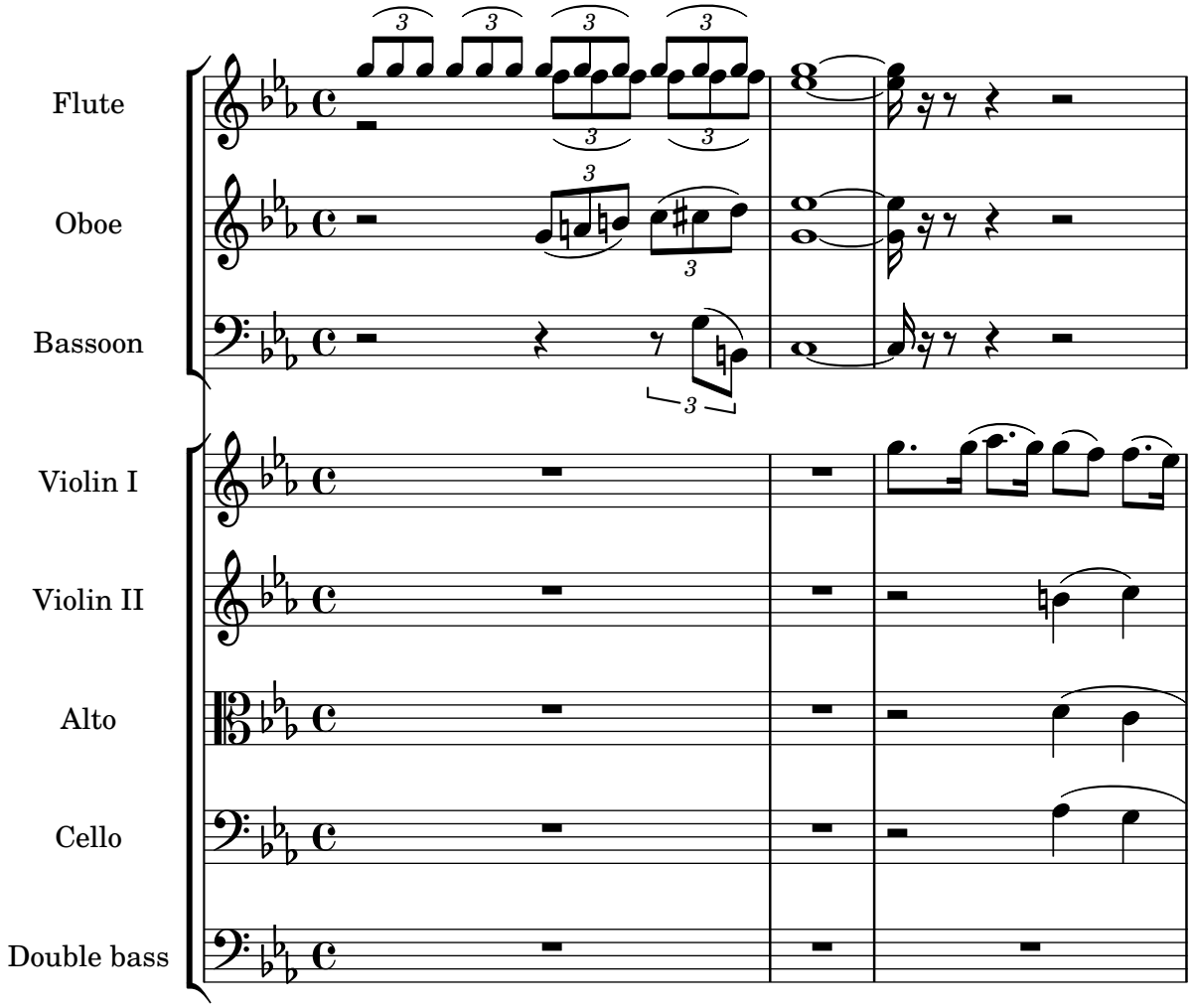
\RemoveEmptyStaves and \RemoveAllEmptyStaves are both predefined shortcuts that set such properties as remove-empty and remove-first for the VerticalAxisGroup object, as explained in Section B.22 [Context modification identifiers], page 964.
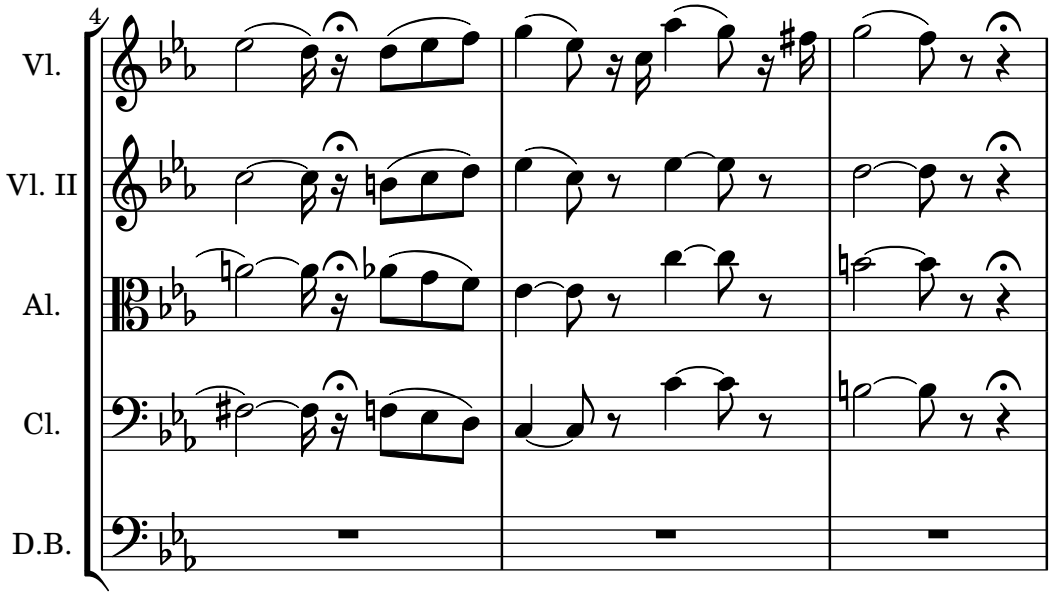
The Keep_alive_together_engraver allows groups of staves to only be removed together and not individually. By default, it is part of the PianoStaff context: a piano part will only be hidden when both of its staves are empty. Similarly, a common engraving practice in orchestral scores is to remove empty groups of staves rather than individual staves; that can be

achieved by adding the `Keep_alive_together_engraver` to the relevant staff grouping context, as explained Section 33.4 [Modifying context plug-ins], page 745, (see Section 6.1.2 [Grouping staves], page 244, for the context names).

```
\layout {
  \context {
    \StaffGroup
    \RemoveEmptyStaves
    \consists Keep_alive_together_engraver
  }
}
```

In the following example, staves devoted to wind instruments are removed in the second system; however, the double bass is not, because it is part of the string section, which is playing.





The `Keep_alive_together_engraver` internally uses the `remove-layer` property of a staff's `VerticalAxisGroup` to decide whether to print it or not when it is considered empty. That property may also be set directly, in which case it acts as a priority index: values closest to zero take precedence over higher numbers, and thus staves whose `remove-layer` is higher will be masked in favor of staves of a lower number.

This is particularly useful for 'divisi' staves, where some individual parts (see Section 6.3 [Writing parts], page 262) occasionally need to be expanded to more than one staff. In the following example, two parts are routed to *three* staves; however, all three staves are never printed at the same time:

- in the first systems, only a single one of them is shown, as the `keepAliveInterfaces` property has been set to an empty list – therefore the other two staves are considered empty and thus hidden, regardless of what they may contain;
- when that property gets unset (and thus reverts to its default setting), it is no longer preventing the two other staves from being printed; however, as their `remove-layer` setting is lower than the single staff's, these two staves are now printed in its place.
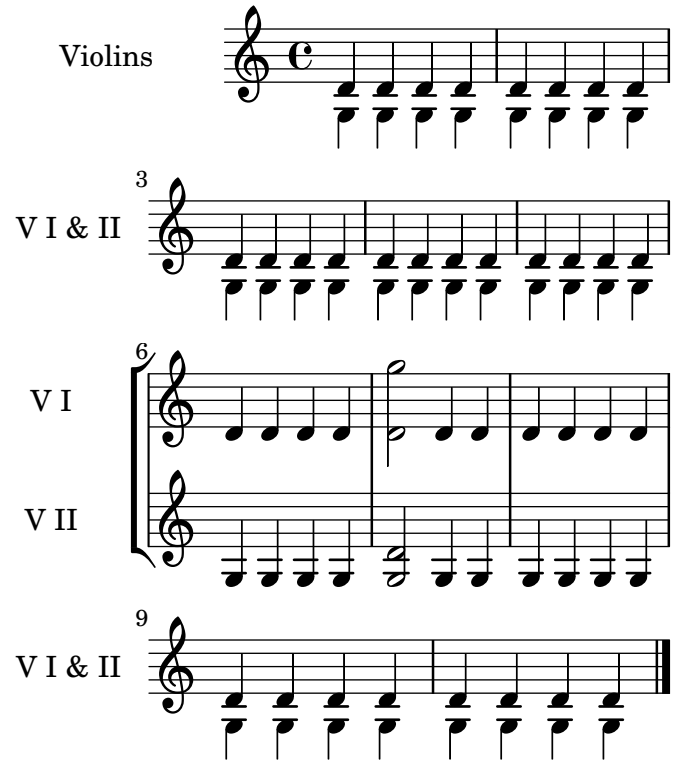
Such substitutions are applied not just to notes, chords and other musical events that occur immediately after the new setting, but to the whole system where it takes place.

```
\layout {
  short-indent = 2\cm
  indent = 3\cm
  \context {
    \Staff
    keepAliveInterfaces = #'()
  }
}

violI = {
  \*24 d'4
  \once \unset Staff.keepAliveInterfaces
  <d' g''>2
  \*14 d'4
  \bar "|."
}
```

```
violII = {
  \*24 g4
  <g d'>2
  \*14 g4
  \bar "|."
}

\new StaffGroup \with { \consists Keep_alive_together_engraver } <<
  \new Staff \with {
    instrumentName = "Violins"
    shortInstrumentName = "V I & II"
    \override VerticalAxisGroup.remove-layer = 2
  } <<  \violI \\ \violII  >>
  \new Staff \with {
    instrumentName = "Violin I"
    shortInstrumentName = "V I"
    \RemoveAllEmptyStaves
    \override VerticalAxisGroup.remove-layer = 1
  } \violI
  \new Staff \with {
    instrumentName = "Violin II"
    shortInstrumentName = "V II"
    \RemoveAllEmptyStaves
    \override VerticalAxisGroup.remove-layer = 1
  } \violII
>>
```



\RemoveAllEmptyStaves can also be used to create ossia sections for a staff. For details, see Section 6.2.2 [Ossia staves], page 254.

Predefined commands

`\RemoveEmptyStaves`, `\RemoveAllEmptyStaves`.

See also

Music Glossary: Section "Frenched staff" in *Music Glossary*.

Learning Manual: Section "Visibility and color of objects" in *Learning Manual*.

Notation Reference: Section 33.5 [Changing context default settings], page 746, Section 6.2.1 [Staff symbol], page 250, Section 6.2.2 [Ossia staves], page 254, Section 7.1.4 [Hidden notes], page 289, Section 2.2.2 [Invisible rests], page 69, Section 36.7 [Visibility of objects], page 784, Section B.22 [Context modification identifiers], page 964, Section 6.1.2 [Grouping staves], page 244, Section 33.4 [Modifying context plug-ins], page 745.

Installed Files: `ly/engraver-init.ly`.

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*, Section "Axis_group_engraver" in *Internals Reference*, Section "Keep_alive_together_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:

```
\omit StaffSymbol
\override NoteHead.no-ledgers = ##t
```

For the Known issues and warnings associated with `\RemoveEmptyStaves` see Section 33.5 [Changing context default settings], page 746.

6.3 Writing parts
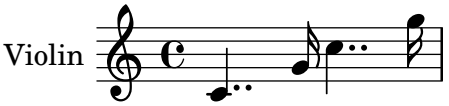
This section explains how to prepare parts for orchestral or ensemble music, which often requires to insert instrument names into the score. Methods to quote other voices and to format cue notes are also described, as well as a way to contract multiple consecutive empty measures in individual parts.
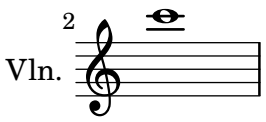
Additionally, a method for printing *divisi* staves, sometimes used in individual or desk parts, can be found in Section 6.2.3 [Hiding staves], page 257.

6.3.1 Instrument names

Instrument names can be printed on the left side of staves in the `Staff`, `PianoStaff`, `StaffGroup`, `GrandStaff` and `ChoirStaff` contexts. The value of `instrumentName` is used for the first staff, and the value of `shortInstrumentName` is used for all succeeding staves.
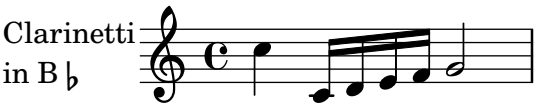
```
\new Staff \with {
  instrumentName = "Violin "
  shortInstrumentName = "Vln. "
} \relative {
  c'4.. g'16 c4.. g'16 \break | c1 |
}
```
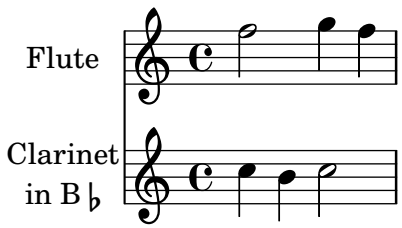




\markup can be used to create more complex instrument names:

```
\new Staff \with {
  instrumentName = \markup {
    \column { "Clarinetti"
      \line { "in B" \smaller \flat }
    }
  }
} \relative {
  c''4 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:

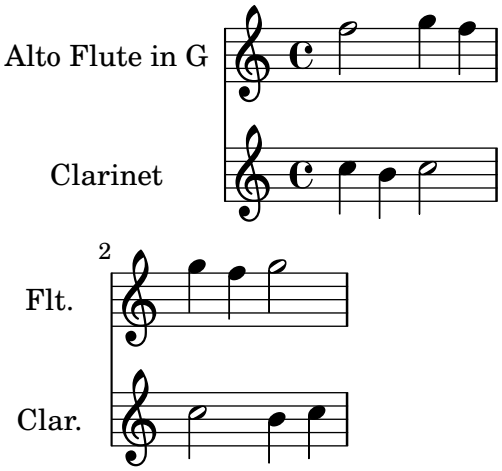
```
<<
  \new Staff \with {
    instrumentName = "Flute"
  } \relative {
    f''2 g4 f
}
  \new Staff \with {
    instrumentName = \markup {
      \center-column { "Clarinet"
        \line { "in B" \smaller \flat }
      }
    }
  } \relative { c''4 b c2 }
>>
```



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 Section 26.5.3 [\paper variables for shifts and indents], page 678.

```
<<
  \new Staff \with {
    instrumentName = "Alto Flute in G"
    shortInstrumentName = "Flt."
  } \relative {
    f''2 g4 f \break
    g4 f g2
  }
  \new Staff \with {
    instrumentName = "Clarinet"
    shortInstrumentName = "Clar."
  } \relative {
    c''4 b c2 \break
    c2 b4 c
  }
>>

\layout {
  indent = 3.0\cm
  short-indent = 1.5\cm
}
```



To add instrument names to other contexts (such as ChordNames or FiguredBass), Instrument_name_engraver must be added to that context. For details, see Section 33.4 [Modifying context plug-ins], page 745.

The shortInstrumentName may be changed in the middle of a piece, along with other settings as needed for the new instrument. However, only the first instance of instrumentName will be printed and subsequent changes will be ignored:

```
prepPiccolo = <>^\markup \italic { muta in Piccolo }

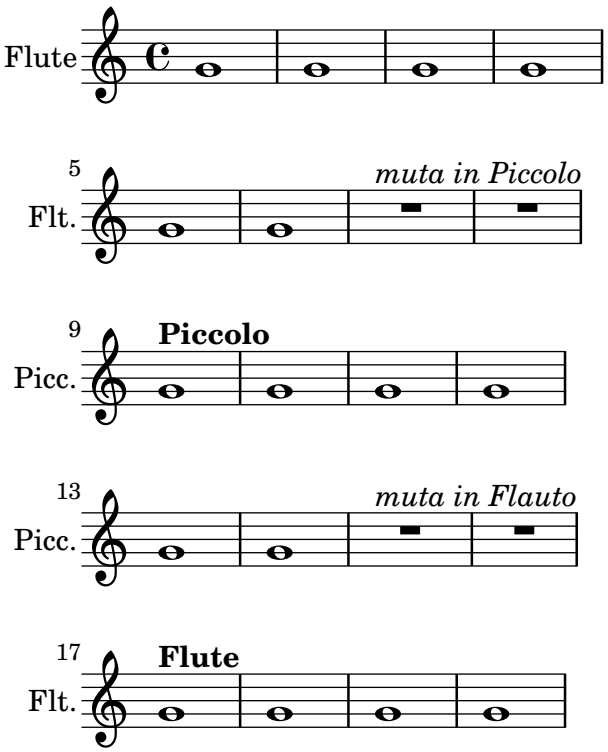
prepFlute = <>^\markup \italic { muta in Flauto }

setPiccolo = {
  <>^\markup \bold { Piccolo }
  \transposition c''
}

setFlute = {
```

```
  <>^\markup \bold { Flute }
  \transposition c'
}

\new Staff \with {
  instrumentName = "Flute"
  shortInstrumentName = "Flt."
}
\relative {
  g'1 g g g \break
  g1 g \prepPiccolo R R \break
  \set Staff.instrumentName = "Piccolo"
  \set Staff.shortInstrumentName = "Picc."
  \setPiccolo
  g1 g g g \break
  g1 g \prepFlute R R \break
  \set Staff.instrumentName = "Flute"
  \set Staff.shortInstrumentName = "Flt."
  \setFlute
  g1 g g g
}
```



See also

Notation Reference: Section 26.5.3 [\paper variables for shifts and indents], page 678, Section 33.4 [Modifying context plug-ins], page 745.

Snippets: Section "Staff notation" in *Snippets*.

Internals Reference: Section "InstrumentName" in *Internals Reference*, Section "PianoStaff" in *Internals Reference*, Section "Staff" in *Internals Reference*.

6.3.2 Quoting other voices

It is very common for one voice to use the same notes as those from another voice. For example, first and second violins playing the same phrase during a particular passage of the music. This is done by letting one voice *quote* the other, without having to reenter the music all over again for the second voice.

The `\addQuote` command, used in the top level scope, defines a stream of music from which fragments can be quoted.
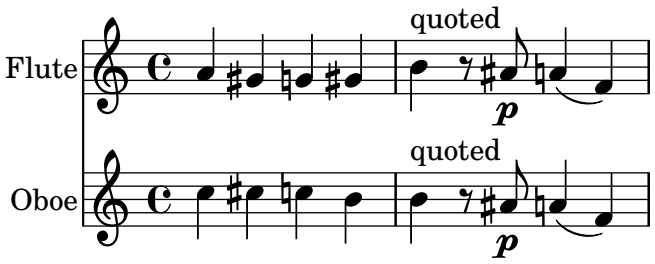
The `\quoteDuring` command is used to indicate the point where the quotation begins. It is followed by two arguments: the name of the quoted voice, as defined with `\addQuote`, and a music expression for the duration of the quote.

```
fluteNotes = \relative {
  a'4 gis g gis | b4^"quoted" r8 ais\p a4( f)
}

oboeNotes = \relative {
  c''4 cis c b \quoteDuring "flute" { s1 }
}

\addQuote "flute" { \fluteNotes }

\score {
  <<
    \new Staff \with { instrumentName = "Flute" } \fluteNotes
    \new Staff \with { instrumentName = "Oboe" } \oboeNotes
  >>
}
```



If the music expression used in `\quoteDuring` contains notes instead of spacer or multi-measure rests then the quote will appear as polyphony and may produce unexpected results.
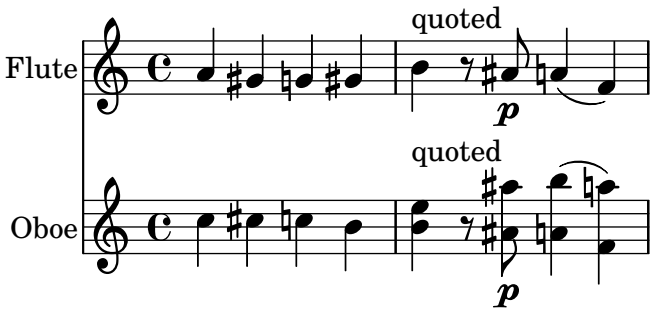
```
fluteNotes = \relative {
  a'4 gis g gis | b4^"quoted" r8 ais\p a4( f)
}

oboeNotes = \relative {
  c''4 cis c b \quoteDuring "flute" { e4 r8 ais b4 a }
}

\addQuote "flute" { \fluteNotes }

\score {
  <<
    \new Staff \with { instrumentName = "Flute" } \fluteNotes
    \new Staff \with { instrumentName = "Oboe" } \oboeNotes
```

```
  >>
}
```



If an \unfoldRepeats command in a music expression is required to be printed when using \quoteDuring, then it too must also contain its own \unfoldRepeats command;

```
fluteNotes = \relative {
  \repeat volta 2 { a'4 gis g gis }
}

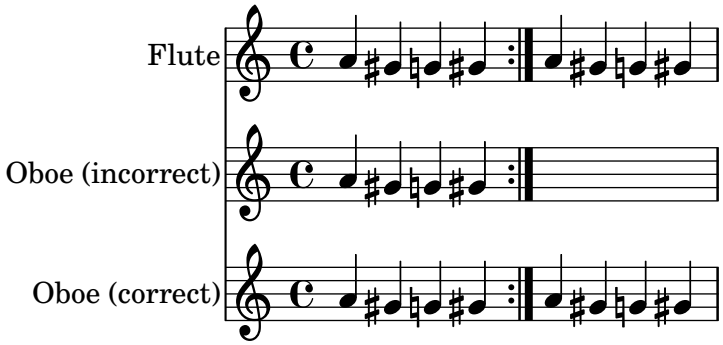
oboeNotesDW = \relative {
  \repeat volta 2 \quoteDuring "incorrect" { s1 }
}

oboeNotesW = \relative {
  \repeat volta 2 \quoteDuring "correct" { s1 }
}


\addQuote "incorrect" { \fluteNotes }

\addQuote "correct" { \unfoldRepeats \fluteNotes }

\score {
  \unfoldRepeats
  <<
    \new Staff \with { instrumentName = "Flute" }
    \fluteNotes
    \new Staff \with { instrumentName = "Oboe (incorrect)" }
    \oboeNotesDW
    \new Staff \with { instrumentName = "Oboe (correct)" }
    \oboeNotesW
  >>
}
```
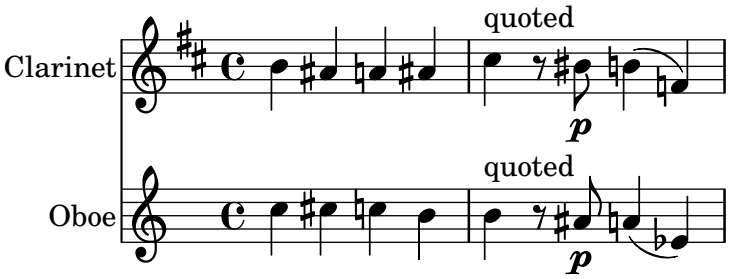


The \quoteDuring command uses the \transposition settings of both quoted and quoting parts to produce notes for the quoting part that have the same sounding pitch as those in the quoted part.

```
clarinetNotes = \relative c'' {
  \transposition bes
  \key d \major
  b4 ais a ais | cis4^"quoted" r8 bis\p b4( f)
}

oboeNotes = \relative {
  c''4 cis c b \quoteDuring "clarinet" { s1 }
}

\addQuote "clarinet" { \clarinetNotes }


\score {
  <<
    \new Staff \with { instrumentName = "Clarinet" } \clarinetNotes
    \new Staff \with { instrumentName = "Oboe" } \oboeNotes
  >>
}
```



By default quoted music will include all articulations, dynamics, markups, etc., in the quoted expression. It is possible to choose which of these objects from the quoted music are displayed by using the quotedEventTypes context property.
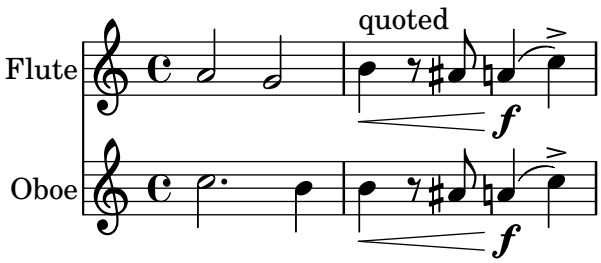
```
fluteNotes = \relative {
  a'2 g2 |
  b4\<^"quoted" r8 ais a4\f( c->)
 }

oboeNotes = \relative {
  c''2. b4 |
```

```
  \quoteDuring "flute" { s1 }
}

\addQuote "flute" { \fluteNotes }

\score {
  <<
    \set Score.quotedEventTypes = #'(note-event articulation-event
                                     crescendo-event rest-event
                                     slur-event dynamic-event)
    \new Staff \with { instrumentName = "Flute" } \fluteNotes
    \new Staff \with { instrumentName = "Oboe" } \oboeNotes
  >>
 }
```



Quotes can also be tagged, see Section 22.2.2 [Using tags], page 631.

See also

Notation Reference: Section 1.3.4 [Instrument transpositions], page 30, Section 22.2.2 [Using tags], page 631.

Installed Files: `scm/define-event-classes.scm`.

Snippets: Section “Staff notation” in *Snippets*.

Internals Reference: Section “Music classes” in *Internals Reference*, Section “QuoteMusic” in *Internals Reference*, Section “Voice” in *Internals Reference*.
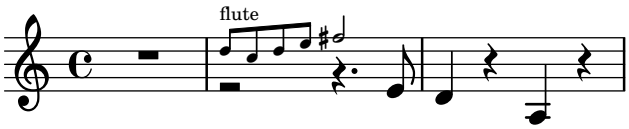
Known issues and warnings

Only the contents of the first `Voice` occurring in an `\addQuote` command will be considered for quotation, so if the music expression contains `\new` or `\context Voice` statements, their contents will not be quoted. Quoting nested triplets may result in poor notation.

6.3.3 Formatting cue notes
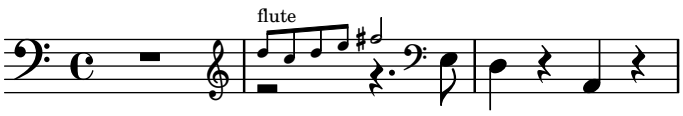
The simplest way to format cue notes is to explicitly create a `CueVoice` context within the part.

```
\relative {
  R1
  <<
    { e'2\rest r4. e8 }
    \new CueVoice {
      \stemUp d'8^"flute" c d e fis2
    }
  >>
  d,4 r a r
}
```
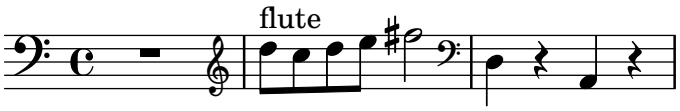


The \cueClef command can also be used with an explicit CueVoice context if a change of clef is required and will print an appropriately sized clef for the cue notes. The \cueClefUnset command can then be used to switch back to the original clef, again with an appropriately sized clef.

```
\relative {
  \clef "bass"
  R1
  <<
    { e'2\rest r4. \cueClefUnset e,8 }
    \new CueVoice {
      \cueClef "treble" \stemUp d''8^"flute" c d e fis2
    }
  >>
  d,,4 r a r
}
```



The \cueClef and \cueClefUnset commands can also be used without a CueVoice if required.

```
\relative {
  \clef "bass"
  R1
  \cueClef "treble"
  d''8^"flute" c d e fis2
  \cueClefUnset
  d,,4 r a r
}
```



For more complex cue note placement like including transposition, or inserting cue notes from multiple music sources, the \cueDuring or \cueDuringWithClef commands can be used. These are more specialized forms of \quoteDuring, see Section 6.3.2 [Quoting other voices], page 266, in the previous section.

The syntax is

\cueDuring *quotename direction music*

and

\cueDuringWithClef *quotename direction clef music*
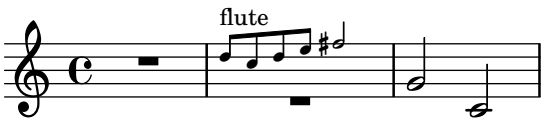
The music from the corresponding measures of *quotename* is added as a CueVoice context and occurs simultaneously with *music*, which then creates a polyphonic situation. The *direction* variable takes the argument #UP or #DOWN, and corresponds to the first and second voice, respectively, determining how the cue notes are printed in relation to the other voice.

```
fluteNotes = \relative {
  r2. c''4 | d8 c d e fis2 | g2 d |
}

oboeNotes = \relative c'' {
  R1
  <>^\markup \tiny { flute }
  \cueDuring "flute" #UP { R1 }
  g2 c,
}

\addQuote "flute" { \fluteNotes }

\new Staff {
  \oboeNotes
}
```
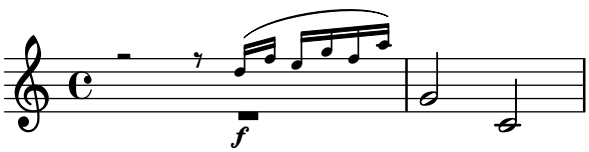


It is possible to adjust which aspects of the music are quoted with `\cueDuring` by setting the `quotedCueEventTypes` property. Its default value is `'(note-event rest-event tie-event beam-event tuplet-span-event)`, which means that only notes, rests, ties, beams and tuplets are quoted, but not articulations, dynamic marks, markup, etc.

> **Note:** When a `Voice` starts with `\cueDuring`, as in the following example, the `Voice` context must be explicitly declared, or else the entire music expression would belong to the `CueVoice` context.

```
oboeNotes = \relative {
  r2 r8 d''16(\f f e g f a)
  g8 g16 g g2.
}
\addQuote "oboe" { \oboeNotes }

\new Voice \relative c'' {
  \set Score.quotedCueEventTypes = #'(note-event rest-event tie-event
                                      beam-event tuplet-span-event
                                      dynamic-event slur-event)
  \cueDuring "oboe" #UP { R1 }
  g2 c,
}
```



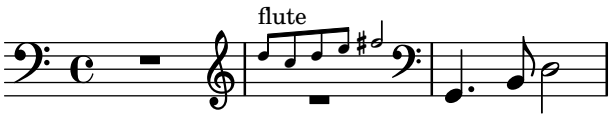
Markup can be used to show the name of the quoted instrument. If the cue notes require a change in clef, this can be done manually but the original clef should also be restored manually at the end of the cue notes.

```
fluteNotes = \relative {
  r2. c''4 d8 c d e fis2 g2 d2
}

bassoonNotes = \relative c {
  \clef bass
  R1
  \clef treble
  <>^\markup \tiny { flute }
  \cueDuring "flute" #UP { R1 }
  \clef bass
  g4. b8 d2
}

\addQuote "flute" { \fluteNotes }

\new Staff {
  \bassoonNotes
}
```
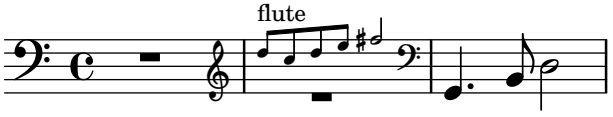


Alternatively, the \cueDuringWithClef function can be used instead. This command takes an extra argument to specify the change of clef that needs to be printed for the cue notes but will automatically print the original clef once the cue notes have finished.

```
fluteNotes = \relative {
  r2. c''4 d8 c d e fis2 g2 d2
}

bassoonNotes = \relative c {
  \clef bass
  R1
  <>^\markup { \tiny "flute" }
  \cueDuringWithClef "flute" #UP "treble" { R1 }
  g4. b8 d2
}

\addQuote "flute" { \fluteNotes }

\new Staff {
  \bassoonNotes
}
```



Like \quoteDuring, \cueDuring takes instrument transpositions into account. Cue notes are produced at the pitches that would be written for the instrument receiving the cue to produce the sounding pitches of the source instrument.

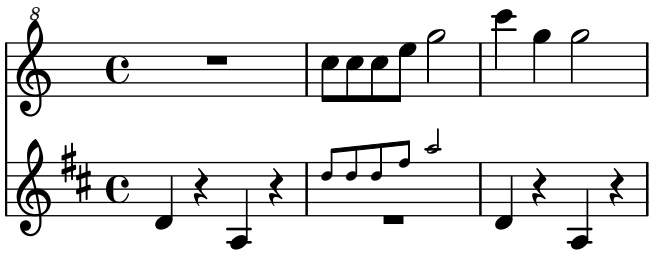
To transpose cue notes differently, use \transposedCueDuring. This command takes an extra argument to specify (in absolute mode) the printed pitch that you want to represent the sound of a concert middle C. This is useful for taking cues from an instrument in a completely different register.

```
piccoloNotes = \relative {
  \clef "treble^8"
  R1
  c'''8 c c e g2
  c4 g g2
}

bassClarinetNotes = \relative c' {
  \key d \major
  \transposition bes,
  d4 r a r
  \transposedCueDuring "piccolo" #UP d { R1 }
  d4 r a r
}

\addQuote "piccolo" { \piccoloNotes }

<<
  \new Staff \piccoloNotes
  \new Staff \bassClarinetNotes
>>
```



The \killCues command removes cue notes from a music expression, so the same music expression can be used to produce the instrument part with cues and the score. The \killCues command removes only the notes and events that were quoted by \cueDuring. Other markup associated with cues, such as clef changes and a label identifying the source instrument, can be tagged for selective inclusion in the score; see Section 22.2.2 [Using tags], page 631.

```
fluteNotes = \relative {
  r2. c''4 d8 c d e fis2 g2 d2
}
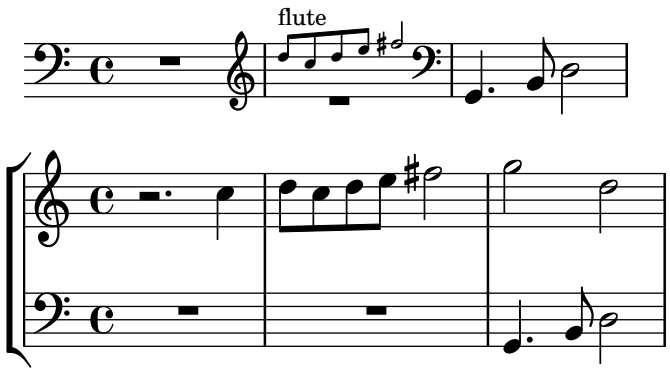
bassoonNotes = \relative c {
  \clef bass
  R1
  \tag #'part {
    \clef treble
    <>^\markup \tiny { flute }
  }
  \cueDuring "flute" #UP { R1 }
  \tag #'part \clef bass
```

```
  g4. b8 d2
}

\addQuote "flute" { \fluteNotes }

\new Staff {
  \bassoonNotes
}

\new StaffGroup <<
  \new Staff {
    \fluteNotes
  }
  \new Staff {
    \removeWithTag #'part { \killCues { \bassoonNotes } }
  }
>>
```



See also

Notation Reference: Section 6.3.2 [Quoting other voices], page 266, Section 1.3.4 [Instrument transpositions], page 30, Section 6.3.1 [Instrument names], page 262, Section 1.3.1 [Clef], page 19, Section 9.6.3 [Musical cues], page 395, Section 22.2.2 [Using tags], page 631.

Snippets: Section "Staff notation" in *Snippets*.

Internals Reference: Section "CueVoice" in *Internals Reference*, Section "Voice" in *Internals Reference*.

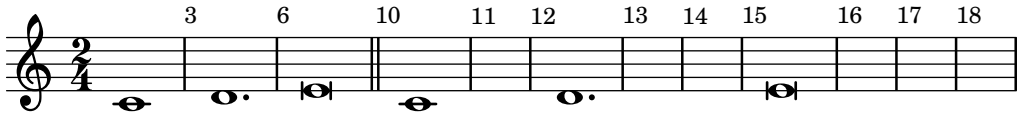
Known issues and warnings

Collisions can occur with rests, when using `\cueDuring`, between `Voice` and `CueVoice` contexts. When using `\cueDuringWithClef` or `\transposedCueDuring` the extra argument required for each case must come after the quote and the direction.

6.3.4 Compressing empty measures

By default, all measures are printed even if they are empty – which can happen if a rhythmic event (such as notes, rests or invisible rests) is so long as to span several measures. This behavior can be changed by contracting all empty measures into a single one, as illustrated here (the second part of this example, with expanded measures, actually reverts back to the default behavior):

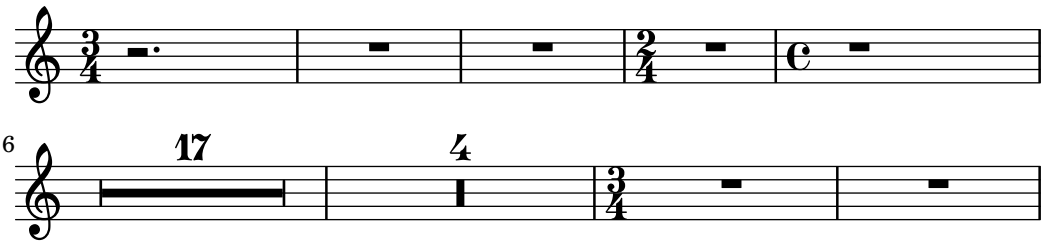
```
\override Score.BarNumber.break-visibility = ##(#f #t #t)
\time 2/4
\compressEmptyMeasures
c'1 d'1. e'\breve
\bar "||"
\expandEmptyMeasures
c'1 d'1. e'\breve
```



Although that notation is syntactically correct, it may be confusing from a musical point of view, as illustrated in the previous example; hence the need for measure numbers to be explicitly printed, using the syntax described in Section 36.7.4 [Using break-visibility], page 786.

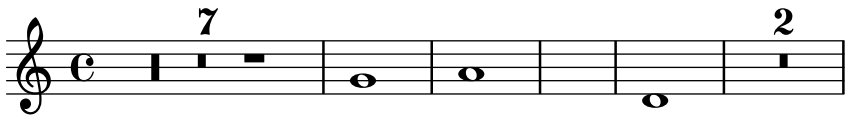
Where such a notation may become more useful is when applied to Section 2.2.3 [Full measure rests], page 71. A multi-measure rest will then be shown as a single measure containing a multi-measure rest symbol, with the number of measures of rest printed above the measure:

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



Unlike \compressEmptyMeasures, the music function \compressMMRests will only apply to rests, leaving any other events uncompressed. As a function rather than a property setting, its syntax differs slightly in that it must be followed by a music expression:

```
\compressMMRests {
  % Rests are compressed...
  R1*7
  % ... but notes can still span multiple measures.
  g'1 a'1*2 d'1
  R1*2
}
```



All of the commands described in this section actually rely on the skipBars internal property, which is set in the Score context, see Section 35.2 [\set and \unset], page 760.

Predefined commands

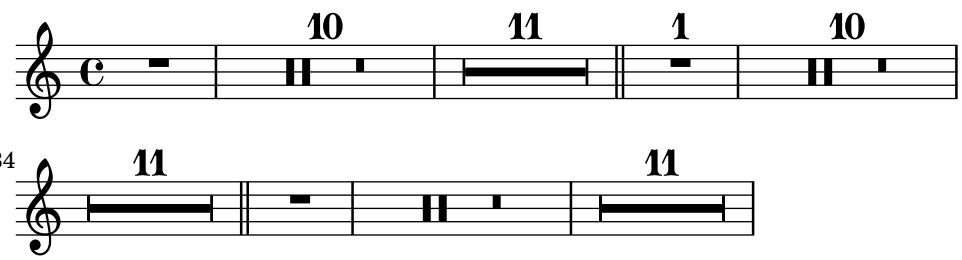
\compressEmptyMeasures, \expandEmptyMeasures, \compressMMRests.

Selected snippets

Numbering single measure rests

Multi-measure rests show their length by a number except for single measures. This can be changed by setting restNumberThreshold.

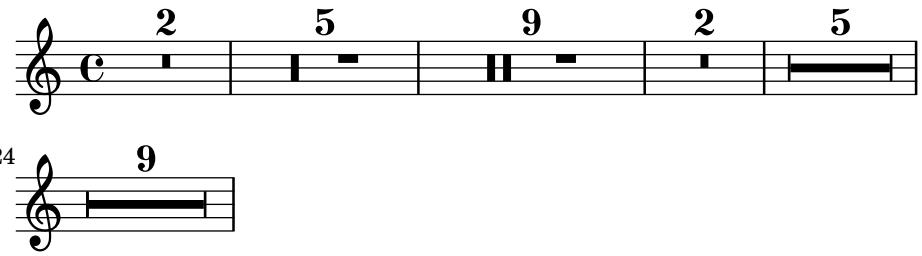
```
{
  \compressEmptyMeasures
  R1 R1*10 R1*11 \bar "||"
  \set restNumberThreshold = 0
  R1 R1*10 R1*11 \bar "||"
  \set restNumberThreshold = 10
  R1 R1*10 R1*11
}
```



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 long, thick horizontal line is shown. This default value of ten may be changed by overriding the expand-limit property.

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

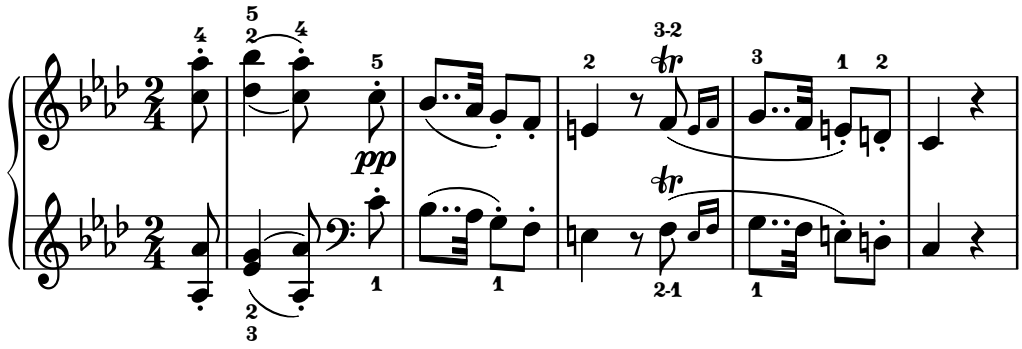


See also

Notation Reference: Section 36.7.4 [Using break-visibility], page 786, Section 2.2.3 [Full measure rests], page 71, Section 35.2 [\set and \unset], page 760.

Internals Reference: Section “MultiMeasureRest” in *Internals Reference*, Section “MultiMeasureRestNumber” in *Internals Reference*, Section “MultiMeasureRestScript” in *Internals Reference*, Section “MultiMeasureRestText” in *Internals Reference*.

7 Editorial annotations



This section discusses the various ways to change the appearance of notes and add analysis or educational emphasis.

7.1 Inside the staff
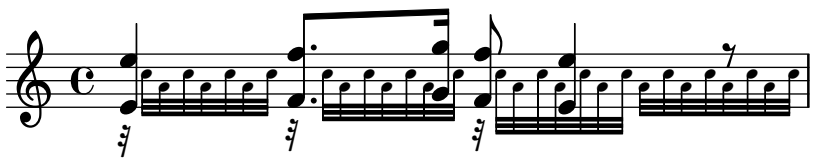
This section discusses how to add emphasis to elements that are inside the staff.

7.1.1 Selecting notation font size

> **Note:**
> For font sizes of text, see Section 8.2.2 [Selecting font and font size], page 324.
> For staff size, see Section 27.2 [Setting the staff size], page 685.
> For cue notes, see Section 6.3.3 [Formatting cue notes], page 269.
> For ossia staves, see Section 6.2.2 [Ossia staves], page 254.

To change the size of the notation without changing the staff size, specify a magnification factor with the `\magnifyMusic` command:

```
\new Staff <<
  \new Voice \relative {
    \voiceOne
    <e' e'>4 <f f'>8. <g g'>16 <f f'>8 <e e'>4 r8
  }
  \new Voice \relative {
    \voiceTwo
    \magnifyMusic 0.63 {
      \override Score.SpacingSpanner.spacing-increment = #(* 1.2 0.63)
      r32 c'' a c a c a c r c a c a c a c
      r c a c a c a c a c a c a c a c
    }
  }
>>
```

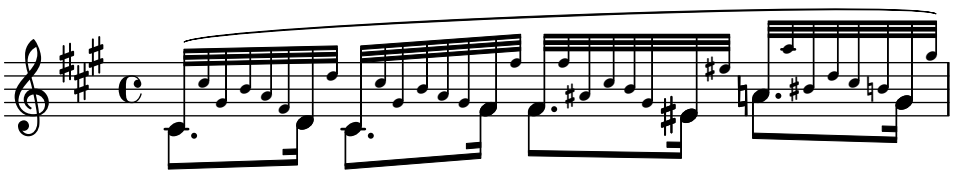


The \override in the example above is a bug workaround. See the "Known issues and warnings" at the end of this section.

If a normal sized note head is merged with a smaller one, the size of the smaller note may need to be reset (with \once \normalsize) so that the stems and accidentals align properly:

```
\new Staff <<
  \key fis \minor
  \mergeDifferentlyDottedOn
  \new Voice \relative {
    \voiceOne
    \magnifyMusic 0.63 {
      \override Score.SpacingSpanner.spacing-increment =
        #(* 1.2 0.63)

      \once \normalsize cis'32( cis' gis b a fis
        \once \normalsize d d'
      \once \normalsize cis, cis' gis b a gis
        \once \normalsize fis fis'
      \once \normalsize fis, fis' ais, cis b gis
        \once \normalsize eis eis'
      \once \normalsize a, a' bis, d cis b
        \once \normalsize gis gis')
    }
  }
  \new Voice \relative {
    \voiceTwo
    cis'8. d16 cis8. fis16 fis8. eis16 a8. gis16
  }
>>
```
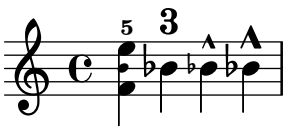


The \magnifyMusic command is not intended for cue notes, grace notes, or ossia staves—there are more appropriate methods of entering each of those constructs. Instead, it is useful when the notation size changes in a single instrumental part on one staff, and where grace notes are not appropriate, such as in cadenza-like passages or in cases such as the above examples. Setting the \magnifyMusic value to 0.63 duplicates the dimensions of the CueVoice context.

> **Note:** The \magnifyMusic command should *not* be used when also resizing the staff. See Section 27.2 [Setting the staff size], page 685.

Resizing individual layout objects

An individual layout object can be resized by using the \tweak or \override commands to adjust its font-size property:

```
\relative {
  % resize a note head
  <f' \tweak font-size -4 b e>-5
  % resize a fingering
  bes-\tweak font-size 0 -3
  % resize an accidental
  \once \override Accidental.font-size = -4 bes!-^
  % resize an articulation
  \once \override Script.font-size = 4 bes!-^
}
```
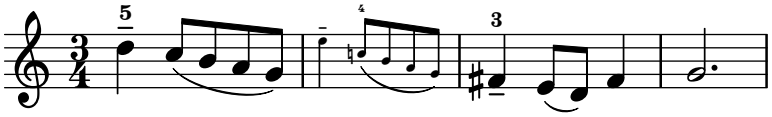


The default `font-size` value for each layout object is listed in the Internals Reference. The `font-size` property can only be set for layout objects that support the `font-interface` layout interface. If `font-size` is not specified in the object's 'Standard settings' list, its value is 0. See Section "All layout objects" in *Internals Reference*.

Understanding the `fontSize` property

The `fontSize` context property adjusts the relative size of all glyph-based notational elements in a context.[1]

```
\relative {
  \time 3/4
  d''4---5 c8( b a g) |
  \set fontSize = -6
  e'4-- c!8-4( b a g) |
  \set fontSize = 0
  fis4---3 e8( d) fis4 |
  g2.
}
```



The `fontSize` value is a number indicating the size relative to the standard size for the current staff height. The default `fontSize` is 0; adding 6 to any `fontSize` value doubles the printed size of the glyphs, and subtracting 6 halves the size. Each step increases the size by approximately 12%.

The Scheme function `magnification->font-size` is provided for convenience since the logarithmic units of the `font-size` property are not entirely intuitive. For example, to adjust the musical notation to 75% of the default size, use:

```
\set fontSize = #(magnification->font-size 0.75)
```

The Scheme function `magstep` does the opposite: it converts a `font-size` value into a magnification factor.

The `fontSize` property will only affect notational elements that are drawn with glyphs, such as note heads, accidentals, scripts, etc. It will not affect the size of the staff itself, nor will

[1] Note the words 'glyph-based' – a stem, for example, is not a glyph but directly constructed by LilyPond with lines and curves; consequently, it is not affected. The same holds for similar objects like slurs or beams.
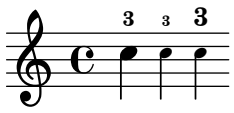
it scale stems, beams, or horizontal spacing. To scale stems, beams, and horizontal spacing along with the notation size (without changing the staff size), use the `\magnifyMusic` command discussed above. To scale everything, including the staff size, see Section 27.2 [Setting the staff size], page 685.

Whenever the `fontSize` *context property* is set, its value is added to the value of the `font-size` *grob property* for individual layout objects, before any glyphs are printed. This can cause confusion when setting individual `font-size` properties while `fontSize` is already set:

```
% the default font-size for NoteHead is 0
% the default font-size for Fingering is -5
c''4-3

\set fontSize = -3
% the effective font size for NoteHead is now -3
% the effective font size for Fingering is now -8
c''4-3
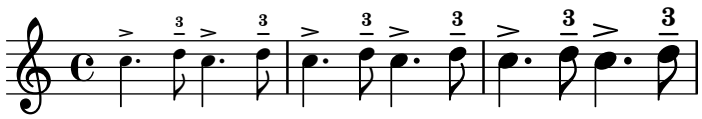
\override Fingering.font-size = 0
% the effective font size for Fingering is now -3
c''4-3
```



The following shorthand commands are also available:

Command	Equivalent to	Relative size
`\teeny`	`\set fontSize = -3`	71%
`\tiny`	`\set fontSize = -2`	79%
`\small`	`\set fontSize = -1`	89%
`\normalsize`	`\set fontSize = 0`	100%
`\large`	`\set fontSize = 1`	112%
`\huge`	`\set fontSize = 2`	126%

```
\relative c'' {
  \teeny
  c4.-> d8---3
  \tiny
  c4.-> d8---3
  \small
  c4.-> d8---3
  \normalsize
  c4.-> d8---3
  \large
  c4.-> d8---3
  \huge
  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 20 pt staff, an 11 pt font is selected.

Predefined commands

`\magnifyMusic`, `\teeny`, `\tiny`, `\small`, `\normalsize`, `\large`, `\huge`.

See also

Notation Reference: Section 8.2.2 [Selecting font and font size], page 324, Section 27.2 [Setting the staff size], page 685, Section 6.3.3 [Formatting cue notes], page 269, Section 6.2.2 [Ossia staves], page 254.

Installed Files: `ly/music-functions-init.ly`, `ly/property-init.ly`.

Snippets: Section "Editorial annotations" in *Snippets*.

Internals Reference: Section "font-interface" in *Internals Reference*.

Known issues and warnings

There are currently two bugs that are preventing proper horizontal spacing when using `\magnifyMusic`. There is only one available workaround, and it is not guaranteed to work in every case. In the example below, replace the *mag* variable with your own value. You may also try removing one or both of the `\newSpacingSection` commands, and/or the `\override` and `\revert` commands:

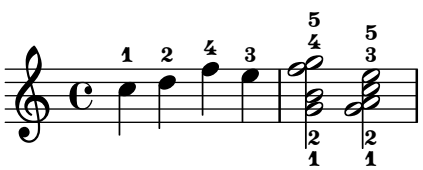
```
\magnifyMusic mag {
  \newSpacingSection
  \override Score.SpacingSpanner.spacing-increment = #(* 1.2 mag)
  [music]
  \newSpacingSection
  \revert Score.SpacingSpanner.spacing-increment
}
```

7.1.2 Fingering instructions

LilyPond provides two engravers for handling fingering instructions. In both cases, the fingering can be entered using the syntax '*note-digit*'.
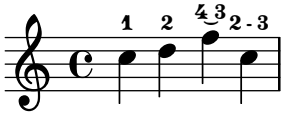
- The first engraver is `Fingering_engraver`, which takes care of fingering instructions that are outside of chord constructs (i.e., outside of `<...>`). The order of fingering given in the input code is directly reflected in the output, and the fingering markup is always stacked vertically within a single column above or below the note or chord.

```
\relative {
  c''4-1 d-2 f-4 e-3 |
  <g, b f' g>2_2_1^4^5 <g a c e>_2_1^3^5
}
```



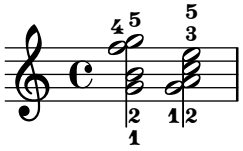
If you want markup texts or strings for fingering, use the `\finger` command instead.

```
\relative {
  c''4-1 d-2 f\finger \markup \tied-lyric "4~3" c\finger "2 - 3"
}
```
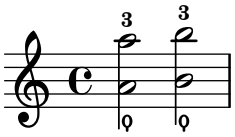


- The second engraver is `New_fingering_engraver`, which handles fingering instructions, articulations, and harmonic note heads inside of chords (i.e., inside of `<...>`).

```
\relative {
  <g'-1 b-2 f'-4 g-5>2 <e'-5 c-3 a-2 g-1>
}
```



A thumb fingering can also be added (e.g., for cello music) to indicate that a note should be played with the thumb.

```
\relative { <a'_\thumb a'-3>2 <b_\thumb b'-3> }
```



Fingering instructions may be manually placed above or below the staff, see Section 36.1 [Direction and placement], page 774.

See the next section for snippet examples that demonstrate how to control the positioning of fingering instructions.

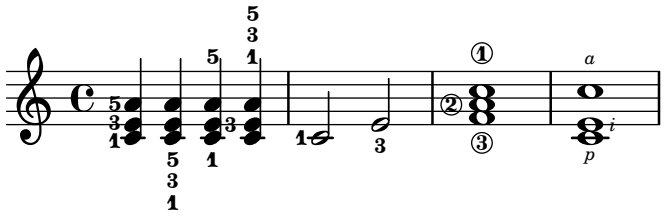
Selected snippets

Controlling the placement of chord fingerings

The placement of fingering numbers can be controlled precisely by using the property `fingeringOrientation`. For fingering orientation to apply, the fingering command must be used within a chord construct (`<...>`), even for single notes. Orientation for string numbers and right-hand fingerings may be controlled in a similar way by using the properties `stringNumberOrientation` and `strokeFingerOrientation`, respectively.

These properties can be set to a list of one to three values. They control whether fingerings may be placed above (if `up` appears in the list), below (if `down` appears), to the left (if `left` appears), or to the right (if `right` appears). Conversely, if a location is not listed, no fingering is placed there. LilyPond takes these constraints and works out the best placement for the fingering of the notes of the following chords. Note that `left` and `right` are mutually exclusive – fingerings may be placed only on one side or the other, not both.
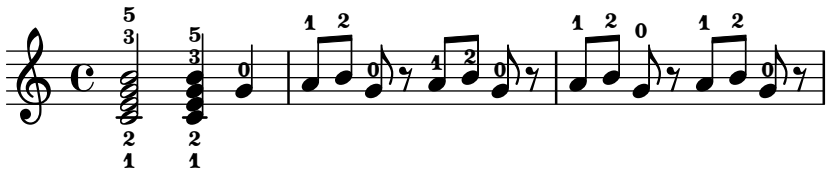
```
\relative c' {
  \set fingeringOrientations = #'(left)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(down)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(down right up)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(up)
  <c-1 e-3 a-5>4
  \set fingeringOrientations = #'(left)
  <c-1>2
  \set fingeringOrientations = #'(down)
  <e-3>2
  \set stringNumberOrientations = #'(up left down)
  <f\3 a\2 c\1>1
  \set strokeFingerOrientations = #'(down right up)
  <c\rightHandFinger 1 e\rightHandFinger 2 c'\rightHandFinger 4 >
}
```



Allowing fingerings to be printed inside the staff

By default, vertically oriented fingerings are positioned outside the staff; that behavior, however, may be disabled. Attention needs to be paid to situations where fingerings and stems are in the same direction: by default, fingerings will avoid only beamed stems. That setting can be changed to avoid no stems or all stems; the following example demonstrates these two options, as well as how to go back to the default behavior.

```
\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
  a8[-1 b]-2 g-0 r
  \override Fingering.add-stem-support = ##f
  a[-1 b]-2 g-0 r
  \override Fingering.add-stem-support = ##t
  a[-1 b]-2 g-0 r
  \override Fingering.add-stem-support = #only-if-beamed
  a[-1 b]-2 g-0 r
}
```



See also

Notation Reference: Section 36.1 [Direction and placement], page 774.

Snippets: Section “Editorial annotations” in *Snippets*.

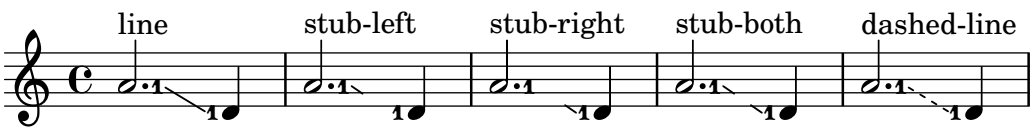
Internals Reference: Section “FingeringEvent” in *Internals Reference*, Section “fingering-event” in *Internals Reference*, Section “Fingering_engraver” in *Internals Reference*, Section “New_fingering_engraver” in *Internals Reference*, Section “Fingering” in *Internals Reference*.
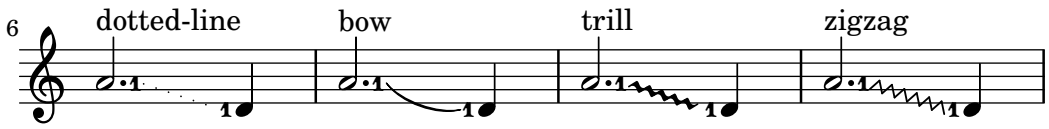
7.1.3 Gliding fingers

For stringed instruments a gliding finger is often indicated by a line connecting the same finger to be used for notes played at different positions on the same string. This line is initiated with `\glide` entered before a `Fingering` and ends with the next occurrence of the same finger. The line may be printed in various styles.

```
mus = {
  \set fingeringOrientations = #'(right)
  <a'\glide-1>2.
  \set fingeringOrientations = #'(left)
  <d'-1>4
}

{
  <>^"line"
  \mus
  <>^"stub-left"
  \override FingerGlideSpanner.style = #'stub-left
  \mus
  <>^"stub-right"
  \override FingerGlideSpanner.style = #'stub-right
  \mus
  <>^"stub-both"
  \override FingerGlideSpanner.style = #'stub-both
  \mus
  <>^"dashed-line"
  \override FingerGlideSpanner.style = #'dashed-line
  \mus
  \break
  <>^"dotted-line"
  \override FingerGlideSpanner.style = #'dotted-line
  \mus
  <>^"bow"
  \override FingerGlideSpanner.style = #'bow
  \mus
  <>^"trill"
  \override FingerGlideSpanner.style = #'trill
  \mus
  <>^"zigzag"
  \override FingerGlideSpanner.style = #'zigzag
  \mus
}
```
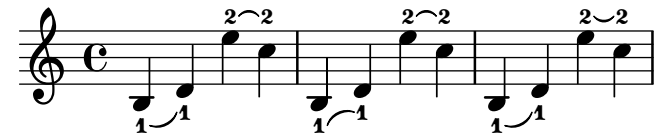




If `style` is set to `'bow` the direction of the bow may be adjusted using direction modifiers.

```
{
  \override FingerGlideSpanner.style = #'bow
  \set fingeringOrientations = #'(down)
  <b\glide-1>4 <d'-1>
  \set fingeringOrientations = #'(up)
  <e''\glide-2> <c''-2>

  \set fingeringOrientations = #'(down)
  <b^\glide-1>4 <d'-1>
  \set fingeringOrientations = #'(up)
  <e''^\glide-2> <c''-2>

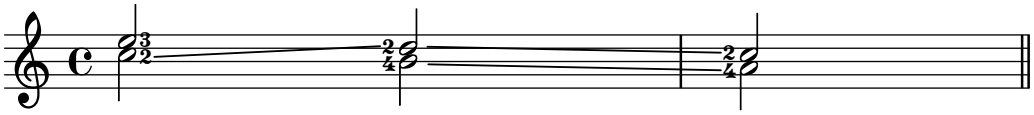
  \set fingeringOrientations = #'(down)
  <b_\glide-1>4 <d'-1>
  \set fingeringOrientations = #'(up)
  <e''_\glide-2> <c''-2>
}
```



If the `Finger_glide_engraver` is moved to `Staff` context `Fingering` grobs from different `Voice` contexts may be connected.

```
\score {
  \new Staff <<
    \new Voice {
      \voiceOne
      \set fingeringOrientations = #'(right)
      <e''-3>2
      \set fingeringOrientations = #'(left)
      <d''-\tweak bound-details.left.padding 2.5 \glide-2>
      <c''-2>
      \bar "||"
    }
    \new Voice {
      \voiceTwo
      \set fingeringOrientations = #'(right)
      <c''\glide-2>
      \set fingeringOrientations = #'(left)
      <b'-\tweak bound-details.left.padding 2.5 \glide-4>
      <a'-4>
    }
  >>
  \layout {
    ragged-right = ##f
```

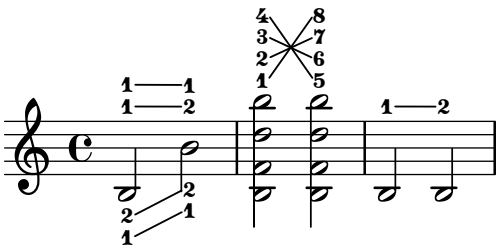
```
    \context {
      \Voice
      \remove Finger_glide_engraver
    }
    \context {
      \Staff
      \consists Finger_glide_engraver
    }
  }
}
```



To connect different fingers or multiple instances of the same finger set the `id` property with `\=` taking a non-negative integer or a symbol or `\tweak` the `text` property.

```
{
  b2 \glide \= #'foo ^1
     \glide \= #'bar ^1
     \glide _2
     \glide _1
  b' \= #'foo ^2
     \= #'bar ^1
     _2
     _1

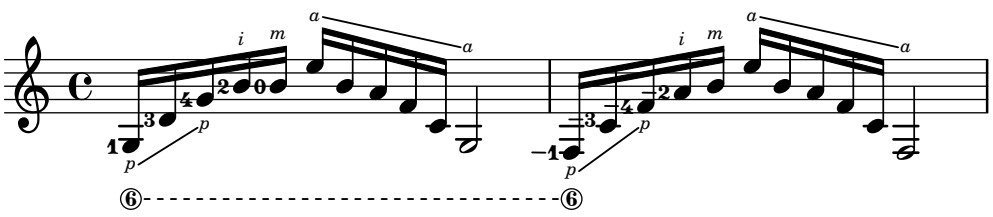
  \set fingeringOrientations = #'(up)
  <
   b\glide \=1 -1
   f'\glide \=2 -2
   d''\glide \=3 -3
   b''\glide \=4 -4
  >
  <
   b\=4 -5
   f'\=3 -6
   d''\=2 -7
   b''\=1 -8
  >
  b\glide -1 b\tweak text "2" -1
}
```



The `FingerGlideSpanner` may also connect string numbers indicating to play on the same string, or stroke fingers indicating an *Arrastre*.

```
{
  \override StringNumber.staff-padding = 7
  \omit TupletNumber
  \set fingeringOrientations = #'(left)
  \tuplet 5/4 4 {
    \set strokeFingerOrientations = #'(down)
    <
     g-\tweak style #'stub-right \glide-1
       \glide \rightHandFinger #1
       -\tweak style #'dashed-line \glide _\6
    >16
    <d'-\tweak style #'stub-right \glide -3 >
    <g'-\tweak style #'stub-right \glide -4 \rightHandFinger #1 >
    \set strokeFingerOrientations = #'(up)
    <b'-\tweak style #'stub-right \glide -2 \rightHandFinger #2 >
    <b'-0\rightHandFinger #3 >
    e''\glide \rightHandFinger #4
    b' a' f' c'
  }
  g2\rightHandFinger #4

  \tuplet 5/4 4 {
    \set strokeFingerOrientations = #'(down)
    <f-1 \glide \rightHandFinger #1 _\6 >16
    % Raise a bit, otherwise the stub line would be hidden
    % by the ledger line.
    <c'\tweak Y-offset #0.5 -3>
    <f' -4 \rightHandFinger #1 >
    \set strokeFingerOrientations = #'(up)
    <a'-2\rightHandFinger #2 >
    b'\rightHandFinger #3
    e''\glide \rightHandFinger #4
    b' a' f' c'
  }
  f2\rightHandFinger #4
}
```



See also

Music Glossary: Section “arrastre” in *Music Glossary*.
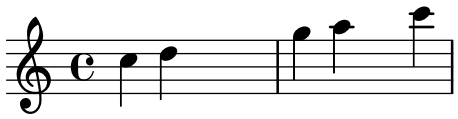
Notation Reference: Section 36.1 [Direction and placement], page 774.

Internals Reference: Section “FingerGlideEvent” in *Internals Reference*, Section “finger-glide-event” in *Internals Reference*, Section “Finger_glide_engraver” in *Internals Reference*, Section “finger-glide-interface” in *Internals Reference*, Section “FingerGlideSpanner” in *Internals Reference*.

7.1.4 Hidden notes
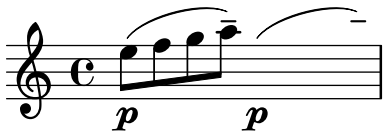
Hidden (or invisible or transparent) notes can be useful in preparing theory or composition exercises.

```
\relative {
  c''4 d
  \hideNotes
  e4 f
  \unHideNotes
  g a
  \hideNotes
  b
  \unHideNotes
  c
}
```



Note heads, stems, and flags, and rests are invisible. Beams are invisible if they start on a hidden note. Objects that are attached to invisible notes are still visible.

```
\relative c'' {
  e8(\p f g a)--
  \hideNotes
  e8(\p f g a)--
}
```



Predefined commands

`\hideNotes`, `\unHideNotes`.

See also

Learning Manual: Section “Visibility and color of objects” in *Learning Manual*.

Notation Reference: Section 2.2.2 [Invisible rests], page 69, Section 36.7 [Visibility of objects], page 784, Section 6.2.3 [Hiding staves], page 257.
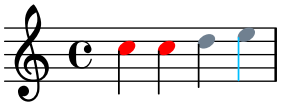
Snippets: Section “Editorial annotations” in *Snippets*.

Internals Reference: Section “Note_spacing_engraver” in *Internals Reference*, Section “NoteSpacing” in *Internals Reference*.

7.1.5 Coloring objects
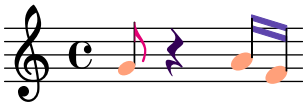
Individual objects may be assigned colors. Valid color names are listed in the Section B.7 [List of colors], page 902.

```
\override NoteHead.color = #red
c''4 c''
\override NoteHead.color = #(x11-color 'SlateGrey)
d''
\override Stem.color = "deepskyblue"
e''
```



In addition to a limited set of simple colors available as predefined variables (see 'Normal colors' in Section B.7 [List of colors], page 902), any color may be entered as a string. That string may be either a CSS (`https://www.w3.org/Style/CSS/`)-style predefined color name, or a hexadecimal color code prefixed by the '#' character (*inside* the double quotes):

```
\override NoteHead.color = "lightsalmon"
\override Flag.color = "#E30074"
\override Beam.color = "#5e45ad"
\override Rest.color = "#3058"
g'8 \huge r4 a'16 f'
```
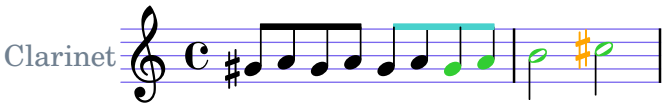


If that color code includes an alpha channel for semi-transparency, by using an eight-character code "*#RRGGBBAA*" or its shorthand form "*#RGBA*", it will be used in SVG output but not in PostScript/PDF output. In the previous example, the rest can be seen through only if the code is compiled with the SVG backend, as explained in Section 23.3 [Alternative output formats], page 651.

In a different way, the full range of colors defined for X11 (`https://en.wikipedia.org/wiki/X11_color_names`) can be accessed by using the Scheme function `x11-color`. That function takes one argument, which can be a symbol, such as `'DarkSeaGreen4`, or a string, such as `"DarkSeaGreen4"`. The first form is quicker to write and slightly more efficient; however, the second form also makes it possible to specify X11 colors as multiple words: in this instance, `"dark sea green 4"`.

If `x11-color` cannot make sense of the parameter, then the color returned defaults to black.

```
\new Staff \with {
  instrumentName = \markup {
    \with-color #(x11-color 'SlateGrey) "Clarinet"
    }
  }
  \relative c'' {
  \override Staff.StaffSymbol.color = #(x11-color 'SlateBlue2)
  gis8 a
  \override Beam.color = #(x11-color "medium turquoise")
  gis a
  \override Accidental.color = #(x11-color 'orange)
  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
}
```
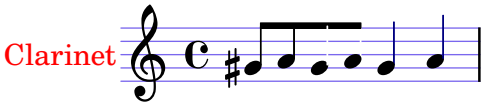


LilyPond also supports a set of eight color names (https://jfly.uni-koeln.de/color) that is unambiguous to both color-blind and non-color-blind people. Use universal-color to access them.

```
\markup \with-color #(universal-color 'vermillion) vermillion
```

vermillion

Exact RGB colors can be specified using the Scheme function rgb-color. This function takes three arguments used respectively for the *red*, *green* and *blue* channels, and an optional *alpha* number for semi-transparency. (All values must be numbers from 0 to 1.) Again, transparency is only supported in SVG output; for example, in the following fragment the staff's clef can be seen through when rendered in SVG.

```
\new Staff \with {
    instrumentName = \markup {
      \with-color #(x11-color 'red) "Clarinet"
      }
    \override Clef.color = #(rgb-color 0 0 0 0.5)
  }
\relative c'' {
  \override Staff.StaffSymbol.color = #(x11-color 'SlateBlue2)
  \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 B.7 [List of colors], page 902, Section 35.7 [\tweak and \single], page 764.

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.
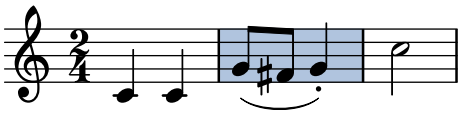
Not all X11 colors are distinguishable in a web browser, i.e., a web browser might not display a difference between LimeGreen and ForestGreen. For web use CSS colors are recommended, as detailed in Section B.7 [List of colors], page 902.

Notes in a chord cannot be separately colored with \override; use \tweak or the equivalent \single\override before the respective note instead, see Section 35.7 [\tweak and \single], page 764.

7.1.6 Staff highlights

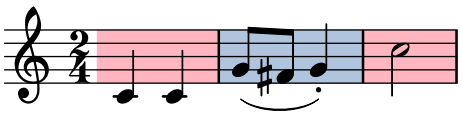
For analytical or pedagogical purposes, it may be useful to "highlight" musical passages, for example in order to show local tonality. This can be done using the `\staffHighlight` command, which expects a color. For all ways to enter colors, see Section 7.1.5 [Coloring objects], page 289. The highlight is terminated using `\stopStaffHighlight`.

```
\relative {
  \time 2/4
  c'4 4
  \staffHighlight "lightsteelblue"
  g'8( fis g4)-.
  \stopStaffHighlight
  c2
}
```



If there are consecutive highlights, it is not necessary to write `\stopStaffHighlight`, as `\staffHighlight` also implicitly terminates the current highlight, if any. Similarly, it is not necessary to add `\stopStaffHighlight` at the end of the piece. This is particularly handy if every measure is to be highlighted.

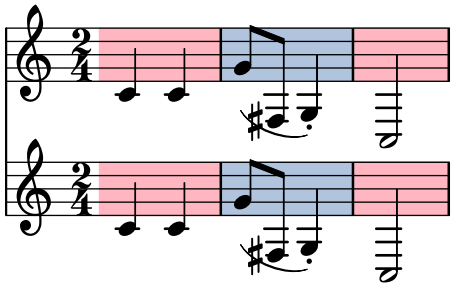
```
\relative {
  \time 2/4
  \staffHighlight "lightpink"
  c'4 4
  \staffHighlight "lightsteelblue"
  g'8( fis g4)-.
  \staffHighlight "lightpink"
  c2
}
```



By default, staves are highlighted separately.

```
music = {
  \time 2/4
  \staffHighlight "lightpink"
  c'4 4
  \staffHighlight "lightsteelblue"
  g'8( fis g4)-.
  \staffHighlight "lightpink"
  c2
}

<<
  \new Staff \music
  \new Staff \music
>>
```

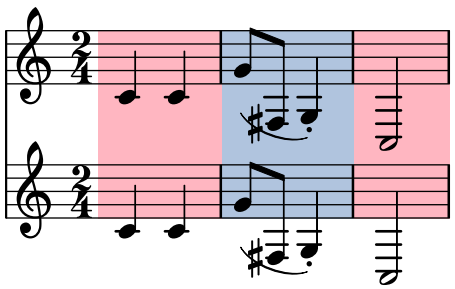


However, several staves can be highlighted together by moving `Staff_highlight_engraver` to a higher context than `Staff` (or `RhythmicStaff`, or similar). This is done using the `\consists` and `\remove` commands; See Section 33.4 [Modifying context plug-ins], page 745, for more information. For example, if the engraver is moved to `Score`, the highlights are shared by all staves.

```
\layout {
  \context {
    \Staff
    \remove Staff_highlight_engraver
  }
  \context {
    \Score
    \consists Staff_highlight_engraver
  }
}

music = {
  \time 2/4
  \staffHighlight "lightpink"
  c'4 4
  \staffHighlight "lightsteelblue"
  g'8( fis g4)-.
  \staffHighlight "lightpink"
  c2
}

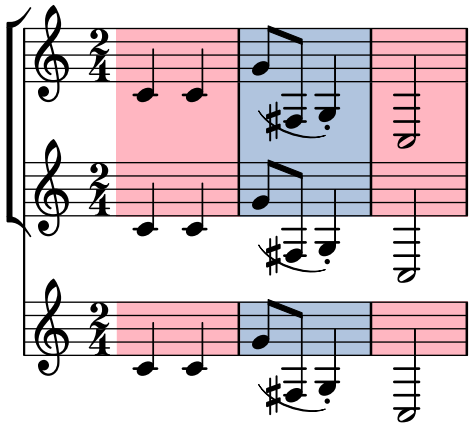
<<
  \new Staff \music
  \new Staff \music
>>
```



`Staff_highlight_engraver` may also be moved to intermediate contexts such as `StaffGroup`.

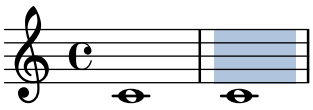
```
music = {
  \time 2/4
  \staffHighlight "lightpink"
  c'4 4
  \staffHighlight "lightsteelblue"
  g'8( fis g4)-.
  \staffHighlight "lightpink"
  c2
}

<<
  \new StaffGroup \with { \consists Staff_highlight_engraver } <<
    \new Staff \with { \remove Staff_highlight_engraver } \music
    \new Staff \with { \remove Staff_highlight_engraver } \music
  >>
  \new Staff \music
>>
```



The StaffHighlight.shorten-pair property may be used to tweak the horizontal start and end of the highlight span.

```
{
  c'1
  \once \override Staff.StaffHighlight.shorten-pair = #'(1.0 . 1.0)
  \staffHighlight lightsteelblue
  c'1
}
```



Predefined commands

\staffHighlight, \stopStaffHighlight.

See also

Notation Reference: Section 7.1.5 [Coloring objects], page 289, Section 33.4 [Modifying context plug-ins], page 745.

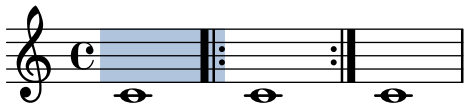
Internals Reference: Section “StaffHighlight” in *Internals Reference*, Section “staff-highlight-interface” in *Internals Reference*, Section “Staff_highlight_engraver” in *Internals*

Reference, Section “StaffHighlightEvent” in *Internals Reference*, Section “staff-highlight-event” in *Internals Reference*.

Known issues and warnings

The behavior of highlights may not be what is expected at start repeat bar lines. The `shorten-pair` property demonstrated above can be used to work around this issue.
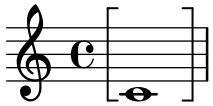
```
{
  \staffHighlight "lightsteelblue"
  c'1
  \stopStaffHighlight
  \repeat volta 2 { c'1 }
  c'1
}
```



7.1.7 Brackets for optional material

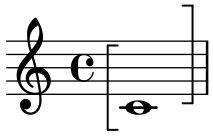
Optional additional material can be enclosed in brackets that pass through the staff.

```
{
  \startOptionalMaterial
  c'1
  \stopOptionalMaterial
}
```



The note positions that the bracket encompasses can be overridden. The endpoints of the interval are measured in staff spaces from the center of the staff to the center of the note.

```
{
  \tweak OptionalMaterialBracket.positions #'(-4 . 1)
  \startOptionalMaterial
  c'1
  \once \override Staff.OptionalMaterialBracket.positions =
  #'(-2 . 4)
  \stopOptionalMaterial
}
```



Predefined commands

`\startOptionalMaterial`, `\stopOptionalMaterial`.

See also
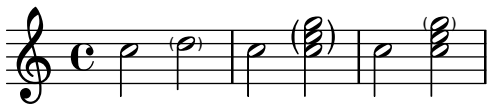
Snippets: Section “Editorial annotations” in *Snippets*.

Internals Reference: Section “Optional_material_bracket_engraver” in *Internals Reference*, Section “OptionalMaterialBracket” in *Internals Reference*, Section “optional-material-bracket-interface” in *Internals Reference*.

7.1.8 Parentheses
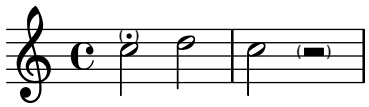
Objects may be parenthesized by prefixing the music event with \parenthesize.

```
\relative {
  c''2 \parenthesize d
  c2 \tweak Parentheses.font-size 2 \parenthesize <c e g>
  c2 <c e \parenthesize g>
}
```



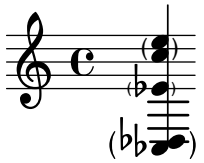
Non-note objects may be parenthesized as well. For articulations, a hyphen is needed before the \parenthesize command.

```
\relative {
  c''2-\parenthesize -. d
  c2 \parenthesize r
}
```



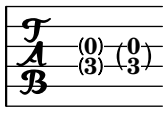
To parenthesize a group of notes in a chord, use a parallel music construct << ... >>.

```
\new Voice \relative c {
  <<
    { \tweak Parentheses.font-size 0 \parenthesize <ces des> }
    { \parenthesize ees' }
    { \tweak Parentheses.font-size -2 \parenthesize <c' e> }
  >>
}
```



In tablature specify NoteColumn to parenthesize the chord.
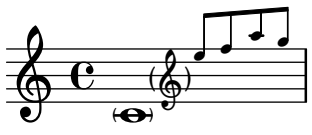
```
\new TabVoice {
  \override Parentheses.font-size = 0
  \parenthesize <f g>
  \parenthesize NoteColumn <f g>
}
```



This second form of the \parenthesize command involves a grob path: either \parenthesize *ContextName.GrobName* or just \parenthesize *GrobName* (the latter implying

the bottommost context, typically `Voice`). This should be added before the musical moment, like a `\once \override`. This form makes it possible to parenthesize grobs that are only caused indirectly by events.

```
\new Staff \relative <<
  {
    \parenthesize NoteHead
    c'1
  }
  \new CueVoice {
    s2
    \voiceOne
    \once \override Staff.Parentheses.font-size = 3
    \parenthesize Staff.CueClef
    \cueClef treble
    e'8 f a g
  }
>>
```



See also

Snippets: Section "Editorial annotations" in *Snippets*.

Internals Reference: Section "Parenthesis_engraver" in *Internals Reference*, Section "Parentheses" in *Internals Reference*, Section "parentheses-interface" in *Internals Reference*.

Known issues and warnings

Currently, the `font-size` property of the `Parentheses` grob has to be adjusted manually to obtain correctly sized parentheses on chords and some other objects.

7.1.9 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 36.1 [Direction and placement], page 774.

Predefined commands
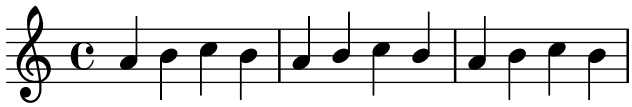
`\stemUp`, `\stemDown`, `\stemNeutral`.

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
}
```
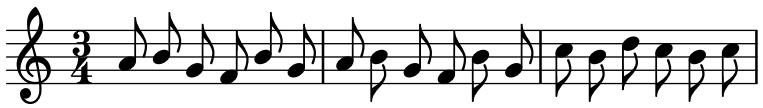


Automatically changing the stem direction of the middle note based on the melody

LilyPond can alter the stem direction of the middle note on a staff so that it follows the melody, by adding the `Melody_engraver` to the `Voice` context.

The context property `suspendMelodyDecisions` may be used to turn off this behavior locally.

```
\relative c'' {
  \time 3/4
  a8 b g f b g |
  \set suspendMelodyDecisions = ##t
  a  b g f b g |
  \unset suspendMelodyDecisions
  c  b d c b c |
}

\layout {
  \context {
    \Voice
    \consists "Melody_engraver"
    \autoBeamOff
  }
}
```



See also

Notation Reference: Section 36.1 [Direction and placement], page 774.

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*.

7.2 Outside the staff
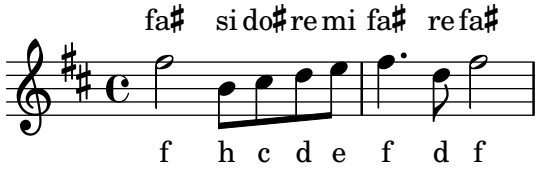
This section discusses how to add emphasis to elements in the staff from outside of the staff.

7.2.1 Note names

Note names can be printed as text, by using the `NoteNames` context. When used simultaneously with a regular staff, that makes it possible to synchronize each note with its name, printed above or below the Staff.

```
\language "italiano"
melody = \relative do'' {
  fad2 si,8 dod re mi fad4. re8 fad2
}

<<
  \new NoteNames { \melody }
  \new Staff { \key si \minor \melody }
  \new NoteNames {
    \set printNotesLanguage = "deutsch"
    \set printAccidentalNames = ##f
    \melody
  }
>>
```

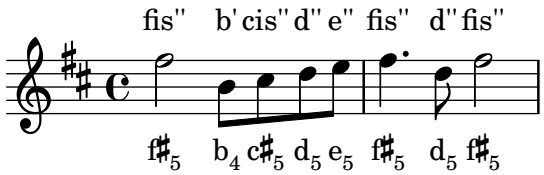


By default, note names are printed in the same language used for music entry; however, the `printNotesLanguage` property allows to select any other language available (see Section 1.1.5 [Note names in other languages], page 10). Whether accidentals should be printed or not is determined through the `printAccidentalNames` property.

By setting both that property to a symbol and `printOctaveNames` to `#t`, note names can be obtained that closely resemble LilyPond entry syntax. If a more general result is desired, 'scientific' octave names may also be obtained.

```
melody = \relative c'' {
  fis2 b,8 cis d e fis4. d8 fis2
}

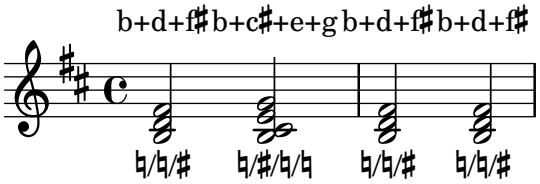
<<
  \new NoteNames {
    \set printOctaveNames = ##t
    \set printAccidentalNames = #'lily
    \melody
  }
  \new Staff { \key b \minor \melody }
  \new NoteNames {
    \set printOctaveNames = #'scientific
    \melody
  }
>>
```



The `noteNameSeparator` property defines how chords will be printed. Other formatting functions may be defined as `noteNameFunction`; such a function must expect a `pitch` and a `context` argument, even if one of these can then be ignored.

```
somechords = \relative c' {
  <b d fis>2 <b cis e g> <b d fis> q
}

<<
  \new NoteNames {
    \set noteNameSeparator = "+"
    \somechords
  }
  \new Staff { \key b \minor \somechords }
  \new NoteNames {
    \set noteNameFunction =
      #(lambda (pitch ctx)
         (accidental->text-markup
          (ly:pitch-alteration pitch)))
    \somechords
  }
>>
```



See also

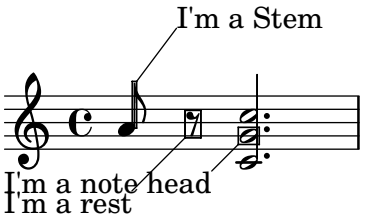
Notation Reference: Section 1.1.5 [Note names in other languages], page 10.

Internals Reference: Section “NoteName” in *Internals Reference*, Section “NoteNames” in *Internals Reference*, Section “Note_name_engraver” in *Internals Reference*.

7.2.2 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.

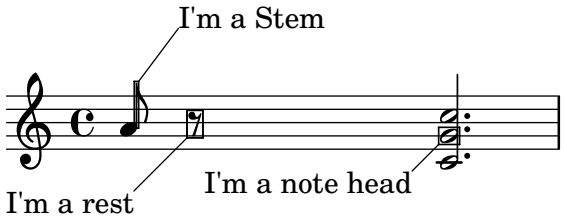
```
\new Voice \with { \consists Balloon_engraver }
\relative c'' {
  \balloonGrobText #'Stem #'(3 . 4) \markup { "I'm a Stem" }
  a8
  \balloonGrobText #'Rest #'(-4 . -4) \markup { "I'm a rest" }
  r
  <c, g'-\balloonText #'(-2 . -2) \markup { "I'm a note head" } c>2.
}
```



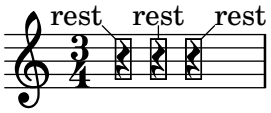
There are two music functions, `\balloonText` and `\balloonGrobText`; the former is used like `\tweak`, but only within chords, to attach text to an individual note of the chord; the latter is used like `\once \override` to attach text to any grob anywhere.

Balloon text does not influence note spacing, but this can be altered:

```
\new Voice \with { \consists Balloon_engraver }
\relative c'' {
  \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.
}
```
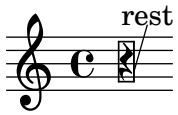


The default behavior for the attachment point of the line on the frame and the alignment of the balloon text is demonstrated below.



The default calculation of the attachment point can be overruled using the `X-attachment` and `Y-attachment` properties, which take values between -1 and 1, with the limits corresponding to the left (resp. bottom) and right (resp. top) of the frame. Alignment of the text is controlled by the properties `text-alignment-X` and `text-alignment-Y`, which have a similar form.

```
\new Voice \with {
  \consists Balloon_engraver
}
{
  \once \override BalloonText.Y-attachment = -0.5
  \once \override BalloonText.text-alignment-X = 0.0
  \balloonGrobText Rest #'(1 . 3.5) "rest"
  r4
}
```



Predefined commands

`\balloonLengthOn`, `\balloonLengthOff`.

See also

Snippets: Section “Editorial annotations” in *Snippets*.

Internals Reference: Section “AnnotateOutputEvent” in *Internals Reference*, Section “Balloon_engraver” in *Internals Reference*, Section “BalloonText” in *Internals Reference*, Section “balloon-interface” in *Internals Reference*.
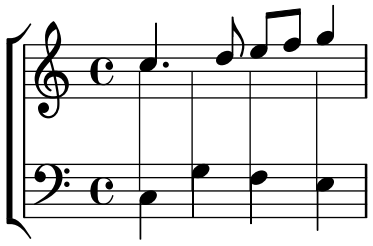
7.2.3 Grid lines

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 = #1/4
  }
  \context {
    \Score
    \consists Grid_line_span_engraver
  }
}

\score {
  \new ChoirStaff <<
    \new Staff \relative {
      \stemUp
      c''4. d8 e8 f g4
    }
    \new Staff \relative {
      \clef bass
      \stemDown
      c4 g' f e
    }
  >>
}
```



Selected snippets

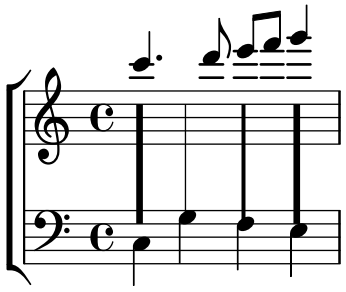
Grid lines: changing their appearance

The appearance of grid lines can be changed by overriding some of their properties.

```
\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 = #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
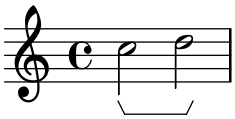
Snippets: Section “Editorial annotations” in *Snippets*.

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*.

7.2.4 Analysis brackets

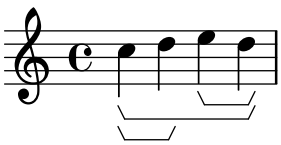
Brackets are used in musical analysis to indicate structure in musical pieces. Simple horizontal brackets are supported.

```
\layout {
  \context {
    \Voice
    \consists Horizontal_bracket_engraver
  }
}
\relative {
  c''2\startGroup
  d\stopGroup
}
```



Analysis brackets may be nested.

```
\layout {
  \context {
    \Voice
    \consists Horizontal_bracket_engraver
  }
}
\relative {
  c''4\startGroup\startGroup
  d4\stopGroup
  e4\startGroup
  d4\stopGroup\stopGroup
}
```



Selected snippets
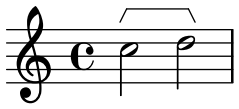
Analysis brackets above the staff

Simple horizontal analysis brackets are added below the staff by default. The following example shows a way to place them above the staff instead.

```
\layout {
  \context {
    \Voice
    \consists "Horizontal_bracket_engraver"
  }
}

\relative c'' {
  \once \override HorizontalBracket.direction = #UP
```

```
  c2\startGroup
  d2\stopGroup
}
```



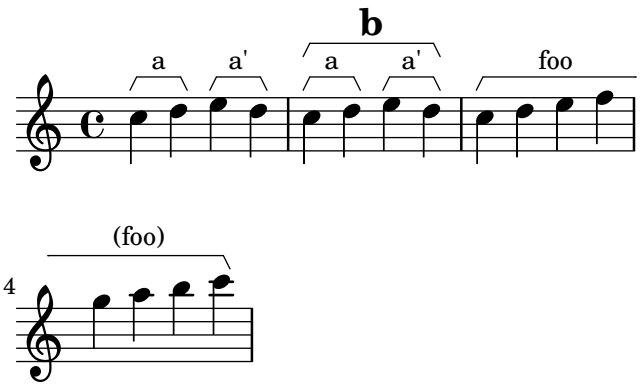
Analysis brackets with labels

Text markup may be added to analysis brackets using the `text` property of the `HorizontalBracketText` grob. Adding different texts to brackets beginning at the same time requires the `\tweak` command.

Bracket text gets parenthesized after a line break. The vertical order of nested brackets can be controlled with the `outside-staff-priority` property.

```
\layout {
  \context {
    \Voice
    \consists "Horizontal_bracket_engraver"
    \override HorizontalBracket.direction = #UP
  }
}

{
  \once\override HorizontalBracketText.text = "a"
    c''\startGroup d''\stopGroup
    \once\override HorizontalBracketText.text = "a'"
    e''\startGroup d''\stopGroup |
  c''-\tweak outside-staff-priority #801
      \tweak HorizontalBracketText.text
        \markup \bold \huge "b" \startGroup
     -\tweak HorizontalBracketText.text "a" \startGroup
    d''\stopGroup
    e''-\tweak HorizontalBracketText.text "a'" \startGroup
    d''\stopGroup\stopGroup |
  c''-\tweak HorizontalBracketText.text foo \startGroup
    d'' e'' f'' | \break
  g'' a'' b'' c'''\stopGroup
}
```

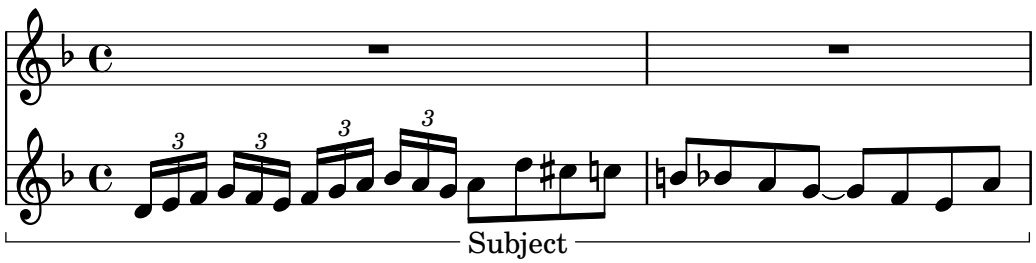


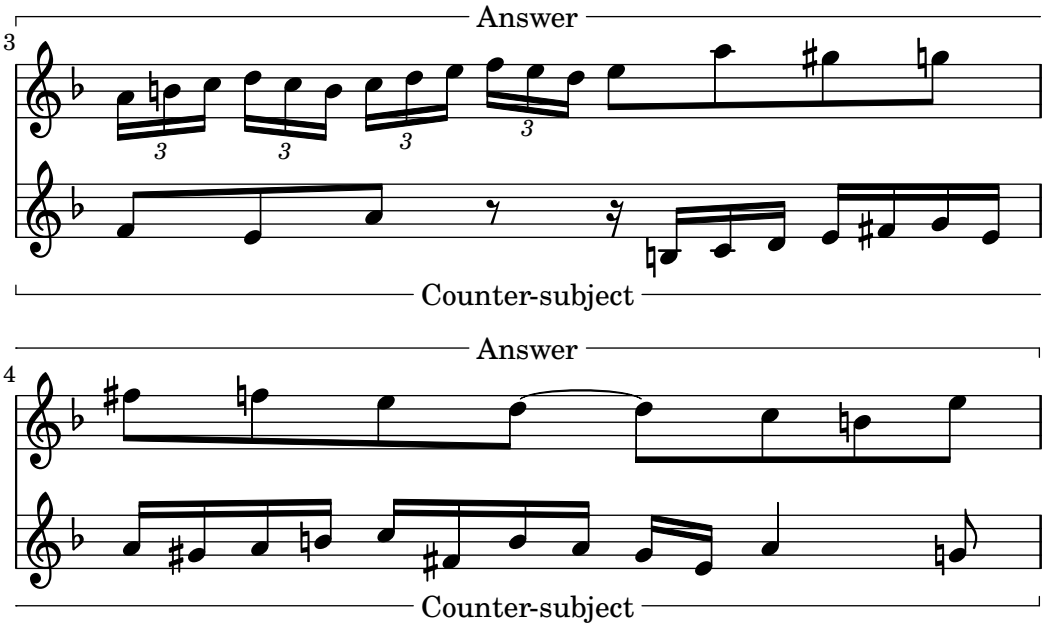
Measure spanner

Measure spanners are an alternate way to print annotated brackets. As opposed to horizontal brackets, they extend between two bar lines rather than two notes. The text is displayed in the center of the bracket.

```
\layout {
  \context {
    \Staff
    \consists Measure_spanner_engraver
  }
}

<<
  \new Staff \relative c'' {
    \key d \minor
    R1*2
    \tweak text "Answer"
    \startMeasureSpanner
    \tuplet 3/2 8 {
      a16[ b c] d[ c b]  c[ d e] f[ e d]
    }
    e8 a gis g
    fis f e d~ d c b e
    \stopMeasureSpanner
  }
  \new Staff \relative c' {
    \key d \minor
    \tweak text "Subject"
      \tweak direction #DOWN
      \startMeasureSpanner
    \tuplet 3/2 8 {
      d16[ e f] g[ f e] f[ g a] bes[ a g]
    }
    a8 d cis c
    b bes a g~ g f e a
    \stopMeasureSpanner
    \tweak text "Counter-subject"
      \tweak direction #DOWN
      \startMeasureSpanner
    f8 e a r r16 b, c d e fis g e
    a gis a b c fis, b a gis e a4 g8
    \stopMeasureSpanner
  }
>>
```
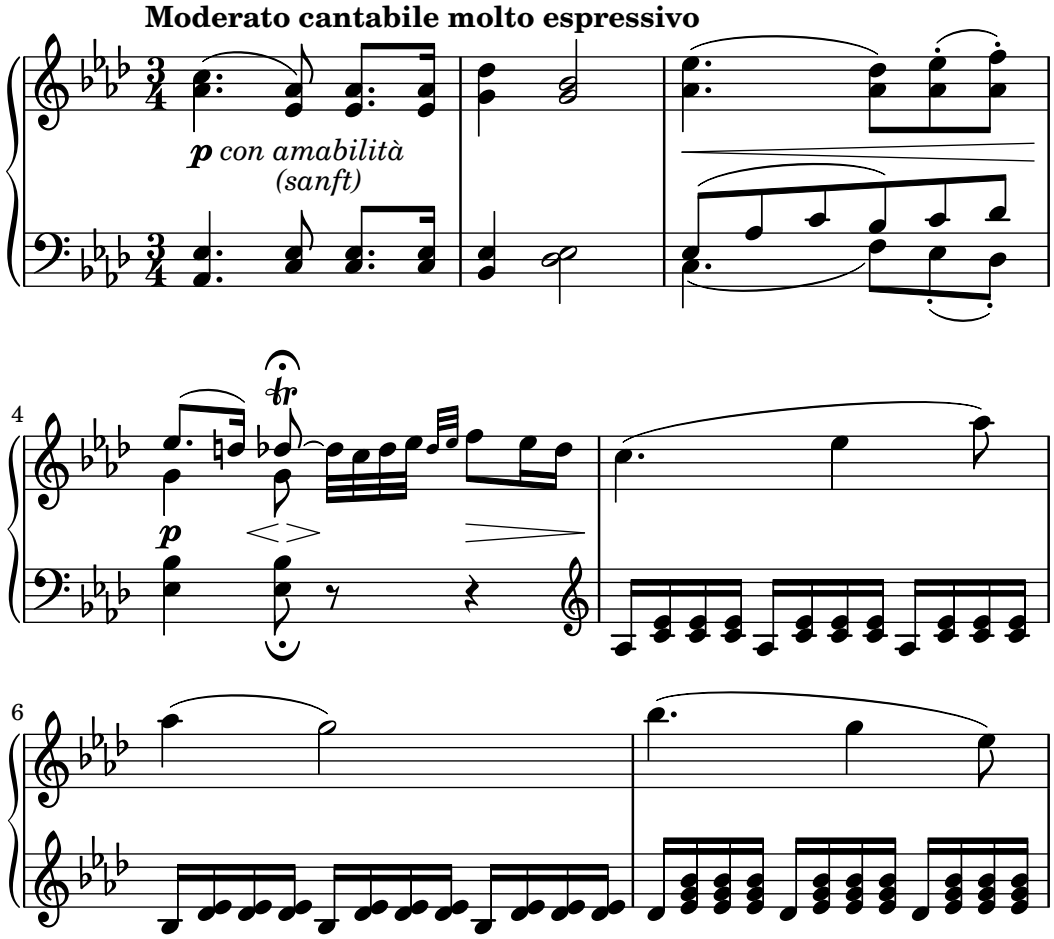




See also

Internals Reference: Section “Horizontal_bracket_engraver” in *Internals Reference*, Section “HorizontalBracket” in *Internals Reference*, Section “horizontal-bracket-interface” in *Internals Reference*, Section “HorizontalBracketText” in *Internals Reference*, Section “horizontal-bracket-text-interface” in *Internals Reference*, Section “Measure_spanner_engraver” in *Internals Reference*, Section “MeasureSpanner” in *Internals Reference*, Section “measure-spanner-interface” in *Internals Reference*, Section “Staff” in *Internals Reference*.

8 Text



This section explains how to include text (with various formatting) in music scores.

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 22.4.1 [Text encoding], page 647.

8.1.1 Text objects overview

Simple text objects are entered as strings between double quotes (these are optional for a single word). The markup mode is a richer tool that can accept a variety of advanced text formatting and graphical enhancements, as detailed in Section 8.2 [Formatting text], page 320.

As such, markup blocks may be used:

- in any `TextScript` object (attached to notes with `-`, `^` or `_`); see Section 8.1.2 [Text scripts], page 310;
- as 'spanners', when some indications are prolonged over several beats or bars; see Section 8.1.3 [Text spanners], page 312;

- in any mark printed above the score, such as `RehearsalMark` or `MetronomeMark` objects respectively introduced with the `\mark` or `\tempo` keywords; see Section 8.1.5 [Text marks], page 314;
- as stand-alone text blocks, entered at the top level outside of any `\score` block (in this specific case the `\markup` or `\markuplist` command is mandatory, and cannot be omitted in favor of a simple text string between double quotes); see Section 8.1.6 [Separate text], page 318;
- in any definition inside the `\header` block (e.g., `title`, `subtitle`, `composer`), or in specific elements defined inside the `\paper` block such as `evenHeaderMarkup` for page numbers. This is explained in Chapter 21 [Titles and headers], page 600.

Many other text-based objects may be entered as markup blocks, even if that is not their primary use.

- Fingerings may easily be replaced with markup blocks, if introduced with the `\finger` command; see Section 7.1.2 [Fingering instructions], page 282.
- Lyric syllables may be formatted through the `\markup` command; see Section 9.1 [Common notation for vocal music], page 345.
- Chord names are in fact defined as markup blocks, and therefore may be redefined in the same way for customizing chord modifiers or chord exceptions; see Section 15.2 [Displaying chords], page 513.
- Dynamics are usually entered in a simple way; however it is possible to define Section 3.1.3 [New dynamic marks], page 168, as markup objects. Some dynamics such as *crescendo* are printed as spanners and may be redefined through properties such as `crescendoText`; see Section 3.1.2 [Dynamics], page 160.
- Less common objects are also made of markup blocks, such as Section 7.2.2 [Balloon help], page 300, indications.

In fact, it is possible to use `\markup` to customize the appearance of virtually any graphical object (or 'grob'), by overriding either its `text` property if it has one, or its `stencil` property. Some of the logic that makes this a possibility is explained in Section "Flexible architecture" in *Essay*.

The following example illustrates the ubiquity of markup blocks, not only as some of the objects listed above, but also by replacing musical objects with text objects through various methods.

```
\header { title = \markup "Header" }

dyn =
#(make-dynamic-script #{ \markup \serif "DynamicText" #})

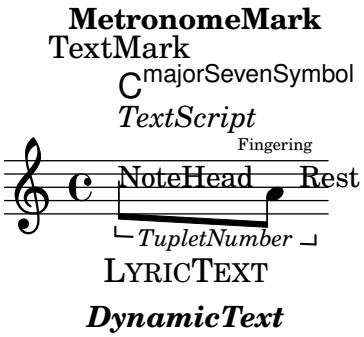
\markup \box "Top-level markup"

\score {
  <<
    \new ChordNames
    \with {
      majorSevenSymbol = \markup "majorSevenSymbol"
    }
    \chordmode { c1:maj7 }
    \new Staff {
      \tempo \markup "MetronomeMark"
      \textMark "TextMark"
```

```
      \once \override TupletNumber.text =
        \markup "TupletNumber"
      \tuplet 3/2 {
        \once \override NoteHead.stencil =
          #ly:text-interface::print
        \once \override NoteHead.text =
          \markup \lower #0.5 "NoteHead"
        c''8^\markup \italic "TextScript"
        a'\finger \markup \serif "Fingering"
        \once \override Rest.stencil =
          #(lambda (grob)
             (grob-interpret-markup grob #{
               \markup  "Rest"
               #}))
        r
      }
    }
    \new Lyrics \lyricmode {
      \markup \smallCaps "LyricText" 1
    }
    \new Dynamics { s1\dyn }
  >>
}
```

Header

Top-level markup



See also

Notation Reference: Section 8.2 [Formatting text], page 320, Section 8.1.2 [Text scripts], page 310, Section 8.1.3 [Text spanners], page 312, Section 8.1.5 [Text marks], page 314, Section 8.1.6 [Separate text], page 318, Section 7.1.2 [Fingering instructions], page 282, Section 9.1 [Common notation for vocal music], page 345, Section 15.2 [Displaying chords], page 513, Section 3.1.3 [New dynamic marks], page 168, Section 3.1.2 [Dynamics], page 160, Section 7.2.2 [Balloon help], page 300.
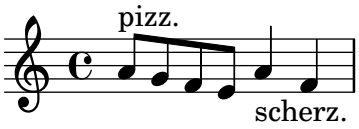
Essay on automated music engraving: Section “Flexible architecture” in *Essay*.

Snippets: Section “Text” in *Snippets*.

8.1.2 Text scripts
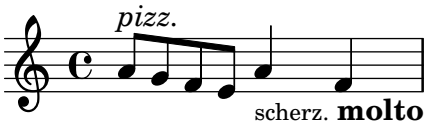
Simple “quoted text” indications may be added to a score, as demonstrated in the following example. Such indications may be manually placed above or below the staff, using the syntax described in Section 36.1 [Direction and placement], page 774.

```
\relative { a'8^"pizz." g f e a4-"scherz." f }
```
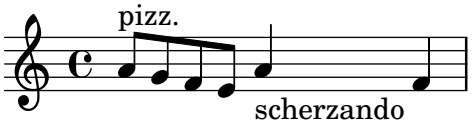


This syntax is actually a shorthand; more complex text formatting may be added to a note by explicitly using a \markup block, as described in Section 8.2 [Formatting text], page 320.

```
\relative {
  a'8^\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.

```
\relative {
  a'8^"pizz." g f e
  \textLengthOn
  a4_"scherzando" f
}
```



In addition to text scripts, articulations can be attached to notes. For more information, see Section 3.1.1 [Articulations and ornamentations], page 156.

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 8.2 [Formatting text], page 320, Section 36.1 [Direction and placement], page 774, Section 3.1.1 [Articulations and ornamentations], page 156.

Snippets: Section "Text" in *Snippets*.

Internals Reference: Section "TextScript" in *Internals Reference*.

Known issues and warnings

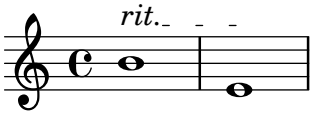
Checking to make sure that text scripts and lyrics are within the margins requires additional calculations. In cases where slightly faster performance is desired, use

```
\override Score.PaperColumn.keep-inside-line = ##f
```

8.1.3 Text spanners

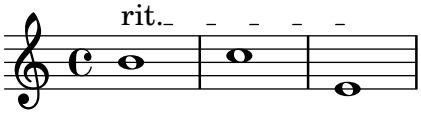
Some performance indications, e.g., *rallentando* or *accelerando*, are written as text and are extended over multiple notes with dotted lines. Such objects, called "spanners", may be created from one note to another using the following syntax:

```
\relative {
  \override TextSpanner.bound-details.left.text = "rit."
  b'1\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 8.2 [Formatting text], page 320.

```
\relative {
  \override TextSpanner.bound-details.left.text =
    \markup { \upright "rit." }
  b'1\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 36.5 [Line styles], page 782.

Predefined commands

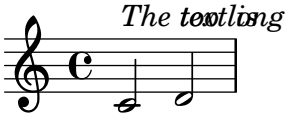
`\textSpannerUp`, `\textSpannerDown`, `\textSpannerNeutral`, `\startTextSpan`, `\stopTextSpan`.

Known issues and warnings

LilyPond is only able to handle one text spanner per voice.

The texts at the bounds of a text spanner may collide.

```
{
  \once \override TextSpanner.bound-details.left.text = "The text is"
  \once \override TextSpanner.bound-details.right.text = "too long"
  c'2\startTextSpan d'2\stopTextSpan
}
```



Workarounds can be found in Chapter 30 [Horizontal spacing], page 713.
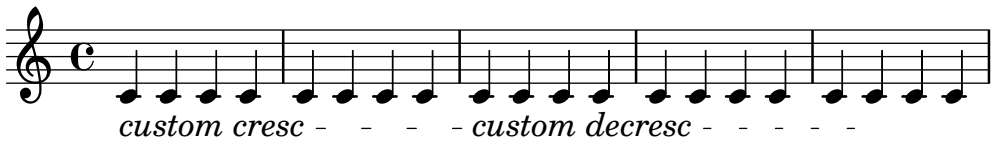
Selected snippets

Dynamics spanner with custom text

Postfix functions for custom crescendo text spanners. The spanners should start on the first note of the measure. One has to use -\mycresc, otherwise the spanner start will rather be assigned to the next note.

```
% Two functions for (de)crescendo spanners where you can explicitly
% give the spanner text.
mycresc =
#(define-music-function (mymarkup) (markup?)
   (make-music 'CrescendoEvent
               'span-direction START
               'span-type 'text
               'span-text mymarkup))
mydecresc =
#(define-music-function (mymarkup) (markup?)
   (make-music 'DecrescendoEvent
               'span-direction START
               'span-type 'text
               'span-text mymarkup))

\relative c' {
  c4-\mycresc "custom cresc" c4 c4 c4 |
  c4 c4 c4 c4 |
  c4-\mydecresc "custom decresc" c4 c4 c4 |
  c4 c4 c4 c4 |
  c4 c4\! c4 c4
}
```



See also

Notation Reference: Section 36.5 [Line styles], page 782, Section 3.1.2 [Dynamics], page 160, Section 8.2 [Formatting text], page 320.

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

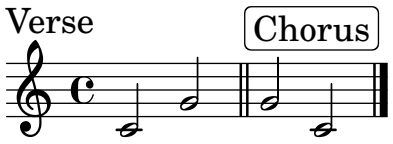
Internals Reference: Section “TextSpanner” in *Internals Reference*.

8.1.4 Section labels

The \sectionLabel command marks the beginning of a named passage. It is well suited for use at a section division created with \section, but it does not imply \section and may be used alone.

```
\fixed c' {
  \sectionLabel "Verse"
  c2 g
  \section
  \sectionLabel \markup { \rounded-box { Chorus } }
  g2 c
  \bar "|."
```

```
}
```



See also

Notation Reference: Section 8.2 [Formatting text], page 320, Section 2.5.5 [Rehearsal marks], page 142, Section 2.5.7 [Section divisions], page 147.

Snippets: Section "Text" in *Snippets*.

Internals Reference: Section "SectionLabel" in *Internals Reference*, Section "SectionLabelEvent" in *Internals Reference*.

8.1.5 Text marks
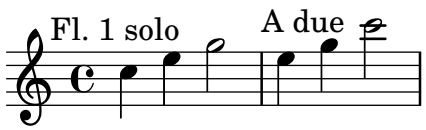
Text marks are textual objects that, unlike text scripts (see Section 8.1.2 [Text scripts], page 310), are not printed over notes but between notes, often aligned to a bar line.

> **Note:** Older LilyPond versions used the `\mark` command for text marks, even though it is primarily intended for rehearsal marks (see Section 2.5.5 [Rehearsal marks], page 142). The `\textMark` and `\textEndMark` commands are better suited for text marks with regard to their default settings as well as the ability to have several text marks at the same moment. It is therefore recommended to use `\textMark` and `\textEndMark` instead of `\mark "Text"` or `\mark \markup ...`. Note that if converting code that uses `\mark` for text marks, overrides using `RehearsalMark` should be changed to `TextMark`.

Before using text marks, it is recommended to seek a more specific command, if available. For text that identifies a section, use `\sectionLabel` (see Section 8.1.4 [Section labels], page 313). For jump instructions, use `\jump` (see Section 4.1.8 [Manual repeat marks], page 205). These commands have different default layout settings, and they create separate objects which can be styled differently from generic text marks in style sheets.
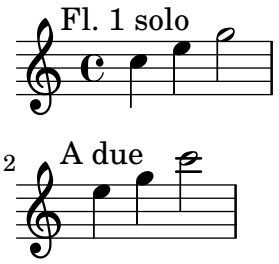
A text mark is entered using either `\textMark` or `\textEndMark`. The `\textMark` command draws a left-aligned mark.

```
\fixed c'' {
  \textMark "Fl. 1 solo"
  c4 e g2
  \textMark "A due"
  e4 g c'2
}
```
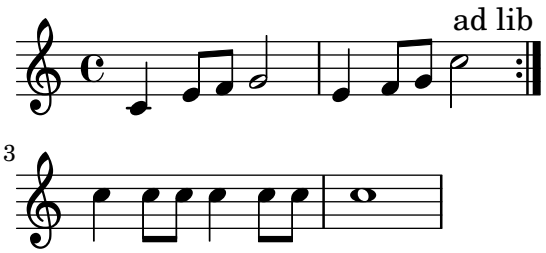


If a line break occurs at the point `\textMark` is used, the text appears on the next system.

```
\fixed c'' {
  \textMark "Fl. 1 solo"
  c4 e g2
  \break
  \textMark "A due"
  e4 g c'2
}
```
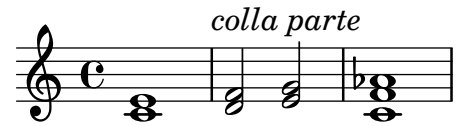


Unlike \textMark, \textEndMark creates a right-aligned mark. If it occurs on a line break, it is printed on the preceding system.

```
\fixed c' {
  \repeat volta 2 {
    c4 e8 f g2
    e4 f8 g c'2
    \textEndMark "ad lib"
  }
  \break
  c'4 8 8 4 8 8
  c'1
}
```



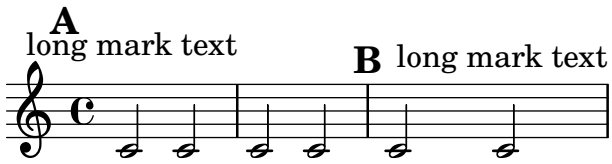
Complex text formatting may be added using a \markup block (see Section 8.2 [Formatting text], page 320).

```
\relative {
  <c' e>1
  \textMark \markup { \italic { colla parte } }
  <d f>2 <e g>
  <c f aes>1
}
```
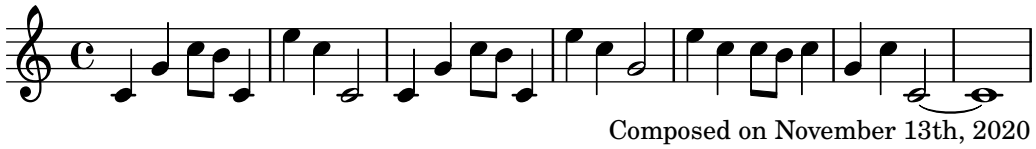


The \markLengthOn and \markLengthOff commands (see Section 2.3.3 [Metronome marks], page 87) can also be used on text marks.

```
{
  \mark \default
  c'2 2
  \textEndMark "long mark text"
  \markLengthOn
  2 2
  \mark \default
  2 2
  \textEndMark "long mark text"
}
```
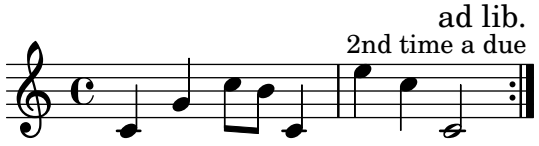


Text marks may be printed below the staff.

```
\fixed c' {
  c4 g c'8 b c4 e' c' c2
  c4 g c'8 b c4 e' c' g2
  e'4 c' c'8 b c'4
  g4 c' c2~ c1
  \tweak direction #DOWN
    \tweak font-size -1
      \textEndMark "Composed on November 13th, 2020"
}
```



There can be several text marks at the same moment. Their stacking order can be overridden using the `outside-staff-priority` property (see Section 29.3 [Vertical collision avoidance], page 710). Alternatively, a single text mark with `\markup \column { ... }` can achieve the same effect.

```
\fixed c' {
  \repeat volta 2 {
    c4 g c'8 b c4 e' c' c2
    \textEndMark "ad lib."
    \tweak outside-staff-priority 1200
      \tweak font-size -1
        \textEndMark "2nd time a due"
  }
}
```

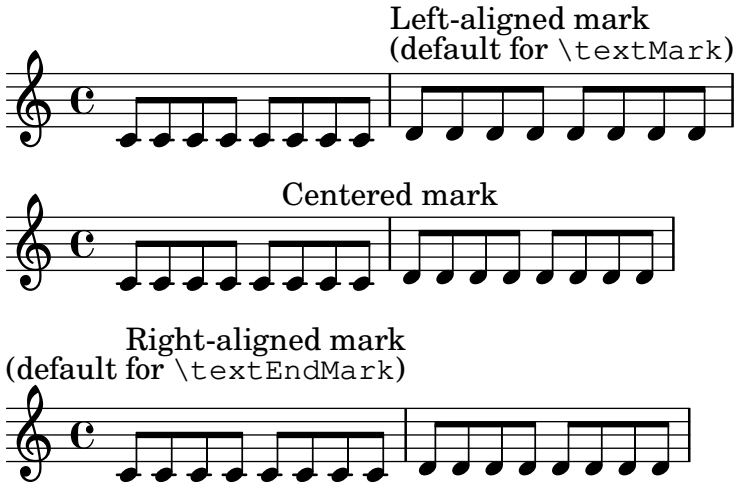


The alignment of a text mark can be changed.

```
{
  c'8 8 8 8 8 8 8 8
  \textMark \markup \column {
    \line { Left-aligned mark }
    \line { (default for \concat { \typewriter "\\textMark" ) } }
  }
  d'8 8 8 8 8 8 8 8
}

{
  c'8 8 8 8 8 8 8 8
  \tweak self-alignment-X #CENTER
    \textMark "Centered mark"
  d'8 8 8 8 8 8 8 8
}

{
  c'8 8 8 8 8 8 8 8
  \tweak self-alignment-X #RIGHT
    \textMark \markup \right-column {
      \line { Right-aligned mark }
      \line { (default for \concat { \typewriter "\\textEndMark" ) } }
    }
  d'8 8 8 8 8 8 8 8
}
```



Predefined commands

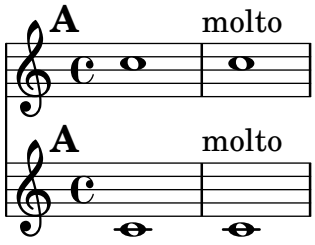
`\textMark`, `\textEndMark`, `\markLengthOn`, `\markLengthOff`.

Selected snippets

Printing marks on every staff

Although rehearsal and text marks are normally only printed above the topmost staff, they may also be printed on every staff.

```
\score {
  <<
    \new Staff { \mark \default c''1 \textMark "molto" c'' }
    \new Staff { \mark \default c'1 \textMark "molto" c' }
  >>
  \layout {
    \context {
      \Score
      \remove Mark_engraver
      \remove Text_mark_engraver
      \remove Staff_collecting_engraver
    }
    \context {
      \Staff
      \consists Mark_engraver
      \consists Text_mark_engraver
      \consists Staff_collecting_engraver
    }
  }
}
```



See also

Notation Reference: Section 2.3.3 [Metronome marks], page 87, Section 2.5.5 [Rehearsal marks], page 142, Section 8.1.4 [Section labels], page 313, Section 8.2 [Formatting text], page 320, Section 8.2.5 [Music notation inside markup], page 335, Section B.8 [The Emmentaler font], page 907.

Snippets: Section "Text" in *Snippets*.

Internals Reference: Section "TextMarkEvent" in *Internals Reference*, Section "Text_mark_engraver" in *Internals Reference*, Section "TextMark" in *Internals Reference*.

8.1.6 Separate text
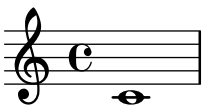
A `\markup` or `\markuplist` block can exist by itself, outside of any `\score` block, as a "top-level expression". This syntax is described in Section 20.5 [File structure], page 596.

```
\markup {
  Tomorrow, and tomorrow, and tomorrow...
}
```
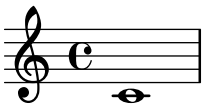
Tomorrow, and tomorrow, and tomorrow...

This allows printing text separately from the music, which is particularly useful when the input file contains several music pieces, as described in Section 20.2 [Multiple scores in a book], page 593.

```
\score {
  c'1
}
\markup {
  Tomorrow, and tomorrow, and tomorrow...
}
\score {
  c'1
}
```



Tomorrow, and tomorrow, and tomorrow...



Using \markuplist, separate text blocks can be spread over multiple pages, making it possible to print text documents or books entirely within LilyPond. For a description of this feature and the specific syntax it requires, see Section 8.2.1 [Text markup introduction], page 320.

Predefined commands

\markup, \markuplist.

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 { O 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 \italic {
      \line { O sacred feast }
      \line { in which Christ is received, }
      \line { the memory of His Passion is renewed, }
      \line { the mind is filled with grace, }
      \line { and a pledge of future glory is given to us. }
      \line { Amen. }
    }
    \hspace #1
```

```
  }
}
```

O sacrum convivium
in quo Christus sumitur,
recolitur memoria passionis ejus,
mens impletur gratia,
futurae gloriae nobis pignus datur.
Amen.

O sacred feast
in which Christ is received,
the memory of His Passion is renewed,
the mind is filled with grace,
and a pledge of future glory is given to us.
Amen.

See also

Notation Reference: Section 8.2 [Formatting text], page 320, Section 20.5 [File structure], page 596, Section 20.2 [Multiple scores in a book], page 593.

Snippets: Section "Text" in *Snippets*.

Internals Reference: Section "TextScript" in *Internals Reference*.

8.2 Formatting text

This section presents basic and advanced text formatting, using the markup mode specific syntax.

8.2.1 Text markup introduction

A `\markup` or `\markuplist` block is used to typeset text with an extensible syntax called "markup mode". Such blocks can be used in many contexts (see Section 8.1.1 [Text objects overview], page 308).

In markup mode, words are written as-is. A single word does not need any quotes.

```
\markup intenso
```

intenso
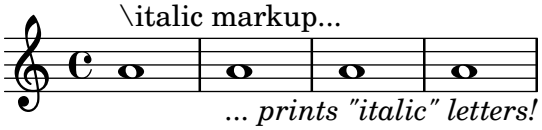
Several words can be grouped together by enclosing them in quotes.

```
\markup "molto intenso"
```

molto intenso

Apart from grouping, quoting also allows writing special characters such as '\' and '#' without affecting the formatting of the text. Double quotation marks themselves may be printed by preceding them with backslashes.

```
\relative {
  a'1^"\\italic markup..."
  a_\markup { \italic "... prints \"italic\" letters!" }
  a a
}
```



Formatting is achieved through markup commands. Their name is written preceded by a backslash. They expect a number of arguments specific to the command. For an exhaustive list of `\markup`-specific commands, see Section A.1 [Text markup commands], page 809.

```
\markup \italic "string. assai"
\markup \with-color "red" intenso
```

string. assai

intenso

Markup commands can be nested. The markup block ends when all commands have received their arguments.

```
\markup \with-color "red" \italic intenso
```

intenso

Several markup expressions can be grouped together within braces to form a so-called *markup list*. Without further formatting, the elements of a markup list are typeset in a row.

```
\markup { molto \italic intenso }
```

molto *intenso*

Some commands do not expect a markup but a markup list, allowing for more complex text arrangements than printing in a row.

```
\markup \center-column {
  \bold "Des Simplicius Simplicissimus Jugend"
  "Karl Amadeus Hartmann"
}
```

Des Simplicius Simplicissimus Jugend
Karl Amadeus Hartmann

Also, some commands do not return a markup but a markup list. The result can then be used where a markup list is expected. For a list of these commands, see Section A.2 [Text markup list commands], page 886.

```
\markup \string-lines
    "Twinkle, twinkle, little star,
     How I wonder what you are!"
```

Twinkle, twinkle, little star, How I wonder what you are!

```
\markup \center-column \string-lines
    "Twinkle, twinkle, little star,
     How I wonder what you are!"
```

Twinkle, twinkle, little star,
How I wonder what you are!

Elements of a nested markup list are simply treated as elements of the main markup list.

```
\markup \center-column {
  \bold "Des Simplicius Simplicissimus Jugend"
  { Karl Amadeus \smallCaps Hartmann }
}
```

Des Simplicius Simplicissimus Jugend
Karl
Amadeus
Hartmann

To group elements of a nested markup list in a row, apply the `\line` command to the markup list. This stacks elements from the markup list horizontally into a single markup.

```
\markup \center-column {
  \bold "Des Simplicius Simplicissimus Jugend"
  \line { Karl Amadeus \smallCaps Hartmann }
}
```

Des Simplicius Simplicissimus Jugend
Karl Amadeus HARTMANN

A special feature is the handling of commands taking markups when applied to markup lists. When a command expects a markup as its last argument, and a markup list is given for this argument, the markup command is applied to each of the individual markups in the list.

```
\markup \box { Karl Amadeus \smallCaps Hartmann }
```

Karl Amadeus HARTMANN

In this case, the result is in turn a markup list, which can be passed to a command expecting a markup list, or to one expecting a markup, with again the mapping behavior described above in the latter case.

```
\markup \center-column \box { Karl Amadeus \smallCaps Hartmann }
\markup \rotate #30 \box { Karl Amadeus \smallCaps Hartmann }
```

Karl
Amadeus
HARTMANN

Karl Amadeus HARTMANN

Apply `\line` to a markup list in order to make it treated as a single markup argument.

```
\markup \box { Karl Amadeus \smallCaps Hartmann }
\markup \box \line { Karl Amadeus \smallCaps Hartmann }
```

Karl Amadeus HARTMANN

Karl Amadeus HARTMANN

When the entire content of a `\markup` expression is a markup list, it is implicitly typeset using the `\line` command. Thus, elements are stacked horizontally and grouped as a single, unbreakable text block. The `\markuplist` command acts differently: it expects a markup list, and prints the individual markups on the page, stacking them vertically, and allowing page breaks. The following example illustrates this difference.

```
\markup \box \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.
}

\markuplist \box \wordwrap-lines {
  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.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
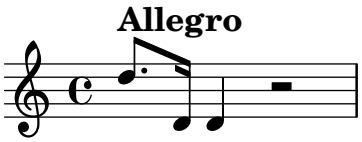
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea

commodo consequat.

Markups can be stored in variables, to be reused in any context where a markup is accepted. For example, such a variable can be directly attached to notes:

```
allegro = \markup \bold \large Allegro

{
  d''8.^\allegro
  d'16 d'4 r2
}
```



The `\etc` syntax allows to define custom shorthands usable as markup commands.

```
\markup reddish = \markup \with-color "tomato" \etc

\markup { molto \reddish intenso }
```

molto intenso

The inner workings of markup commands and how to implement more complex ones is explained in Section “Markup functions” in *Extending*.

See also

Notation Reference: Section 8.1.1 [Text objects overview], page 308, Section A.1 [Text markup commands], page 809, Section A.2 [Text markup list commands], page 886.

Extending LilyPond: Section “Markup functions” in *Extending*.

Installed Files: `scm/markup.scm`, `scm/define-markup-commands.scm`.

Snippets: Section “Text” in *Snippets*.

Internals Reference: Section “TextScript” in *Internals Reference*.
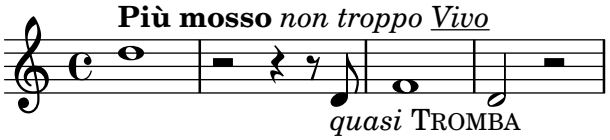
Known issues and warnings

Syntax error messages for markup mode can be confusing.

8.2.2 Selecting font and font size

Basic font switching is supported in markup mode:

```
\relative {
  d''1^\markup {
    \bold { Più mosso }
    \italic { non troppo \underline Vivo }
  }
  r2 r4 r8
  d,_\markup { \italic quasi \smallCaps Tromba }
  f1 d2 r
}
```

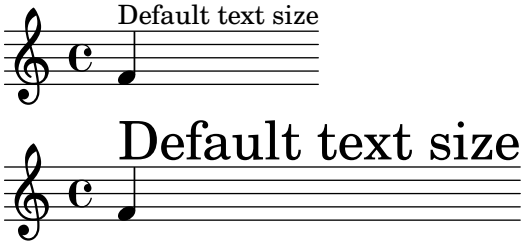


The global text size to be used in markups can be set with the `text-font-size` paper variable. This is useful to adjust to a different main font that might appear smaller or larger despite of having the same nominal font sizes. The value is given in points (without specifying a unit); the default value depends on the staff height and is computed as `(staff-height / 20 * 11)`.

See Section 36.2 [Distances and measurements], page 775, for more information on dimensions used by LilyPond.

```
\score {
  { f'^"Default text size" }
  \layout { text-font-size = 10 }
}

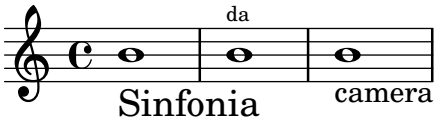
\score {
  { f'^"Default text size" }
  \layout { text-font-size = 20 }
}
```



The font size can be altered, relative to the global text size, in a number of different ways.

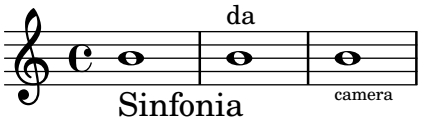
It can be set to predefined size.

```
\relative b' {
  b1_\markup { \huge Sinfonia }
  b1^\markup { \teeny da }
  b1-\markup { \normalsize camera }
}
```



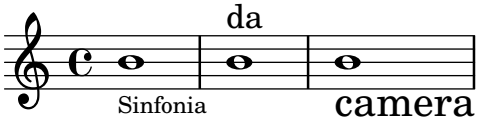
It can be set relative to its previous value.

```
\relative b' {
  b1_\markup { \larger Sinfonia }
  b1^\markup { \smaller da }
  b1-\markup { \magnify #0.6 camera }
}
```



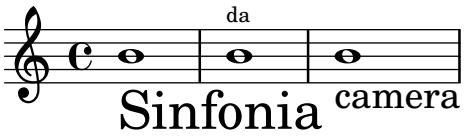
It can be increased or decreased relative to the value set by the global staff size.

```
\relative b' {
  b1_\markup { \fontsize #-2 Sinfonia }
  b1^\markup { \fontsize #1 da }
  b1-\markup { \fontsize #3 camera }
}
```



It can also be set to a fixed point size, regardless of the global staff size.

```
\relative b' {
  b1_\markup { \abs-fontsize #20 Sinfonia }
  b1^\markup { \abs-fontsize #8 da }
  b1-\markup { \abs-fontsize #14 camera }
}
```



If the text includes spaces, then it is best to put it all inside quote marks, so that the size of each space is appropriate for the size of the other characters.

```
\markup \fontsize #6 \bold { Sinfonia da camera }
\markup \fontsize #6 \bold { "Sinfonia da camera" }
```

Sinfonia da camera

Sinfonia da camera

Text may be printed as subscript or superscript. By default these are printed in a smaller size, but a normal size can be used as well:

```
\markup {
  \column {
    \line { 1 \super st movement }
    \line { 1 \normal-size-super st movement
      \sub { (part two) }  }
  }
}
```

1 st movement
1 st movement (part two)

The markup mode provides an easy way to select alternate font families. A serif font is selected by default unless specified otherwise; on the last line of the following example, there is no difference between the first and the second word.

```
\markup {
  \column {
    \line { Act \number 1 }
    \line { \sans { Scene I. } }
    \line { \typewriter { Verona. An open place. } }
    \line { Enter \serif Valentine and Proteus. }
  }
}
```

Act 1
Scene I.
Verona. An open place.
Enter 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 Section 3.1.3 [New dynamic marks], page 168, and Section 4.1.8 [Manual repeat marks], page 205.

When used inside a word, some font-switching or formatting commands may produce an unwanted blank space. This can easily be solved by concatenating the text elements together:

```
\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.1.1 [Font markup], page 809.

Defining custom font sets is also possible, as explained in Section 8.3 [Fonts], page 337.

Predefined commands

`\teeny`, `\tiny`, `\small`, `\normalsize`, `\large`, `\huge`, `\smaller`, `\larger`.

See also

Notation Reference: Section A.1.1 [Font markup], page 809, Section 3.1.3 [New dynamic marks], page 168, Section 4.1.8 [Manual repeat marks], page 205, Section 8.3 [Fonts], page 337.

Installed Files: `scm/define-markup-commands.scm`.

Snippets: Section "Text" in *Snippets*.

Internals Reference: Section "TextScript" in *Internals Reference*.

Known issues and warnings
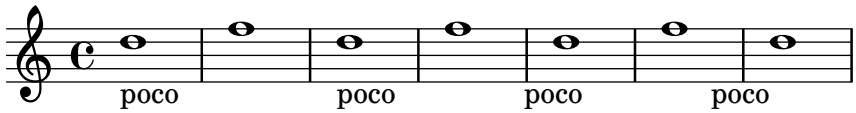
Using the font sizing commands `\teeny`, `\tiny`, `\small`, `\normalsize`, `\large`, and `\huge` will lead to inconsistent line spacing compared to using `\fontsize`.

8.2.3 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*.

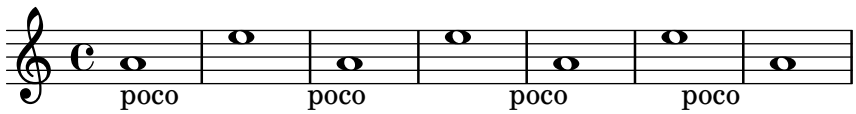
Markup objects may be aligned in different ways. By default, a text indication is aligned on its left edge: in the following example, there is no difference between the first and the second markup. That example also demonstrates various syntactically correct ways of placing the alignment commands:

```
\relative {
  d''1-\markup { poco }
  f
  d-\markup { \left-align poco }
  f
  d-\markup { \center-align { poco } }
  f
  d-\markup \right-align { poco }
}
```



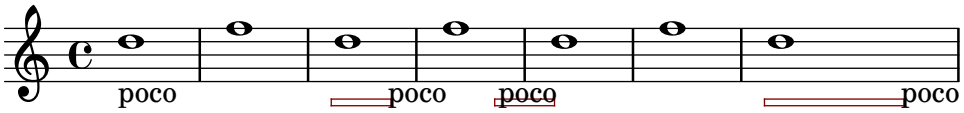
Horizontal alignment may be fine-tuned using a numeric value:

```
\relative {
  a'1-\markup { \halign #-1 poco }
  e'
  a,-\markup { \halign #0 poco }
  e'
  a,-\markup { \halign #0.5 poco }
  e'
  a,-\markup { \halign #2 poco }
}
```



Lastly, words and any other objects may be moved horizontally by preceding them with \hspace. Negative values are also supported and move any objects that follow into the opposite direction. Here, we put \hspace into a box to better show its effect.

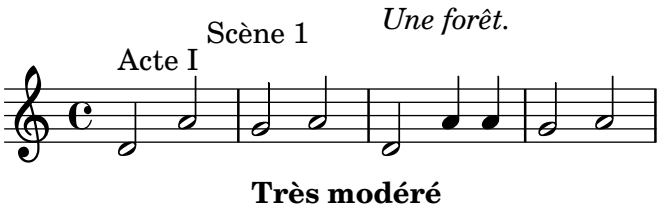
```
\relative {
  d''1-\markup { poco }
  f
  d-\markup \concat { \with-color #darkred \box \hspace #4 poco }
  f
  d-\markup \concat { \with-color #darkred \box \hspace #-4 poco }
  f
  d-\markup \concat { \with-color #darkred \box \hspace #10 poco }
}
```



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 Section 8.1.5 [Text marks], page 314.

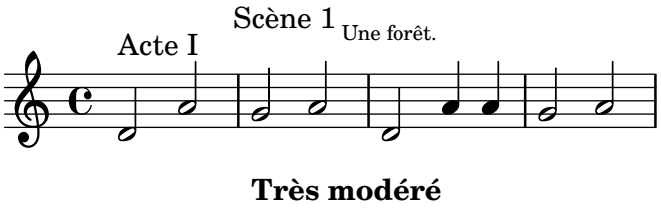
Vertical alignment can be set in a similar way. As stated above, markup objects can be moved as a whole; however, it is also possible to move specific elements inside a markup block.

```
\relative {
  d'2^\markup {
    Acte I
    \raise #2 { Scène 1 }
  }
  a'
  g_\markup {
    \lower #4 \bold { Très modéré }
  }
  a
  d,^\markup \raise #4 \italic {
    Une forêt.
  }
  a'4 a g2 a
}
```
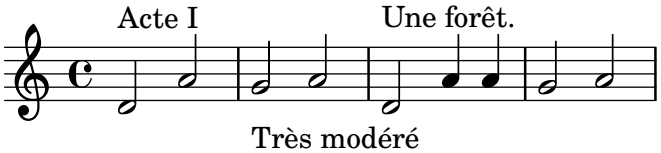


Some commands can affect both the horizontal and vertical alignment of text objects in markup mode:

```
\relative {
  d'2^\markup {
    Acte I
    \translate #'(2 . 2) "Scène 1"
  }
  a'
  g_\markup {
    \general-align #Y #5 \bold "Très modéré"
  }
  a
  d,^\markup \translate-scaled #'(-3 . 2) \teeny {
   "Une forêt."
  }
  a'4 a g2 a
}
```



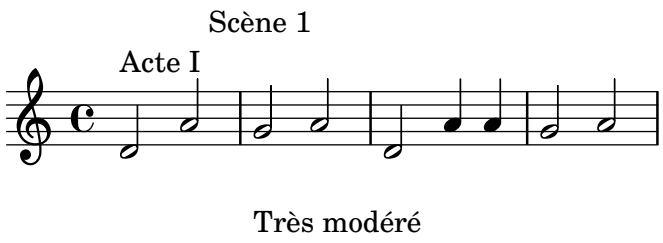
Note that \vspace can *not* be used in general to move arbitrary objects up or down within a \column markup due to the way the latter is implemented. The following naïve approach thus fails.

```
\relative {
  d'2^\markup {
    Acte I
    \column {
      \vspace #-2
      "Scène 1"
    }
  }
  a'
  g_\markup \column {
    \vspace #1
    "Très modéré"
  }
  a
  d,^\markup \column {
    "Une forêt."
    \vspace #2
  }
  a'4 a g2 a
}
```



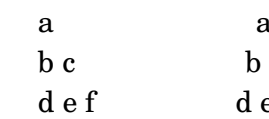
What actually works is to put \vspace between two objects that have non-empty extents.

```
\relative {
  d'2^\markup {
    Acte I
    \column {
      " "
      \vspace #-2
      "Scène 1"
    }
  }
  a'
  g_\markup \column {
    " "
    \vspace #1
    "Très modéré"
  }
  a
  d,^\markup \column {
    "Une forêt."
    \vspace #2
    " "
  }
  a'4 a g2 a
}
```



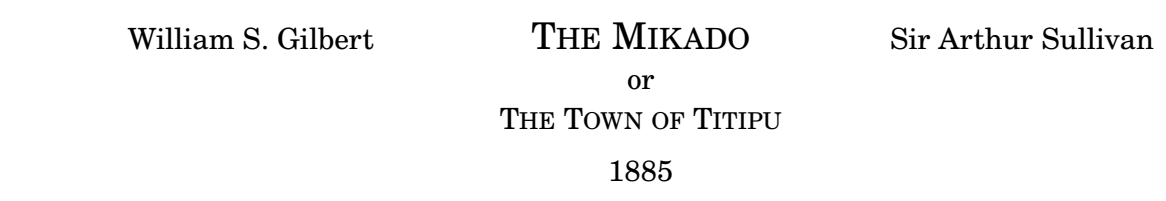
A markup object may include several lines of text. In the following example, each element or expression is placed on its own line, either left-aligned or centered:

```
\markup {
  \column {
    a
    "b c"
    \line { d e f }
  }
  \hspace #10
  \center-column {
    a
    "b c"
    \line { d e f }
  }
}
```



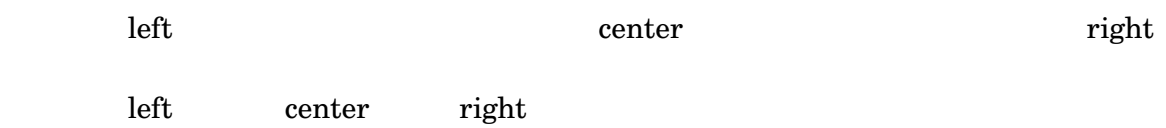
Similarly, a list of elements or expressions may be spread to fill the entire horizontal line width (if there is only one element, it will be centered on the page). These expressions can, in turn, include multi-line text or any other markup expression:

```
\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 }
}
```



Elements may be spread to fill any specified width by overriding the `line-width` property. By default it is set to `#f` which indicates the entire line:

```
\markup {
  \column {
    \fill-line { left center right }
    \null
    \override #'(line-width . 30)
    \fill-line { left center right }
  }
}
```



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.1.2 [Markup for text alignment], page 822.

See also

Learning Manual: Section “Moving objects” in *Learning Manual*.

Notation Reference: Section A.1.2 [Markup for text alignment], page 822, Section 8.1.5 [Text marks], page 314.

Installed Files: `scm/define-markup-commands.scm`.

Snippets: Section “Text” in *Snippets*.

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

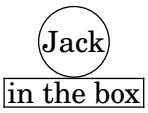
8.2.4 Graphic notation inside markup

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
  }
}
```

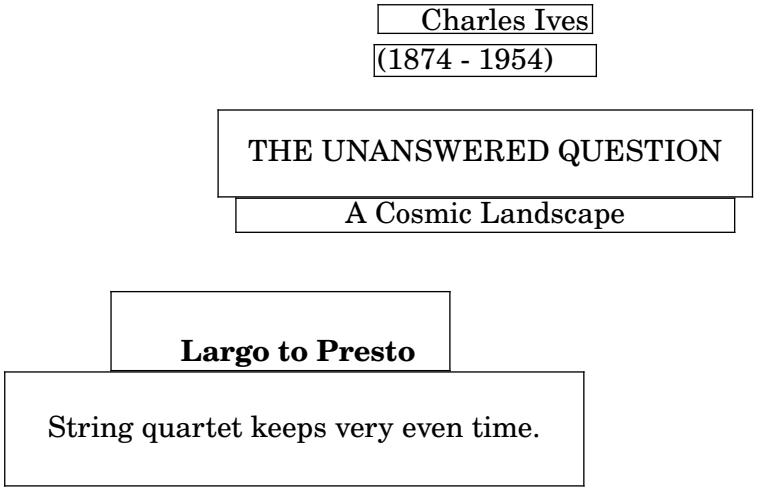


Erik Satie [1866 - 1925]

Prelude

Some commands may require increased padding around the text; this is achieved with some markup commands exhaustively described in Section A.1.2 [Markup for text alignment], page 822.

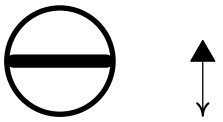
```
\markup \fill-line {
  \center-column {
    \box \pad-x-left #3 "Charles Ives"
    \box \pad-x-right #3 "(1874 - 1954)"
    \null
    \box \pad-markup #2 "THE UNANSWERED QUESTION"
    \box \pad-x #8 "A Cosmic Landscape"
    \null
  }
}
\markup \column {
  \line {
    \hspace #10
    \box \pad-to-box #'(-5 . 20) #'(0 . 5)
      \bold "Largo to Presto"
  }
  \box \pad-around #3 "String quartet keeps very even time."
}
```



Other graphic elements or symbols may be printed without requiring any text. As with any markup expression, such objects can be combined.

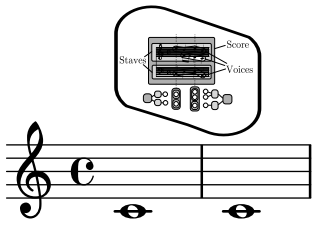
```
\markup {
  \combine
    \draw-circle #4 #0.4 ##f
    \filled-box #'(-4 . 4) #'(-0.5 . 0.5) #1
  \hspace #5

  \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:

```
c'1^\markup {
  \combine
    \epsfile #X #10 "./context-example.eps"
    \with-dimensions #'(-0.5 . 10.5) #'(-2 . 8)
    \postscript "
      -0.3 1 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.1.3 [Graphical markup], page 840.

See also

Notation Reference: Section A.1.2 [Markup for text alignment], page 822, Section 36.3 [Dimensions], page 776, Chapter 7 [Editorial annotations], page 278, Section A.1.3 [Graphical markup], page 840.

Installed Files: `scm/define-markup-commands.scm`, `scm/stencil.scm`.

Snippets: Section "Text" in *Snippets*.

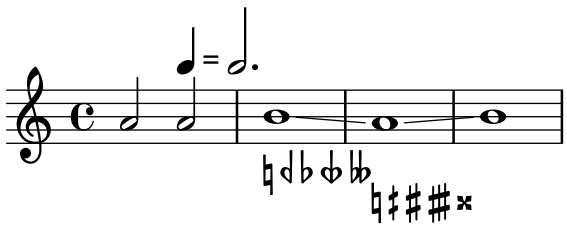
Internals Reference: Section "TextScript" in *Internals Reference*.

8.2.5 Music notation inside markup

Various musical notation elements may be added to a score, inside a markup object.

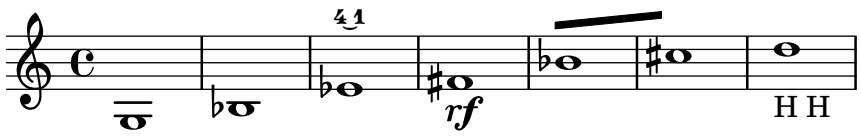
Notes and accidentals can be entered using markup commands:

```
a'2 a'^\markup {
  \note {4} #1
  =
  \note-by-number #1 #1 #1.5
}
b'1_\markup {
  \natural \semiflat \flat
  \sesquiflat \doubleflat
}
\glissando
a'1_\markup {
  \natural \semisharp \sharp
  \sesquisharp \doublesharp
}
\glissando b'
```



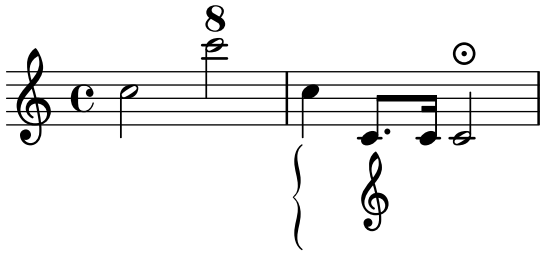
Other notation objects may also be printed in markup mode:

```
\relative {
  g1 bes
  ees\finger \markup \tied-lyric "4~1"
  fis_\markup { \dynamic rf }
  bes^\markup {
    \beam #8 #0.1 #0.5
  }
  cis
  d-\markup {
    \markalphabet #8
    \markletter #8
  }
}
```



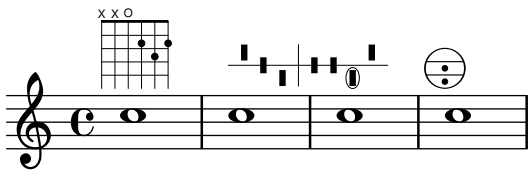
More generally, any available musical symbol may be included separately in a markup object, as demonstrated below; an exhaustive list of these symbols and their names can be found in Section B.8 [The Emmentaler font], page 907.

```
\relative {
  c''2
  c'^\markup { \musicglyph "eight" }
  c,4_\markup { \left-brace #40 }
  c,8._\markup { \musicglyph "clefs.G_change" }
  c16
  c2^\markup { \musicglyph "timesig.neomensural94" }
}
```



The markup mode also supports diagrams for specific instruments:
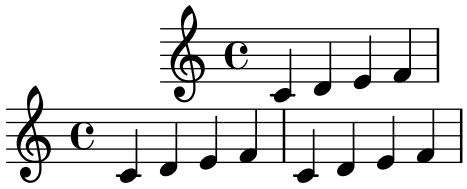
```
\relative {
  c''1^\markup {
    \fret-diagram-terse "x;x;o;2;3;2;"
  }
  c^\markup {
    \harp-pedal "^-v|--ov^"
  }
  c
  c^\markup {
    \combine
      \musicglyph "accordion.discant"
      \combine
        \raise #0.5 \musicglyph "accordion.dot"
        \raise #1.5 \musicglyph "accordion.dot"
  }
}
```



Such diagrams are documented in Section A.1.6 [Instrument-specific markup], page 864.

A whole score can even be nested inside a markup object:

```
\relative {
  c'4 d^\markup {
    \score {
      \relative { c'4 d e f }
    }
  }
  e f |
  c d e f
}
```



An exhaustive list of music notation related commands can be found in Section A.1.4 [Markup for music and musical symbols], page 851.

See also

Notation Reference: Section A.1.4 [Markup for music and musical symbols], page 851, Section B.8 [The Emmentaler font], page 907.

Installed Files: `scm/define-markup-commands.scm`, `scm/fret-diagrams.scm`, `scm/harp-pedals.scm`.

Snippets: Section "Text" in *Snippets*.

Internals Reference: Section "TextScript" in *Internals Reference*.

Known issues and warnings

Vertical spacing of a `\score` inside a markup object is controlled by `baseline-skip`. Any `\paper` settings are ignored.

8.3 Fonts

Fonts in LilyPond are handled by several libraries; two of them are of relevance to the user: *FontConfig* (`https://fontconfig.org`) is used to detect available fonts, and selected fonts are then rendered by *Pango* (`https://pango.org`) to display text strings.

This section shows how to access fonts in LilyPond, and how to change them in scores.

8.3.1 Unsupported font formats

A word of caution at the beginning: LilyPond creates its final PDF output using an intermediate PostScript file. As a consequence, some font formats cannot be used because PostScript, being an old standard and no longer receiving any updates, lacks support for them. Here is a list of font formats that won't work.

CFF2 In the last years this font format became the default for OpenType (`.otf`) fonts. While its predecessor (CFF, also using the extension `.otf`) is supported, the more compact CFF2 is not. LilyPond warns on the console if it encounters such a font, and the conversion to a PDF will either fail or the PDF will miss glyphs from this font.

Font Variations
: Such fonts usually have 'VF' somewhere in their name; unfortunately, they are not supported regardless of whether they are TrueType or OpenType Font Variations, and consequently filtered out in advance.

color fonts Not supported; this affects in particular scalable Emoji fonts. Some fonts, however, also contain B/W versions of the Emojis (as normal outline glyphs), and these should work just fine – provided they come in a supported font format.

OpenType collections
: This is a container format (usually with extension `.otc`) that holds multiple fonts in a single file. Today, it is often used as a last resort font, i.e., it gets used by the OS if no other font fits, for whatever reason.

bitmap fonts
: Not supported at all and automatically filtered out before LilyPond sees the list of available fonts. This includes some older Emoji color fonts.

Many fonts are available in both the TrueType and the OpenType format; if the `.otf` version fails, try the `.ttf` one. If you cannot find an alternative to a CFF2 font, try LilyPond's `-dbackend=cairo` command line option; this currently experimental backend bypasses the PostScript step and directly creates PDF output, thus being able to support CFF2 (and also OpenType collections).

8.3.2 Finding fonts

In addition to any font already installed on the operating system, more fonts may be added to the ones detected by FontConfig (and thus available in LilyPond scores) by the following commands:

```
#(ly:font-config-add-font "path/to/font-file")
#(ly:font-config-add-directory "path/to/directory/")
```

Both commands accept either absolute or relative paths, which makes it possible to compile a score on any system by simply distributing the relevant font files together with the LilyPond input files.

To verify that the desired fonts are found by FontConfig, use the command `#(ly:font-config-display-fonts)`, which prints the complete list of available fonts to the console log. It also shows the actual font names to be used with LilyPond; these may differ from the file names themselves. Alternatively, running `lilypond -dshow-available-fonts` in a terminal has the same effect.

8.3.3 Font families

Three generic aliases for text font families[1] are available: 'serif', 'sans', and 'typewriter'. Depending on the backend, these families get mapped to different font family aliases.

For the `svg` backend:

generic family	**SVG font family**
serif	serif
sans	sans-serif
typewriter	monospace

'serif', 'sans-serif', and 'monospace' are 'generic-family' in SVG and CSS specifications.

For other backends:

[1] In its simplest form, a *font family* usually contains fonts in roman, italic, bold, and bold italic styles.

generic family	default font family alias	font families contained in alias
serif	LilyPond Serif	C059, Century SchoolBook URW, Century Schoolbook L, TeX Gyre Schola, DejaVu Serif, ..., serif
sans	LilyPond Sans Serif	Nimbus Sans, Nimbus Sans L, TeX Gyre Heros, DejaVu Sans, ..., sans-serif
typewriter	LilyPond Monospace	Nimbus Mono PS, Nimbus Mono, Nimbus Mono L, TeX Gyre Cursor, DejaVu Sans Mono, ..., monospace

If a character does not exist in the appropriate font of the first listed family, the appropriate font of the next listed family gets used instead for that character.

Note that the URW font families distributed with LilyPond ('C059', 'Nimbus Sans', and 'Nimbus Mono PS') have a peculiarity: By default, in addition to the standard ligatures like 'fl' or 'ffi', they substitute the string 'Nr.' with the Numero Sign (U+2116) if the 'latn' script is selected. To circumvent this locally, insert a *zero-width non-joiner character* (ZWNJ, U+200C) between the 'N' and 'r' characters. To circumvent this globally, use the following code to make LilyPond always insert a ZWNJ character.
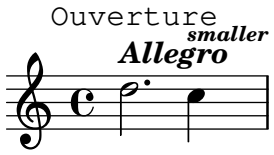
```
\paper {
  #(add-text-replacements!
      `(("Nr." . ,(format #f "N~ar." (ly:wide-char->utf-8 #x200C)))))
}
```

'LilyPond Serif', 'LilyPond Sans Serif', and 'LilyPond Monospace' are font family aliases defined in the additional FontConfig configuration file `00-lilypond-fonts.conf`, which can be usually found in directory `/usr/local/share/lilypond/2.27.5/fonts`, and which is used exclusively by LilyPond.

The exact mapping from these three font families to the actual fonts can be controlled via the `fonts` property, see Section 8.3.5 [Changing fonts], page 341.

Each font family may include different shapes and series. The following example demonstrates that, including code to also change the size. The value supplied to `font-size` is taken relative to the default font size.

```
\override Score.TextMark.font-family = #'typewriter
\textMark "Ouverture"
\override Voice.TextScript.font-shape = #'italic
\override Voice.TextScript.font-series = #'bold
d''2.^\markup "Allegro"
\override Voice.TextScript.font-size = -3
c''4^"smaller"
```



A similar syntax may be used in markup mode; however, in most cases it is preferable to use the simpler syntax explained in Section 8.2.2 [Selecting font and font size], page 324:

```
\markup {
  \column {
    \line {
      \override #'((font-shape . italic) (font-size . 4))
      Idomeneo,
    }
    \line {
      \override #'(font-family . typewriter) {
        \override #'(font-series . bold) re
        di
      }
      \override #'(font-family . sans) Creta
    }
  }
}
```

Idomeneo,
re di Creta

8.3.4 Font features

When using OpenType fonts, font features can be used.[2] Note that not all OpenType fonts have all features. If you request a feature that does not exist in the chosen font, the feature is simply ignored. The example below uses the font 'TeX Gyre Schola' (this is, the roman style of the family).

```
\paper {
  property-defaults.fonts.serif = "TeX Gyre Schola"
}

\markup "normal style: Hello HELLO"

\markup \caps "small caps: Hello"

\markup \override #'(font-features . ("smcp")) "true small caps: Hello"

\markup "normal number style: 0123456789"

\markup \override #'(font-features . ("onum"))
  "old number style: 0123456789"

\markup \override #'(font-features . ("salt 0"))
  "stylistic alternate 0: ϵϕπρθ"

\markup \override #'(font-features . ("salt 1"))
  "stylistic alternate 1: ϵϕπρθ"

\markup \override #'(font-features . ("onum" "smcp" "salt 1"))
  "multiple features: Hello 0123456789 ϵϕπρθ"
```

normal style: Hello HELLO

SMALL CAPS: HELLO

[2] Selecting OpenType font scripts and languages is not supported yet.

TRUE SMALL CAPS: HELLO

normal number style: 0123456789

old number style: 0123456789

stylistic alternate 0: εφπρθ

stylistic alternate 1: ϵϕϖϱϑ

MULTIPLE FEATURES: HELLO 0123456789 ϵϕϖϱϑ

For the full OpenType font feature list see `https://www.microsoft.com/typography/otspec/featurelist.htm`; for identifying features of OpenType fonts see `https://lists.gnu.org/archive/html/lilypond-devel/2017-08/msg00004.html`.

See also

Notation Reference: Section B.8 [The Emmentaler font], page 907, Section 8.2.5 [Music notation inside markup], page 335, Section 36.8 [Rotating objects], page 791, Section 8.2.2 [Selecting font and font size], page 324, Section A.1.1 [Font markup], page 809. Section 8.3.2 [Finding fonts], page 338, Section 8.3.3 [Font families], page 338, Section 8.3.5 [Changing fonts], page 341.

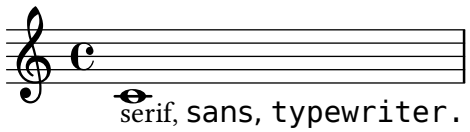
Snippets: Section "Text" in *Snippets*.

8.3.5 Changing fonts

It is possible to change the fonts used in LilyPond's default font families.

```
\paper {
  property-defaults.fonts.serif = "Linux Libertine O"
  property-defaults.fonts.sans = "DejaVu Sans"
  property-defaults.fonts.typewriter = "DejaVu Sans Mono"
}

\relative c'{
  c1-\markup {
    serif,
    \sans sans,
    \typewriter typewriter. }
}
```

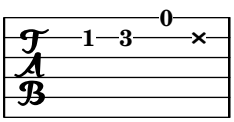


The same syntax can be used to change the music font; see Section 23.5 [Replacing the notation font], page 652.

To change the fonts used for one specific grob, override its `fonts` property. The following example changes the font for normal tablature "note heads" (which use the `serif` family) while keeping the default font for those that are drawn as a cross (which use the `music` family).

```
\layout {
  \override TabVoice.TabNoteHead.property-defaults.fonts.serif =
    "Linux Libertine O"
}

\new TabStaff { c' d' e' \deadNote c' }
```



To change the fonts in one specific markup, use the `\font-select` command:

```
\markup \font-select #'((serif . "Linux Libertine O")
                        (typewriter . "DejaVu Sans Mono")) {
  Copyright © John Doe \typewriter john@doe.org
}
```

Copyright ' John Doe john@doe.org

Known issues and warnings

There are rare situations where the FontConfig library is not able to unambiguously select a font: if the font family name and the font weight (i.e., the value of LilyPond's `font-series` subproperty) of two different fonts are identical, they are handled as being equal. An example for this situation is 'Minion Pro' and 'Minion Pro Display', which both have the family name 'Minion Pro' and the weight 'Regular'. The only solution to fix this is to make the unwanted font unknown to FontConfig.

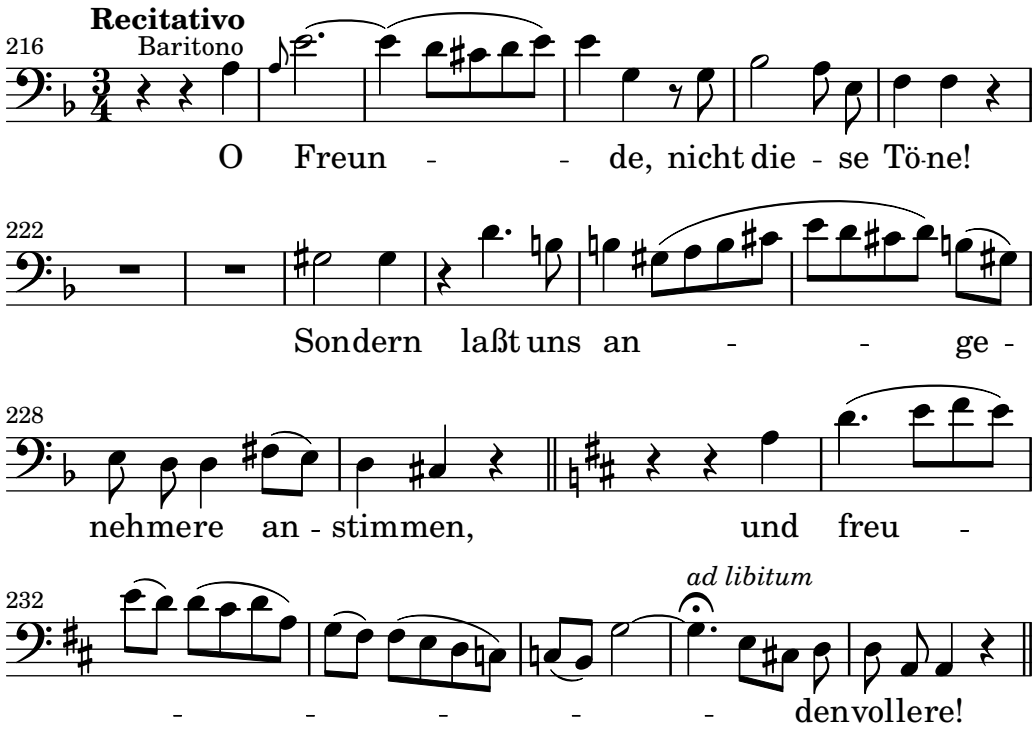
- If your fonts are in a separate directory not scanned by FontConfig by default, use `ly:font-config-add-font` instead of `ly:font-config-add-directory` to explicitly load the wanted font only.
- If the fonts are in directory searched by default (for example, `~/.fonts`), you have to remove the offending font so that FontConfig only finds the wanted font.

See also

Notation Reference: Section 8.3.2 [Finding fonts], page 338, Section 8.3.3 [Font families], page 338, Section 8.2.2 [Selecting font and font size], page 324, Section A.1.1 [Font markup], page 809, Section 23.5 [Replacing the notation font], page 652.

Specialist notation

9 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.

9.1 Common notation for vocal music

This section discusses issues common to most types of vocal music.

9.1.1 References for vocal music

This section indicates where to find details of notation issues that may arise in any type of vocal music.

- 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 (characters’ names, etc.). This syntax is described in Section 8.2.1 [Text markup introduction], page 320.
- *Ambitus* may be added at the beginning of vocal staves, as explained in Section 1.3.7 [Ambitus], page 40.
- Dynamic markings by default are placed below the staff, but in choral music they are usually placed above the staff in order to avoid the lyrics, as explained in Section 9.5.2 [Score layouts for choral], page 390.

See also

Music Glossary: Section “ambitus” in *Music Glossary*.

Learning Manual: Section “Setting simple songs” in *Learning Manual*.

Notation Reference: Section 8.2.1 [Text markup introduction], page 320, Section 1.3.7 [Ambitus], page 40, Section 9.5.2 [Score layouts for choral], page 390.

Snippets: Section “Vocal music” in *Snippets*.

9.1.2 Entering lyrics

Lyrics are entered in a special input mode, which can be introduced by the keyword `\lyricmode`, or by using `\addlyrics` or `\lyricsto`. In this special input mode, the input `d` is not parsed as the pitch *D*, but rather as a one-letter syllable of text. In other words, syllables are entered like notes but with pitches replaced by text.

For example:

```
\lyricmode { Three4 blind mice,2 three4 blind mice2 }
```

There are two main methods for specifying the horizontal placement of the syllables, either by specifying the duration of each syllable explicitly, as in the example above, or by leaving the lyrics to be aligned automatically to a melody or other voice of music, using `\addlyrics` or `\lyricsto`. The former method is described below in Section 9.1.5 [Manual syllable durations], page 352. The latter method is described in Section 9.1.4 [Automatic syllable durations], page 349.

A word or syllable of lyrics begins with an alphabetic character (plus some other characters, see below) and is terminated by any white space or a digit. Later characters in the syllable can be any character that is not a digit or white space.

Because any character that is not a digit or white space is regarded as part of the syllable, a word is valid even if it ends with `}`, which often leads to the following mistake:
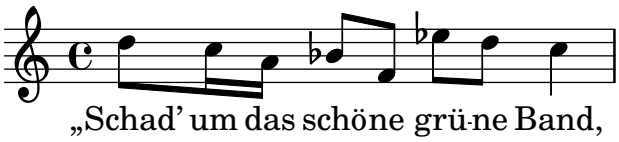
```
\lyricmode { lah lah lah}
```

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. Instead, braces should always be surrounded with white space:

```
\lyricmode { lah lah lah }
```
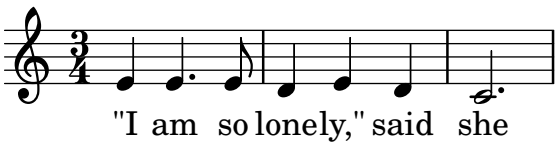
Punctuation, lyrics with accented characters, characters from non-English languages, or special characters (such as the heart symbol or slanted quotes), may simply be inserted directly into the input file, providing it is saved with UTF-8 encoding. For more information, see Section 22.4 [Special characters], page 647.

```
\relative { d''8 c16 a bes8 f ees' d c4 }
\addlyrics { „Schad’ um das schö -- ne grü -- ne Band, }
```



Normal quotes may be used in lyrics, but they have to be preceded with a backslash character and the whole syllable has to be enclosed between additional quotes. For example,

```
\relative { \time 3/4 e'4 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 is one that begins with an alphabetic character, `_`, `?`, `!`, `:`, `'`, the control characters `^A` through `^F`, `^Q` through `^W`, `^Y`, `^^`, any 8-bit character with an ASCII code over 127, or a two-character combination of a backslash followed by one of `` ` ``, `'`, `"`, or `^`.

Great control over the appearance of lyrics comes from using `\markup` inside the lyrics themselves. For explanation of many options, see Section 8.2 [Formatting text], page 320.

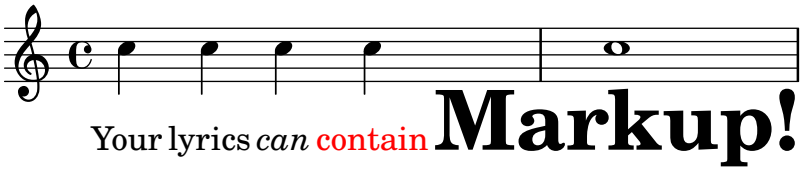
Selected snippets

Formatting lyrics syllables

The \markup command can be used in \lyricmode blocks to format individual syllables in lyrics.

```
mel = \relative c'' { c4 c c c c1 }
lyr = \lyricmode {
  Your lyrics \markup { \italic can }
  \markup { \with-color #red contain }
  \markup { \fontsize #8 \bold Markup! }
}

<<
  \new Voice = "melody" \mel
  \new Lyrics \lyricsto "melody" \lyr
>>
```



See also

Learning Manual: Section “Songs” in *Learning Manual*.

Notation Reference: Section 9.1.4 [Automatic syllable durations], page 349, Section 8.3 [Fonts], page 337, Section 8.2 [Formatting text], page 320, Chapter 19 [Input modes], page 589, Section 9.1.5 [Manual syllable durations], page 352, Section 22.4 [Special characters], page 647.

Internals Reference: Section “LyricText” in *Internals Reference*.

Snippets: Section “Text” in *Snippets*.

9.1.3 Aligning lyrics to a melody

Lyrics are interpreted in \lyricmode and printed in a Lyrics context, see Section 33.1 [Contexts explained], page 736.

```
\new Lyrics \lyricmode { ... }
```

Two variants of \lyricmode additionally set an associated context used to synchronize the lyric syllables to music. The more convenient \addlyrics immediately follows the musical content of the Voice context with which it should be synchronized, implicitly creating a Lyrics context of its own. The more versatile \lyricsto requires both specifying the associated Voice context by name and explicitly creating a containing Lyrics context. For details see Section 9.1.4 [Automatic syllable durations], page 349.

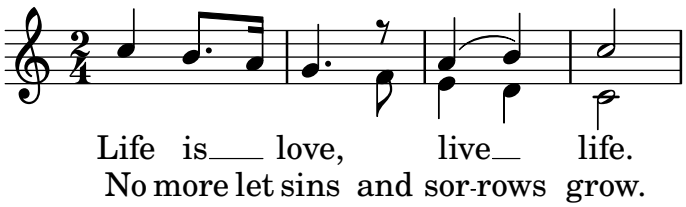
Lyrics can be aligned with melodies in two main ways:

- Lyrics can be aligned automatically, with the durations of the syllables being taken from another voice of music or (in special circumstances) an associated melody, using \addlyrics, \lyricsto, or by setting the associatedVoice property. For more details, see Section 9.1.4 [Automatic syllable durations], page 349.

```
<<
  \new Staff <<
    \time 2/4
    \new Voice = "one" \relative {
      \voiceOne
      c''4 b8. a16 g4. r8 a4 ( b ) c2
    }
    \new Voice = "two" \relative {
      \voiceTwo
       s2 s4. f'8 e4 d c2
    }
  >>

% takes durations and alignment from notes in "one"
  \new Lyrics \lyricsto "one" {
    Life is __ _ love, live __ life.
  }

% takes durations and alignment from notes in "one" initially
% then switches to "two"
  \new Lyrics \lyricsto "one" {
    No more let
    \set associatedVoice = "two"  % must be set one syllable early
    sins and sor -- rows grow.
  }
>>
```



The first stanza shows the normal way of entering lyrics.

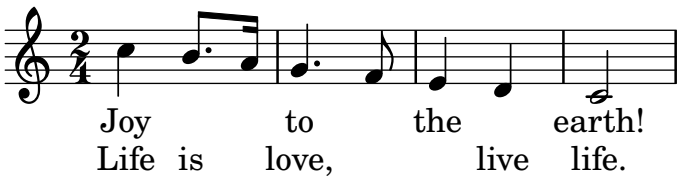
The second stanza shows how the voice from which the lyric durations are taken can be changed. This is useful if the words to different stanzas fit the notes in different ways and all the durations are available in Voice contexts. For more details, see Section 9.3 [Stanzas], page 377.

- Lyrics can be aligned independently of the duration of any notes if the durations of the syllables are specified explicitly, and entered with `\lyricmode`.

```
<<
  \new Voice = "one" \relative {
    \time 2/4
    c''4 b8. a16 g4. f8 e4 d c2
  }

% uses previous explicit duration of 2;
  \new Lyrics \lyricmode {
    Joy to the earth!
  }
```

```
% explicit durations, set to a different rhythm
  \new Lyrics \lyricmode {
    Life4 is love,2. live4 life.2
  }
>>
```



The first stanza is not aligned with the notes because the durations were not specified, and the previous value of 2 is used for each word.

The second stanza shows how the words can be aligned quite independently from the notes. This is useful if the words to different stanzas fit the notes in different ways and the required durations are not available in a music context. For more details see Section 9.1.5 [Manual syllable durations], page 352. This technique is also useful when setting dialogue over music; for examples showing this, see Section 9.6.5 [Dialogue over music], page 399.

See also

Learning Manual: Section "Aligning lyrics to a melody" in *Learning Manual*.

Notation Reference: Section 33.1 [Contexts explained], page 736, Section 9.1.4 [Automatic syllable durations], page 349, Section 9.3 [Stanzas], page 377, Section 9.1.5 [Manual syllable durations], page 352, Section 9.6.5 [Dialogue over music], page 399, Section 9.1.5 [Manual syllable durations], page 352.

Internals Reference: Section "Lyrics" in *Internals Reference*.

9.1.4 Automatic syllable durations

Lyrics can be automatically aligned to the notes of a melody in three ways:

- by specifying the named Voice context containing the melody with `\lyricsto`,
- by introducing the lyrics with `\addlyrics` and placing them immediately after the Voice context containing the melody,
- by setting the `associatedVoice` property, the alignment of the lyrics may be switched to a different named Voice context at any musical moment.
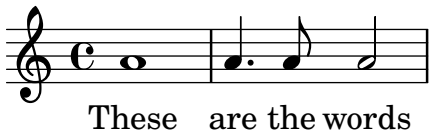
In all three methods hyphens can be drawn between the syllables of a word and extender lines can be drawn beyond the end of a word. For details, see Section 9.1.8 [Extenders and hyphens], page 357.

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 33.3 [Keeping contexts alive], page 742.

Using `\lyricsto`

Lyrics can be aligned under a melody automatically by specifying the named Voice context containing the melody with `\lyricsto`:

```
<<
  \new Voice = "melody" \relative {
    a'1 a4. a8 a2
  }
  \new Lyrics \lyricsto "melody" {
    These are the words
  }
>>
```



This aligns the lyrics to the notes of the named `Voice` context, which must already exist. Therefore normally the `Voice` context is specified first, followed by the `Lyrics` context. The lyrics themselves follow the `\lyricsto` command. The `\lyricsto` command invokes lyric mode automatically. By default, the lyrics are placed underneath the notes. For other placements, see Section 9.2.2 [Placing lyrics vertically], page 360.

Using `\addlyrics`

The `\addlyrics` command is just a convenient shortcut that can sometimes be used instead of having to set up the lyrics through a more complicated LilyPond structure.
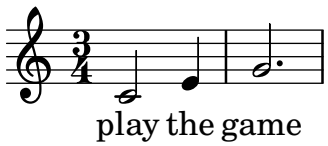
```
{ MUSIC }
\addlyrics { LYRICS }
```

is the same as

```
\new Voice = "blah" { MUSIC }
\new Lyrics \lyricsto "blah" { LYRICS }
```
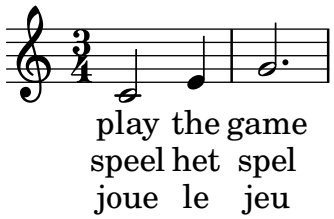
Here is an example,

```
{
  \time 3/4
  \relative { c'2 e4 g2. }
  \addlyrics { play the game }
}
```



More stanzas can be added by adding more `\addlyrics` sections:

```
{
  \time 3/4
  \relative { c'2 e4 g2. }
  \addlyrics { play the game }
  \addlyrics { speel het spel }
  \addlyrics { joue le jeu }
}
```



The command \addlyrics cannot handle polyphonic settings. Also, it cannot be used to associate lyrics to a TabVoice. For these cases one should use \lyricsto.

Using associatedVoice
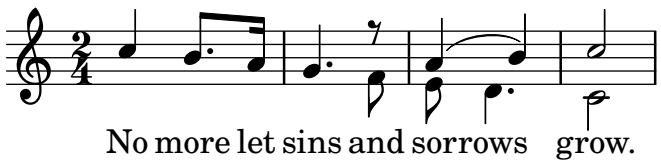
The melody to which the lyrics are being aligned can be changed by setting the associatedVoice property,

```
\set associatedVoice = "lala"
```

The value of the property (here: "lala") should be the name of a Voice context. For technical reasons, the \set command must be placed one syllable before the one to which the change in voice is to apply.

Here is an example demonstrating its use:

```
<<
  \new Staff <<
    \time 2/4
    \new Voice = "one" \relative {
      \voiceOne
      c''4 b8. a16 g4. r8 a4 ( b ) c2
    }
    \new Voice = "two" \relative {
      \voiceTwo
       s2 s4. f'8 e8 d4. c2
    }
  >>
% takes durations and alignment from notes in "one" initially
% then switches to "two"
  \new Lyrics \lyricsto "one" {
    No more let
    \set associatedVoice = "two"  % must be set one syllable early
    sins and sor -- rows grow.
  }
>>
```



See also

Notation Reference: Section 9.1.8 [Extenders and hyphens], page 357, Section 33.3 [Keeping contexts alive], page 742, Section 9.2.2 [Placing lyrics vertically], page 360.
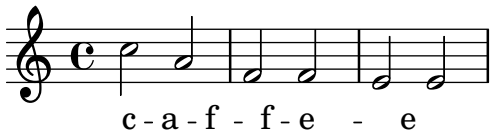
9.1.5 Manual syllable durations

In some complex vocal music, it may be desirable to place lyrics completely independently of notes. In this case do not use `\lyricsto` or `\addlyrics` and do not set `associatedVoice`. Syllables are entered like notes – but with pitches replaced by text – and the duration of each syllable is entered explicitly after the syllable.
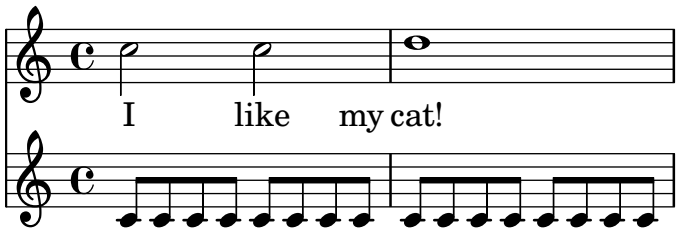
Hyphenated lines may be drawn between syllables as usual, but extender lines cannot be drawn when there is no associated voice.

Here are two examples:

```
<<
  \new Voice = "melody" \relative {
    c''2 a f f e e
  }
  \new Lyrics \lyricmode {
    c4. -- a -- f -- f -- e2. -- e
  }
>>
```
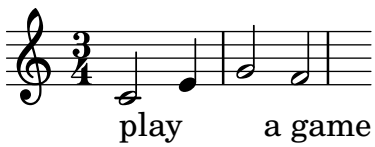



```
<<
  \new Staff {
    \relative {
      c''2 c2
      d1
    }
  }
  \new Lyrics {
    \lyricmode {
      I2 like4. my8 cat!1
    }
  }
  \new Staff {
    \relative {
      c'8 c c c c c c c
      c8 c c c c c c c
    }
  }
>>
```



This technique is useful when writing dialogue over music, see Section 9.6.5 [Dialogue over music], page 399.

To change syllable alignment, simply override the `self-alignment-X` property:

```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    c'2 e4 g2 f
  }
  \new Lyrics \lyricmode {
    \override LyricText.self-alignment-X = #LEFT
    play1 a4 game4
  }
>>
```



See also

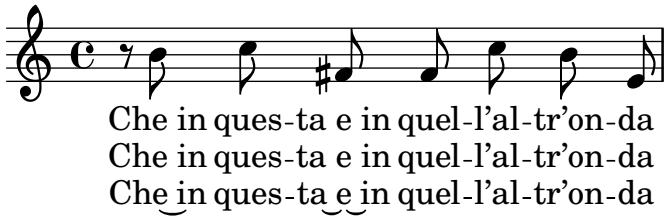
Notation Reference: Section 9.6.5 [Dialogue over music], page 399.

Internals Reference: Section “Lyrics” in *Internals Reference*, Section “Voice” in *Internals Reference*.

9.1.6 Multiple syllables to one note

In order to assign more than one syllable to a single note with spaces between the syllables, you can surround the phrase with quotes or use a _ character. Alternatively, you can use the tilde symbol (~) to get a lyric tie.

```
{
  \relative {
    \autoBeamOff
    r8 b' c fis, fis c' b e,
  }
  \addlyrics
  {
    % Ensure hyphens are visible
    \override LyricHyphen.minimum-distance = 1.0
    Che_in ques -- ta_e_in quel -- l'al -- tr'on -- da
  }
  \addlyrics { "Che in" ques -- "ta e in" quel -- l'al -- tr'on -- da }
  \addlyrics { Che~in ques -- ta~e~in quel -- l'al -- tr'on -- da }
}
```



See also

Internals Reference: Section “LyricCombineMusic” in *Internals Reference*.

9.1.7 Multiple notes to one syllable

Sometimes, particularly in medieval and Baroque music, several notes are sung on one syllable; this is called melisma, see Section "melisma" in *Music Glossary*. The syllable to a melisma is usually left-aligned with the first note of the melisma.
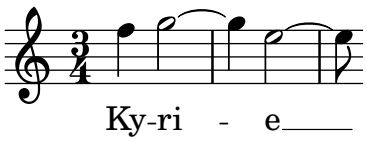
When a melisma occurs on a syllable other than the last one in a word, that syllable is usually joined to the following one with a hyphenated line. This is indicated by placing a double hyphen, `--`, immediately after the syllable.

Alternatively, when a melisma occurs on the last or only syllable in a word an extender line is usually drawn from the end of the syllable to the last note of the melisma. This is indicated by placing a double underscore, `__`, immediately after the word.

There are five ways in which melismata can be indicated:
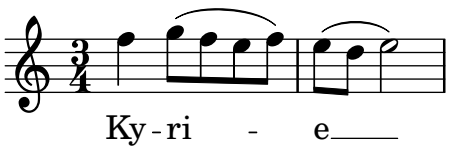
- Melismata are created automatically over notes which are tied together:

```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    f''4 g2 ~ |
    4 e2 ~ |
    8
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- e __
  }
>>
```



- Melismata can be created automatically from the music by placing slurs over the notes of each melisma. This is the usual way of entering lyrics:
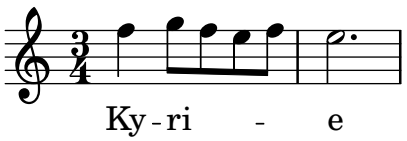
```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    f''4 g8 ( f e f )
    e8 ( d e2 )
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- e __
  }
>>
```



  Note that phrasing slurs do not affect the creation of melismata.

- Notes are considered a melisma if they are manually beamed, providing automatic beaming is switched off. See Section 2.4.2 [Setting automatic beam behavior], page 106.
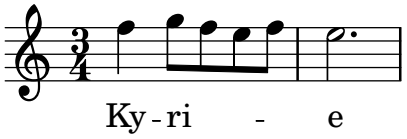
```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    \autoBeamOff
    f''4 g8[ f e f]
    e2.
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- e
  }
>>
```



Clearly this is not suited to melismata over notes which are longer than eighth notes.
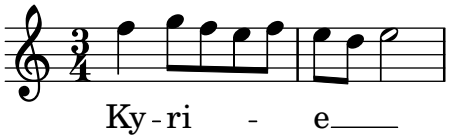
- An unslurred group of notes will be treated as a melisma if they are bracketed between `\melisma` and `\melismaEnd`.

```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    f''4 g8
    \melisma
    f e f
    \melismaEnd
    e2.
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- e
  }
>>
```
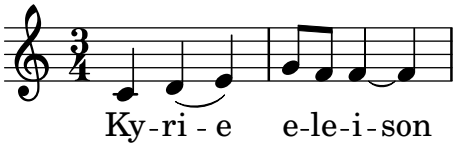


- A melisma can be defined entirely in the lyrics by entering a single underscore character, _, for every extra note that has to be added to the melisma.

```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    f''4 g8 f e f
    e8 d e2
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- _ _ _ e __ _ _
  }
>>
```



It is possible to have ties, slurs and manual beams in the melody without their indicating melismata. To do this, set `melismaBusyProperties`:

```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    \set melismaBusyProperties = #'()
    c'4 d ( e )
    g8 [ f ] f4 ~ 4
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- e e -- le -- i -- son
  }
>>
```
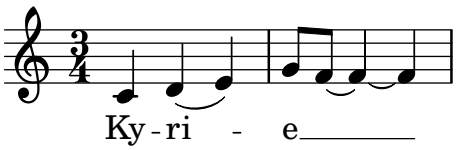


Other settings for `melismaBusyProperties` can be used to selectively include or exclude ties, slurs, and beams from the automatic detection of melismata; see `melismaBusyProperties` in Section "Tunable context properties" in *Internals Reference*.

Alternatively, if all melismata indications are to be ignored, `ignoreMelismata` may be set true; see Section 9.3.5 [Stanzas with different rhythms], page 382.

If a melisma is required during a passage in which `melismaBusyProperties` is active, it may be indicated by placing a single underscore in the lyrics for each note which should be included in the melisma:

```
<<
  \new Voice = "melody" \relative {
    \time 3/4
    \set melismaBusyProperties = #'()
    c'4 d ( e )
    g8 [ f ] ~ 4 ~ f
  }
  \new Lyrics \lyricsto "melody" {
    Ky -- ri -- _ e __ _ _ _
  }
>>
```



Predefined commands

`\autoBeamOff`, `\autoBeamOn`, `\melisma`, `\melismaEnd`.

See also

Musical Glossary: Section “melisma” in *Music Glossary*.

Learning Manual: Section “Aligning lyrics to a melody” in *Learning Manual*.

Notation Reference: Section 9.1.3 [Aligning lyrics to a melody], page 347, Section 9.1.4 [Automatic syllable durations], page 349, Section 2.4.2 [Setting automatic beam behavior], page 106, Section 9.3.5 [Stanzas with different rhythms], page 382.

Internals Reference: Section “Tunable context properties” in *Internals Reference*.

Known issues and warnings

Extender lines under melismata are not created automatically; they must be inserted manually with a double underscore.

9.1.8 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 manually as ‘ `__` ’ (note the spaces before and after the two underscore characters). It is also possible to create extender lines automatically by setting `autoExtenders` to `#t`.

> **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) properties of `LyricHyphen`.

By default a hyphen is not repeated after a system break when the next line begins with a new syllable. Setting the `after-line-breaking` property to `#t` allows hyphens to be drawn in such situations.

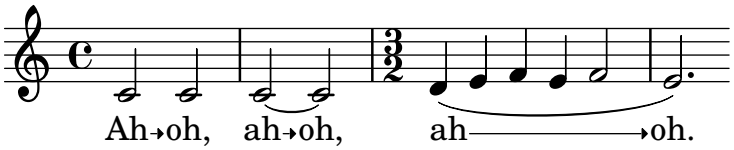
See also

Internals Reference: Section “LyricExtender” in *Internals Reference*, Section “LyricHyphen” in *Internals Reference*.

9.1.9 Gradual changes of vowel

Vowel transitions (gradual changes of vowel or sustained consonant), which may be indicated by arrows between syllables, are entered with the command `\vowelTransition` (see Gould pp. 452–453). The arrow shows the length of the transition, and it is by default always drawn (space is added if necessary in tightly engraved music). Ties between notes of unchanged pitch or slurs show that there is no new articulation, despite a change of vowel. The minimum length of the arrows may be adjusted with the `minimum-length` property of `VowelTransition`.

```
{
  c'2 c'
  \set melismaBusyProperties = #'()
  c'2 ~ c'
  \time 3/2
  d'4( e' f' e' f'2
  e'2.)
}
\addlyrics
{
  Ah \vowelTransition oh,
  ah \vowelTransition oh,
  ah \vowelTransition _ _ _ _
  oh.
}
```



See also

Musical Glossary: Section “vowel transition” in *Music Glossary*.
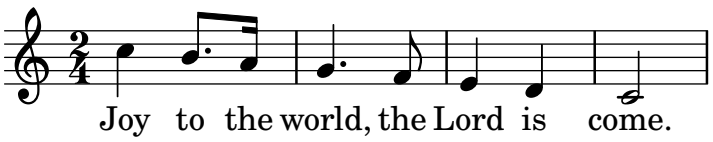
Internals Reference: Section “VowelTransition” in *Internals Reference*.

9.2 Techniques specific to lyrics

9.2.1 Working with lyrics and variables

Variables containing lyrics can be created, but the lyrics must be entered in lyric mode:

```
musicOne = \relative {
  c''4 b8. a16 g4. f8 e4 d c2
}
verseOne = \lyricmode {
  Joy to the world, the Lord is come.
}
\score {
  <<
    \new Voice = "one" {
      \time 2/4
      \musicOne
    }
    \new Lyrics \lyricsto "one" {
      \verseOne
    }
  >>
}
```
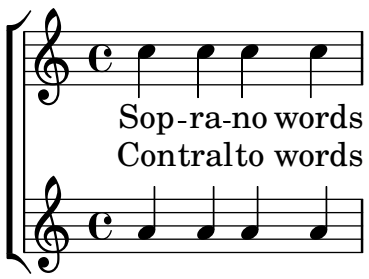


Durations do not need to be added if the variable is to be invoked with `\addlyrics` or `\lyricsto`.

For different or more complex orderings, the best way is to define the music and lyric variables first, then set up the hierarchy of staves and lyrics, omitting the lyrics themselves, and then add the lyrics using `\context` underneath. This ensures that the voices referenced by `\lyricsto` have always been defined earlier. For example:

```
sopranoMusic = \relative { c''4 c c c }
contraltoMusic = \relative { a'4 a a a }
sopranoWords = \lyricmode { Sop -- ra -- no words }
contraltoWords = \lyricmode { Con -- tral -- to words }

\score {
  \new ChoirStaff <<
    \new Staff {
      \new Voice = "sopranos" {
        \sopranoMusic
      }
    }
    \new Lyrics = "sopranos"
    \new Lyrics = "contraltos"
    \new Staff {
      \new Voice = "contraltos" {
        \contraltoMusic
      }
    }
    \context Lyrics = "sopranos" {
      \lyricsto "sopranos" {
        \sopranoWords
      }
    }
    \context Lyrics = "contraltos" {
      \lyricsto "contraltos" {
        \contraltoWords
      }
    }
  >>
}
```



See also
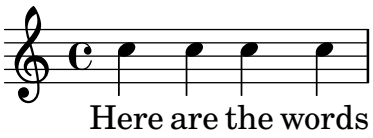
Notation Reference: Section 9.2.2 [Placing lyrics vertically], page 360.

Internals Reference: Section “LyricCombineMusic” in *Internals Reference*, Section “Lyrics” in *Internals Reference*.

9.2.2 Placing lyrics vertically

Depending on the type of music, lyrics may be positioned above the staff, below the staff, or between staves. Placing lyrics below the associated staff is the easiest, and can be achieved by simply defining the Lyrics context below the Staff context:

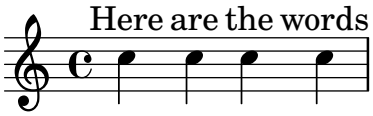
```
\score {
  <<
    \new Staff {
      \new Voice = "melody" {
        \relative { c''4 c c c }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Here are the words
      }
    }
  >>
}
```



Lyrics may be positioned above a staff using one of two methods. The simplest (and preferred) method is to use the same syntax as described above, explicitly specifying the position of the lyrics. The argument to `alignAboveContext` is the name of a `Staff` context or one of its siblings (using `Voice` doesn't work).

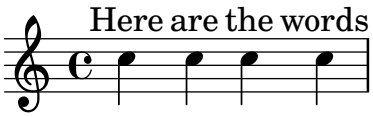
Use `alignBelowContext` to position lyrics below a staff. This is usually only needed to enforce alignment to a staff that is not the default.

```
\score {
  <<
    \new Staff = "staff" {
      \new Voice = "melody" {
        \relative { c''4 c c c }
      }
    }
    \new Lyrics \with { alignAboveContext = "staff" } {
      \lyricsto "melody" {
        Here are the words
      }
    }
  >>
}
```



Alternatively, a two-step process may be used. First the Lyrics context is declared (without any content) before the Staff and Voice contexts, then the `\lyricsto` command is placed after the Voice declaration it references by using `\context`, as follows:

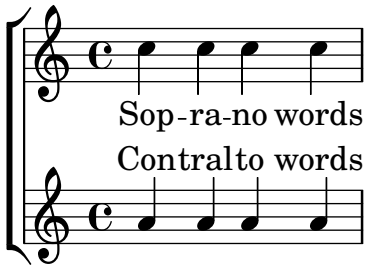
```
\score {
  <<
    \new Lyrics = "lyrics" \with {
      % lyrics above a staff should have this override
      \override VerticalAxisGroup.staff-affinity = #DOWN
    }
    \new Staff {
      \new Voice = "melody" {
        \relative { c''4 c c c }
      }
    }
    \context Lyrics = "lyrics" {
      \lyricsto "melody" {
        Here are the words
      }
    }
  >>
}
```



When there are two voices on separate staves the lyrics may be placed between the staves using either of these methods. Here is an example of the second method:

```
\score {
  \new ChoirStaff <<
    \new Staff {
      \new Voice = "sopranos" {
        \relative { c''4 c c c }
      }
    }
    \new Lyrics = "sopranos"
    \new Lyrics = "contraltos" \with {
      % lyrics above a staff should have this override
      \override VerticalAxisGroup.staff-affinity = #DOWN
    }
    \new Staff {
      \new Voice = "contraltos" {
        \relative { a'4 a a a }
      }
    }
    \context Lyrics = "sopranos" {
      \lyricsto "sopranos" {
        Sop -- ra -- no words
      }
    }
    \context Lyrics = "contraltos" {
      \lyricsto "contraltos" {
        Con -- tral -- to words
      }
    }
```

```
  >>
}
```



Other combinations of lyrics and staves may be generated by elaborating these examples, or by examining the templates in the Learning Manual, see Section "Vocal ensembles templates" in *Learning Manual*.

Selected snippets

Arranging separate lyrics on a single line

Sometimes you may want to put lyrics for different performers on a single line: where there is rapidly alternating text, for example. This snippet shows how it can be done with adjusting the `nonstaff-nonstaff-spacing` property of the `VerticalAxisGroup` grob.

```
\layout {
  \context {
    \Lyrics
    \override VerticalAxisGroup
              .nonstaff-nonstaff-spacing
              .minimum-distance = ##f
  }
}

aliceSings = \markup { \smallCaps "Alice" }
eveSings = \markup { \smallCaps "Eve" }

<<
  \new Staff <<
    \new Voice = "alice" {
      f'4^\aliceSings g' r2 |
      s1 |
      f'4^\aliceSings g' r2 |
      s1 | \break
      % ...
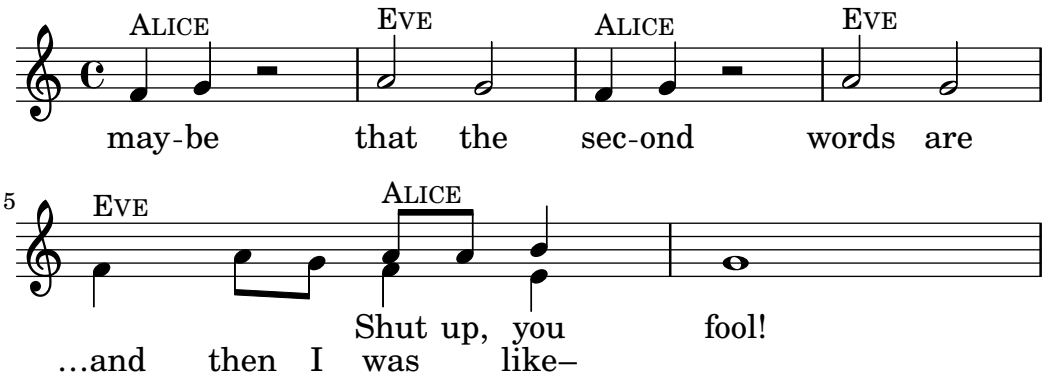
      \voiceOne
      s2 a'8^\aliceSings a' b'4 |
      \oneVoice
      g'1
    }
    \new Voice = "eve" {
      s1 |
      a'2^\eveSings g' |
      s1 |
      a'2^\eveSings g'
```

```
      % ...

      \voiceTwo
      f'4^\eveSings a'8 g' f'4 e' |
      \oneVoice
      s1
    }
  >>

  \new Lyrics \lyricsto "alice" {
    may -- be
    sec -- ond
    % ...
    Shut up, you fool!
  }

  \new Lyrics \lyricsto "eve" {
    that the
    words are
    % ...
    ...and then I was like--
  }
>>
```



See also

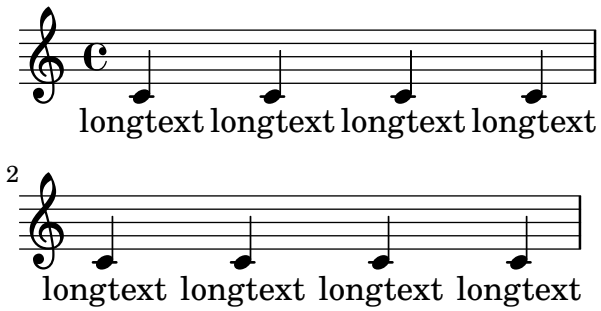
Learning Manual: Section "Vocal ensembles templates" in *Learning Manual*.

Notation Reference: Section 33.7 [Context layout order], page 755, Section 33.2 [Creating and referencing contexts], page 739.

9.2.3 Placing syllables horizontally

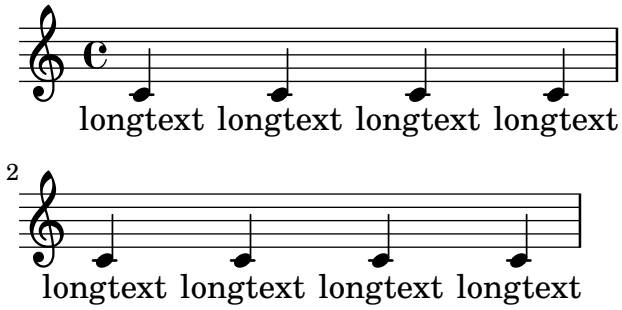
To increase the spacing between lyrics, set the `minimum-distance` property of `LyricSpace`.

```
\relative c' {
  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` block.

```
\score {
  \relative {
  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

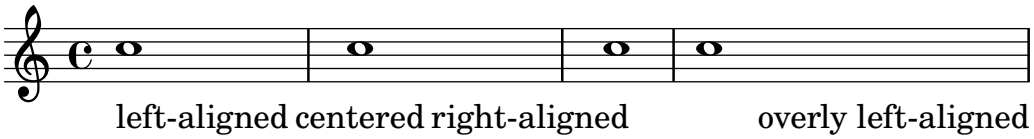
Lyrics alignment

Horizontal alignment for lyrics can be set by overriding the `self-alignment-X` property of the `LyricText` object. Value -1 means left-aligned, 0 centered, and 1 right-aligned. Alternatively, you can use the Scheme values `LEFT`, `CENTER`, and `RIGHT` instead of numbers. Other numeric values are possible, too – don't forget to add the '`#`' Scheme prefix for negative numbers!

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

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

```
\addlyrics {
  \once \override LyricText.self-alignment-X = #LEFT
  "left-aligned"
  \once \override LyricText.self-alignment-X = #CENTER
  "centered"
  \once \override LyricText.self-alignment-X = 1
  "right-aligned"
  \once \override LyricText.self-alignment-X = #-1.5
  "overly left-aligned"
}
```

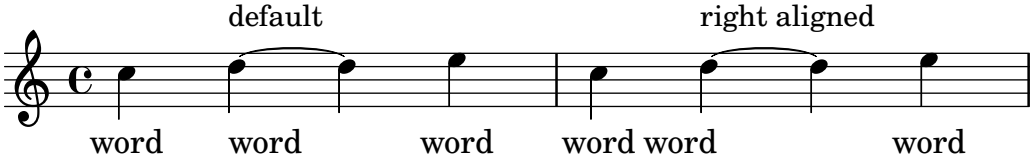


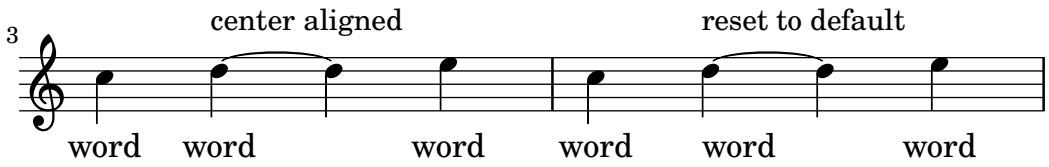
Aligning syllables with melisma

By default, lyrics syllables that start a melisma are left-aligned on their note. The alignment can be altered using the `lyricMelismaAlignment` property.

```
<<
  \new Staff {
    \new Voice = "vocal" \relative c'' {
      \override TextScript.staff-padding = #2
      c d~^\markup default  d e
      c d~^\markup "right aligned" d e
      c d~^\markup "center aligned" d e
      c d~^\markup "reset to default" d e
    }
  }
  \new Lyrics \lyricsto "vocal" {
    word word word
    \set lyricMelismaAlignment = #RIGHT
    word word word
    \set lyricMelismaAlignment = #CENTER
    word word word
    \unset lyricMelismaAlignment
    word word word
  }
>>

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





Known issues and warnings

Checking to make sure that text scripts and lyrics are within the margins requires additional calculations. To speed up processing slightly, this feature can be disabled:

```
\override Score.PaperColumn.keep-inside-line = ##f
```

To make lyrics avoid bar lines as well, use

```
\layout {
  \context {
    \Lyrics
      \consists Bar_engraver
      \consists Separating_line_group_engraver
      \hide BarLine
  }
}
```
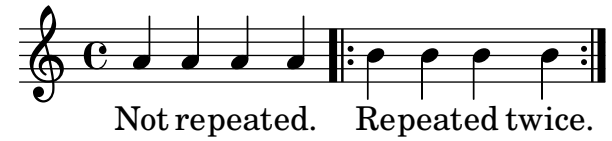
9.2.4 Lyrics and repeats

Simple repeats

Repeats in *music* are fully described elsewhere; see Chapter 4 [Repeats], page 189. This section explains how to add lyrics to repeated sections of music.
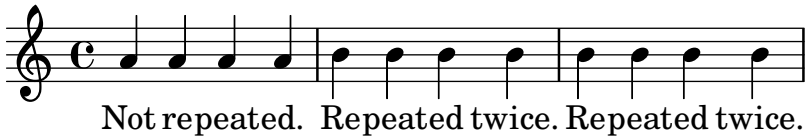
Lyrics to a section of music that is repeated should be surrounded by exactly the same repeat construct as the music, if the words are unchanged.

```
\score {
  <<
    \new Staff {
      \new Voice = "melody" {
        \relative {
          a'4 a a a
          \repeat volta 2 { b4 b b b }
        }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Not re -- peat -- ed.
        \repeat volta 2 { Re -- peat -- ed twice. }
      }
    }
  >>
}
```



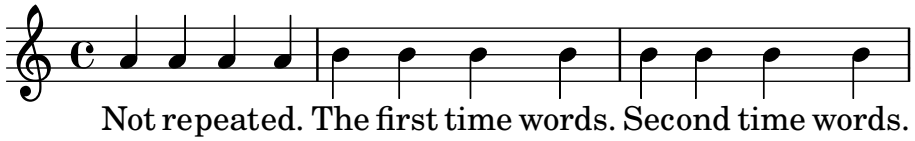
The words will then be correctly expanded if the repeats are unfolded.

```
\score {
  \unfoldRepeats {
    <<
      \new Staff {
        \new Voice = "melody" {
          \relative {
            a'4 a a a
            \repeat volta 2 { b4 b b b }
          }
        }
      }
      \new Lyrics {
        \lyricsto "melody" {
          Not re -- peat -- ed.
          \repeat volta 2 { Re -- peat -- ed twice. }
        }
      }
    >>
  }
}
```
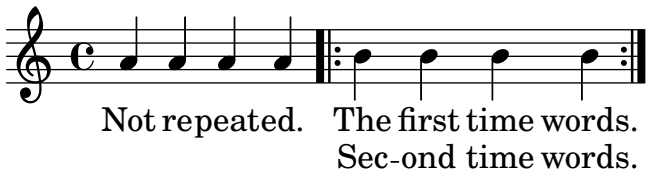


If the repeated section is to be unfolded and has different words, simply enter all the words:

```
\score {
  <<
    \new Staff {
      \new Voice = "melody" {
        \relative {
          a'4 a a a
          \*2 { b4 b b b }
        }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Not re -- peat -- ed.
        The first time words.
        Sec -- ond time words.
      }
    }
  >>
}
```



When the words to a repeated volta section are different, the words to each repeat must be entered in separate Lyrics contexts, correctly nested in parallel sections:

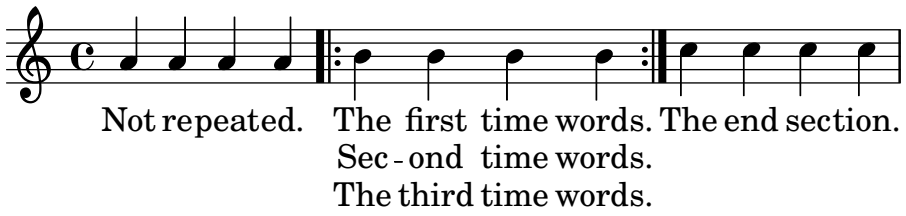
```
\score {
  <<
    \new Staff {
      \new Voice = "melody" {
        \relative {
          a'4 a a a
          \repeat volta 2 { b4 b b b }
        }
      }
    }
    \new Lyrics \lyricsto "melody" {
      Not re -- peat -- ed.
      <<
        { The first time words. }
        \new Lyrics {
          \set associatedVoice = "melody"
          Sec -- ond time words.
        }
      >>
    }
  >>
}
```



More verses may be added in a similar way:

```
\score {
  <<
    \new Staff {
      \new Voice = "singleVoice" {
        \relative {
          a'4 a a a
          \repeat volta 3 { b4 b b b }
          c4 c c c
        }
      }
    }
    \new Lyrics \lyricsto "singleVoice" {
      Not re -- peat -- ed.
      <<
        { The first time words. }
        \new Lyrics {
          \set associatedVoice = "singleVoice"
          Sec -- ond time words.
        }
        \new Lyrics {
```

```
        \set associatedVoice = "singleVoice"
        The third time words.
      }
    >>
    The end sec -- tion.
  }
>>
}
```

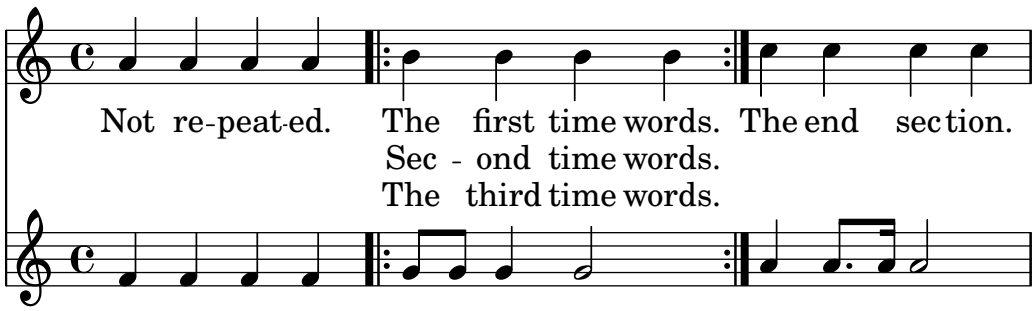


However, if this construct is embedded within a multi-staff context such as a `ChoirStaff` the lyrics of the second and third verses will appear beneath the bottom staff.

To position them correctly use `alignBelowContext`:

```
\score {
  <<
    \new Staff {
      \new Voice = "melody" {
        \relative {
          a'4 a a a
          \repeat volta 3 { b4 b b b }
          c4 c c c
        }
      }
    }
    \new Lyrics = "firstVerse" \lyricsto "melody" {
      Not re -- peat -- ed.
      <<
        { The first time words. }
        \new Lyrics = "secondVerse"
        \with { alignBelowContext = "firstVerse" } {
          \set associatedVoice = "melody"
          Sec -- ond time words.
        }
        \new Lyrics = "thirdVerse"
        \with { alignBelowContext = "secondVerse" } {
          \set associatedVoice = "melody"
          The third time words.
        }
      >>
      The end sec -- tion.
    }
    \new Voice = "harmony" {
      \relative {
        f'4 f f f
        \repeat volta 3 { g8 g g4 g2 }
        a4 a8. a16 a2
```
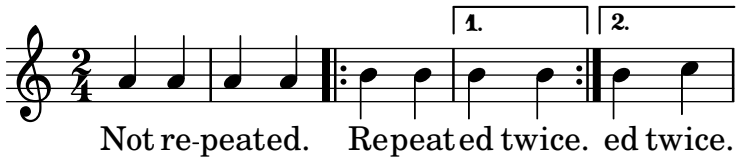
```
      }
    }
  >>
}
```



Repeats with alternative endings

If the words of the repeated section are the same, and none of the `\alternative` blocks start with a rest, exactly the same structure can be used for both the lyrics and music. This has the advantage that `\unfoldRepeats` will expand both music and lyrics correctly.

```
\score {
  <<
    \new Staff {
      \time 2/4
      \new Voice = "melody" {
        \relative {
          a'4 a a a
          \repeat volta 2 { b4 b }
          \alternative {
            \volta 1 { b b }
            \volta 2 { b c }
          }
        }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Not re -- peat -- ed.
        \repeat volta 2 { Re -- peat -- }
        \alternative {
          \volta 1 { ed twice. }
          \volta 2 { ed twice. }
        }
      }
    }
  >>
}
```
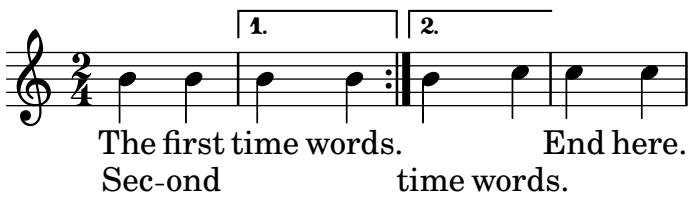


But when the repeated section has different words, or when one of the `\alternative` blocks starts with a rest, a repeat construct cannot be used around the words and `\skip` commands have to be inserted manually to skip over the notes in the alternative sections which do not apply.

Note: do not use an underscore, _, to skip notes – an underscore indicates a melisma, causing the preceding syllable to be left-aligned.

> **Note:** The `\skip` command must be followed by a number, but this number is ignored in lyrics which derive their durations from the notes in an associated melody through `\addlyrics` or `\lyricsto`. Each `\skip` skips a single note of any value, irrespective of the value of the following number.

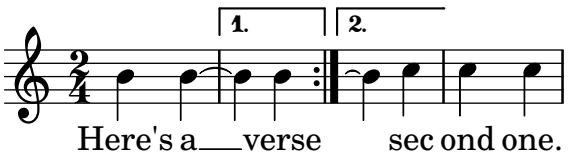
```
\score {
  <<
    \new Staff {
      \time 2/4
      \new Voice = "melody" {
        \relative {
          \repeat volta 2 { b'4 b }
          \alternative {
            \volta 1 { b b }
            \volta 2 { b c }
          }
          c4 c
        }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        The first time words.
        \*2 \skip 1
        End here.
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Sec -- ond
        \*2 \skip 1
        time words.
      }
    }
  >>
}
```



When a note is tied over into two or more alternative endings a tie is used to carry the note into the first alternative ending and a \repeatTie is used in the second and subsequent endings. This structure causes difficult alignment problems when lyrics are involved and increasing the length of the alternative sections so the tied notes are contained wholly within them may give a more acceptable result.

The tie creates a melisma into the first alternative, but not into the second and subsequent alternatives, so to align the lyrics correctly it is necessary to disable the automatic creation of melismata over the volta section and insert manual skips.

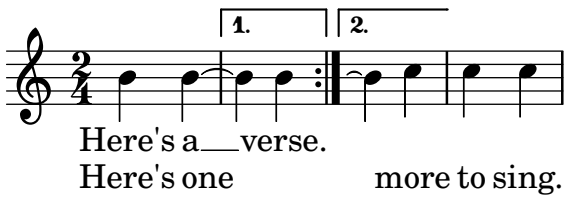
```
\score {
  <<
    \new Staff {
      \time 2/4
      \new Voice = "melody" {
        \relative {
          \set melismaBusyProperties = #'()
          \repeat volta 2 { b'4 b ~}
          \alternative {
            \volta 1 { b b }
            \volta 2 { b \repeatTie c }
          }
          \unset melismaBusyProperties
          c4 c
        }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        \repeat volta 2 { Here's a __ }
        \alternative {
          \volta 1 { \skip 1 verse }
          \volta 2 { \skip 1 sec }
        }
        ond one.
      }
    }
  >>
}
```



Note that if \unfoldRepeats is used around a section containing \repeatTie, the \repeatTie should be removed to avoid both types of tie being printed.

When the repeated section has different words a \repeat cannot be used around the lyrics and \skip commands need to be inserted manually, as before.
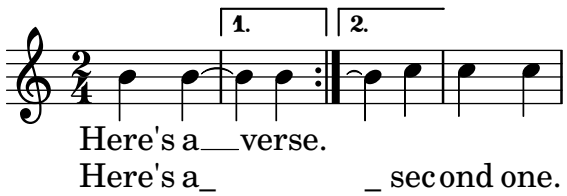
```
\score {
  <<
    \new Staff {
      \time 2/4
      \new Voice = "melody" {
        \relative {
          \repeat volta 2 { b'4 b ~}
          \alternative {
            \volta 1 { b b }
            \volta 2 { b \repeatTie c }
          }
          c4 c
        }
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Here's a __ verse.
        \*2 \skip 1
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Here's one
        \*2 \skip 1
        more to sing.
      }
    }
  >>
}
```



If you wish to show extenders and hyphens into and out of alternative sections these must be inserted manually.

```
\score {
  <<
    \new Staff {
      \time 2/4
      \new Voice = "melody" {
        \relative {
          \repeat volta 2 { b'4 b ~}
          \alternative {
            \volta 1 { b b }
            \volta 2 { b \repeatTie c }
          }
          c4 c
        }
```

```
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Here's a __ verse.
        \*2 \skip 1
      }
    }
    \new Lyrics {
      \lyricsto "melody" {
        Here's "a_"
        \skip 1
        "_" sec -- ond one.
      }
    }
  >>
}
```
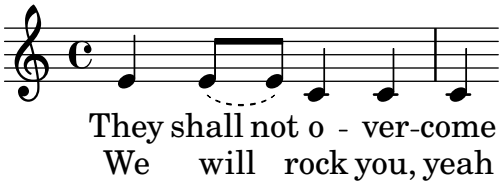


See also

Notation Reference: Section 33.3 [Keeping contexts alive], page 742, Chapter 4 [Repeats], page 189.

9.2.5 Divisi lyrics
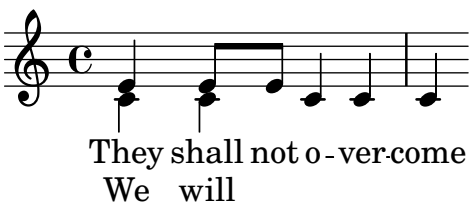
When just the words and rhythms of the two parts differ while the pitches remain the same, temporarily turning off the automatic detection of melismata and indicating the melisma in the lyrics may be the appropriate method to use:

```
\score {
  <<
    \new Voice = "melody" {
      \relative c' {
        \set melismaBusyProperties = #'()
        \slurDashed
        e4 e8( e) c4 c |
        \unset melismaBusyProperties
        c
      }
    }
    \new Lyrics \lyricsto "melody" {
      They shall not o -- ver -- come
    }
    \new Lyrics \lyricsto "melody" {
      We will _ rock you, yeah
    }
  >>
}
```



When both music and words differ it may be better to display the differing music and lyrics by naming voice contexts and attaching lyrics to those specific contexts:

```
\score {
  <<
    \new Voice = "melody" {
      \relative {
        <<
          {
            \voiceOne
            e'4 e8 e
          }
          \new Voice = "splitpart" {
            \voiceTwo
            c4 c
          }
        >>
        \oneVoice
        c4 c |
        c
      }
    }
    \new Lyrics \lyricsto "melody" {
      They shall not o -- ver -- come
    }
    \new Lyrics \lyricsto "splitpart" {
      We will
    }
  >>
}
```



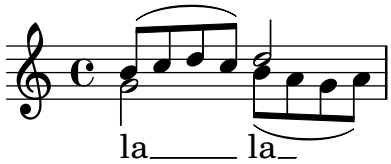
It is common in choral music to have a voice part split for several measures. The << {...} \\ {...} >> construct, where the two (or more) musical expressions are separated by double backslashes, might seem the proper way to set the split voices. This construct, however, will assign **all** the expressions within it to **NEW Voice contexts** which will result in *no lyrics* being set for them since the lyrics will be set to the original voice context – not, typically, what one wants. The temporary polyphonic passage is the proper construct to use, see section *Temporary polyphonic passages* in Section 5.2.1 [Single-staff polyphony], page 222.

9.2.6 Polyphony with shared lyrics

When two voices with different rhythms share the same lyrics, aligning the lyrics to one of the voices may lead to problems in the other voice. For example, the second lyric extender below is too short, since the lyrics are aligned only to the top voice:

```
soprano = \relative { b'8( c d c) d2 }
alto = \relative { g'2 b8( a g a) }
words = \lyricmode { la __ la __ }

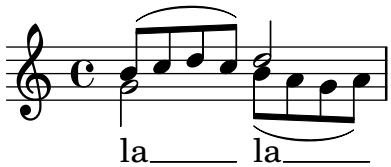
\new Staff <<
  \new Voice = "sopranoVoice" { \voiceOne \soprano }
  \new Voice { \voiceTwo \alto }
  \new Lyrics \lyricsto "sopranoVoice" \words
>>
```



To get the desired result, align the lyrics to a new `NullVoice` context containing a suitable combination of the two voices. The notes of the `NullVoice` context do not appear on the printed page, but can be used to align the lyrics appropriately:

```
soprano = \relative { b'8( c d c) d2 }
alto = \relative { g'2 b8( a g a) }
aligner = \relative { b'8( c d c) b( a g a) }
words = \lyricmode { la __ la __ }

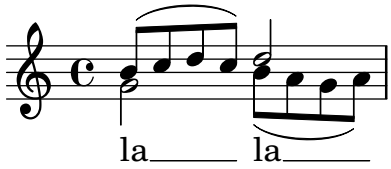
\new Staff <<
  \new Voice { \voiceOne \soprano }
  \new Voice { \voiceTwo \alto }
  \new NullVoice = "aligner" \aligner
  \new Lyrics \lyricsto "aligner" \words
>>
```



This method also can be used with the `\partCombine` function, which does not allow lyrics on its own:

```
soprano = \relative { b'8( c d c) d2 }
alto = \relative { g'2 b8( a g a) }
aligner = \relative { b'8( c d c) b( a g a) }
words = \lyricmode { la __ la __ }

\new Staff <<
  \new Voice \partCombine \soprano \alto
  \new NullVoice = "aligner" \aligner
  \new Lyrics \lyricsto "aligner" \words
>>
```



Known issues and warnings

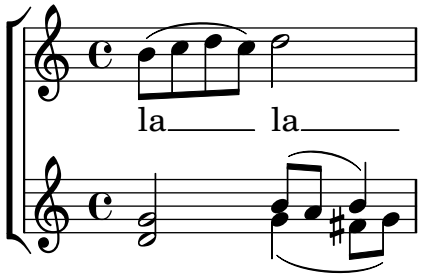
The `\addlyrics` function only works with `Voice` lyrics and so cannot be used with `NullVoice`.

The `\partCombine` function is described in Section 5.2.5 [Automatic part combining], page 233.

Lastly, this method can be used even when the voices are in different staves, and is not limited to only two voices:

```
soprano = \relative { b'8( c d c) d2 }
altoOne = \relative { g'2 b8( a b4) }
altoTwo = \relative { d'2 g4( fis8 g) }
aligner = \relative { b'8( c d c) d( d d d) }
words = \lyricmode { la __ la __ }

\new ChoirStaff \with {\accepts NullVoice } <<
  \new Staff \soprano
  \new NullVoice = "aligner" \aligner
  \new Lyrics \lyricsto "aligner" \words
  \new Staff \partCombine \altoOne \altoTwo
>>
```

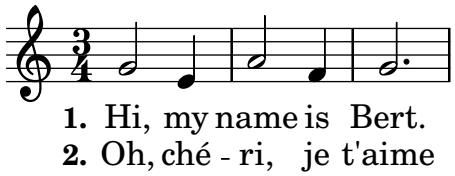


9.3 Stanzas

9.3.1 Adding stanza numbers

Stanza numbers can be added with the setting `\stanza` command, e.g.,

```
\new Voice \relative {
  \time 3/4 g'2 e4 a2 f4 g2.
} \addlyrics {
  \stanza "1."
  Hi, my name is Bert.
} \addlyrics {
  \stanza "2."
  Oh, ché -- ri, je t'aime
}
```

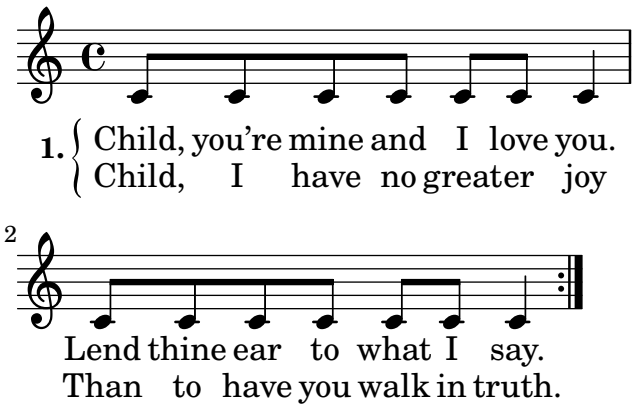


These numbers are put just before the start of the first syllable. Two lines of a stanza can also be grouped together, for example in case of a repeat with different lyrics:

```
stanzaOneOne = \lyricmode {
  \stanza \markup {
    \column {
      \vspace #.2
      \line { "1." \left-brace #30 }
    }
  }
  Child, you're mine and I love you.
  Lend thine ear to what I say.
}

stanzaOneThree = \lyricmode {
  Child, I have no great -- er joy
  Than to have you walk \bf\italic in truth.
}

\new Voice {
  \repeat volta 2 {
    c'8 c' c' c' c' c' c'4
    c'8 c' c' c' c' c' c'4
  }
}
\addlyrics { \stanzaOneOne }
\addlyrics { \stanzaOneThree }
```
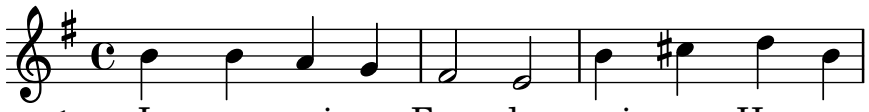


9.3.2 Stanza reminders

Sometimes it is desirable to repeat the stanza numbers at the beginning of each system. This can be accomplished by setting the `stanzaReminders` context property:
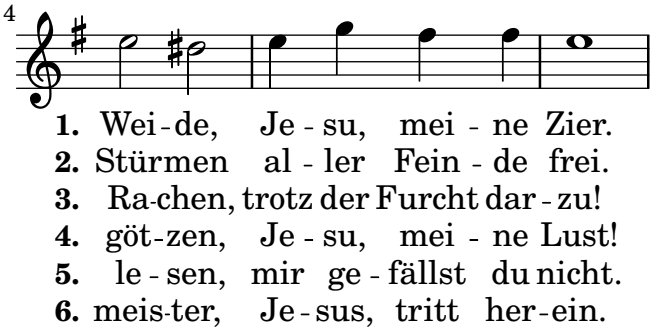
```
\layout {
  \set stanzaReminders = ##t
}

\new Voice \relative {
  \key e \minor
  b'4 b a g
  fis2 e
  b'4 cis d b \break
```

```
  e2 dis
  e4 g fis fis
  e1
}
\addlyrics {
  \stanza "1."
  Je -- su, mei -- ne Freu -- de,
  mei -- nes Her -- zens Wei -- de,
  Je -- su, mei -- ne Zier.
}
\addlyrics {
  \stanza "2."
  Un -- ter dei -- nem Schir -- men
  bin ich vor den Stür -- men
  al -- ler Fein -- de frei.
}
\addlyrics {
  \stanza "3."
  Trotz dem al -- ten Dra -- chen,
  trotz des To -- des Ra -- chen,
  trotz der Furcht dar -- zu!
}
\addlyrics {
  \stanza "4."
  Weg mit al -- len Schät -- zen!
  Du bist mein Er -- göt -- zen,
  Je -- su, mei -- ne Lust!
}
\addlyrics {
  \stanza "5."
  Gu -- te Nacht, o We -- sen,
  das die Welt er -- le -- sen,
  mir ge -- fällst du nicht.
}
\addlyrics {
  \stanza "6."
  Weicht, ihr Trau -- er -- geis -- ter,
  denn mein Freu -- den -- meis -- ter,
  Je -- sus, tritt her -- ein.
}
```
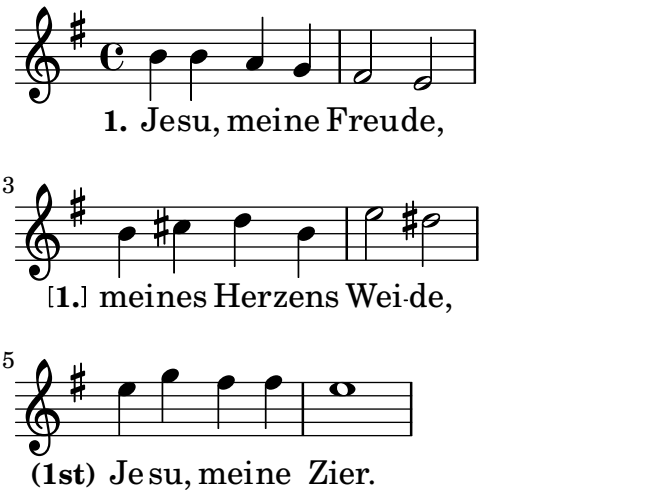


1. Je - su, mei - ne Freu - de, mei - nes Her zens
2. Un - ter dei - nem Schir men bin ich vor den
3. Trotz dem al - ten Dra - chen, trotz des To - des
4. Weg mit al - len Schätzen! Du bist mein Er-
5. Gu - te Nacht, o We - sen, das die Welt er-
6. Weicht, ihr Trau - er - geis - ter, denn mein Freuden-



By default, the stanza reminders are identical to their respective stanza numbers. By setting the `stanzaReminderText` property, this behavior can be changed, by either directly supplying a markup or by giving a function that accepts the original stanza number markup as an argument:
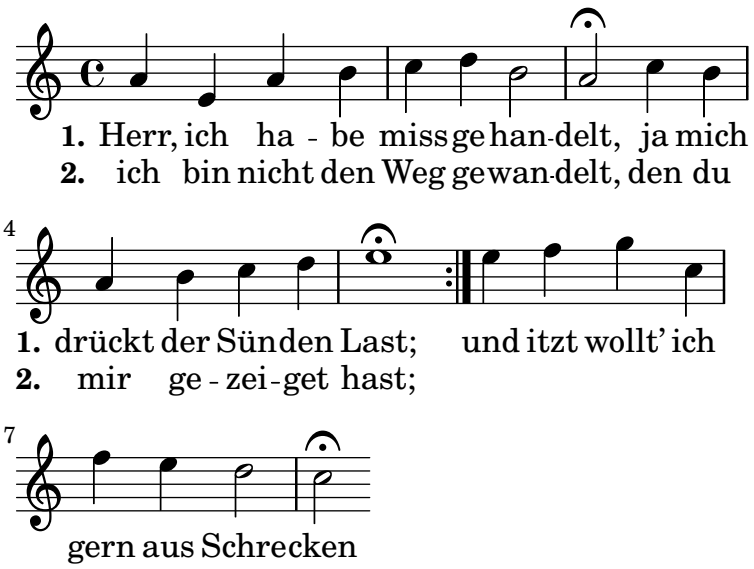
```
\new Voice \relative {
  \key e \minor
  b'4 b a g
  fis2 e \break
  b'4 cis d b
  e2 dis \break
  e4 g fis fis
  e1
}
\addlyrics {
  \stanza "1."
  \set stanzaReminders = ##t
  \set stanzaReminderText = #make-bracket-markup
  Je -- su, mei -- ne Freu -- de,
  mei -- nes Her -- zens Wei -- de,
  \set stanzaReminderText = "(1st)"
  Je -- su, mei -- ne Zier.
}
```



The `\noStanza` command resets the stanza number so that no subsequent automatic stanza reminders are printed:

```
\layout {
  \set stanzaReminders = ##t
}
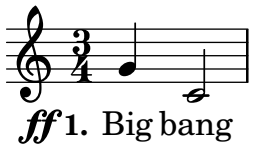
\new Voice \relative {
  \repeat volta 2 {
    a'4 e a b
    c d b2
    a\fermata c4 b
    \break
    a b c d
    e1\fermata
  }
  e4 f g c, \break
  f e d2
  c2 \fermata
}
\addlyrics {
  \stanza "1."
  Herr, ich ha -- be miss -- ge -- han -- delt,
  ja mich drückt der Sün -- den Last;
  \noStanza
  und itzt wollt' ich gern aus Schre -- cken
}
\addlyrics {
  \stanza "2."
  ich bin nicht den Weg ge -- wan -- delt,
  den du mir ge -- zei -- get hast;
}
```



9.3.3 Adding dynamics marks to stanzas

Stanzas differing in loudness may be indicated by putting a dynamics mark before each stanza. While there is no special support for such pre-stanza markings, they can easily be included in the stanza number itself:

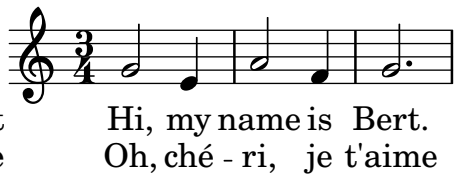
```
<<
  \new Voice = tune {
    \time 3/4
    g'4 c'2
  }
  \new Lyrics \lyricsto tune {
    \stanza \markup { \dynamic ff 1. }
     Big bang
  }
>>
```



9.3.4 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 \relative {
  \time 3/4 g'2 e4 a2 f4 g2.
} \addlyrics {
  \set vocalName = "Bert"
  Hi, my name is Bert.
} \addlyrics {
  \set vocalName = "Ernie"
  Oh, ché -- ri, je t'aime
}
```



9.3.5 Stanzas with different rhythms

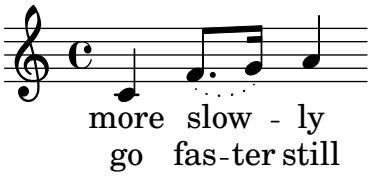
Often, different stanzas of one song are put to one melody in slightly differing ways. Such variations can still be captured with `\lyricsto`.

Ignoring melismata

One possibility is that the text has a melisma in one stanza, but multiple syllables in another. One solution is to make the faster voice ignore the melisma. This is done by setting `ignoreMelismata` in the Lyrics context.

```
<<
  \relative \new Voice = "lahlah" {
    \set Staff.autoBeaming = ##f
    c'4
    \slurDotted
    f8.[( g16])
    a4
```

```
  }
  \new Lyrics \lyricsto "lahlah" {
    more slow -- ly
  }
  \new Lyrics \lyricsto "lahlah" {
    go
    \set ignoreMelismata = ##t
    fas -- ter
    \unset ignoreMelismata
    still
  }
>>
```
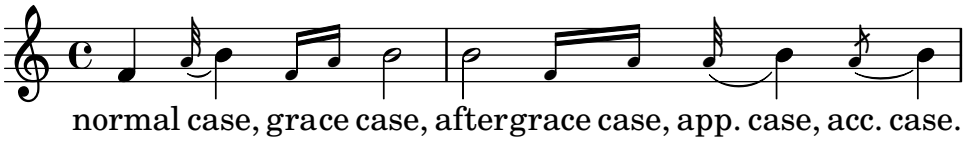


Known issues and warnings

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:
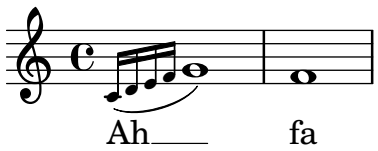
```
<<
  \new Voice = melody \relative {
    f'4 \appoggiatura a32 b4
    \grace { f16 a16 } b2
    \afterGrace b2 { f16[ a16] }
    \appoggiatura a32 b4
    \acciaccatura a8 b4
  }
  \new Lyrics
  \lyricsto melody {
    normal
    \set includeGraceNotes = ##t
    case,
    gra -- ce case,
    after -- grace case,
    \set ignoreMelismata = ##t
    app. case,
    acc. case.
  }
>>
```



Known issues and warnings

Like associatedVoice, includeGraceNotes needs to be set at latest one syllable before the one which is to be put under a grace note. In the case of a grace note at the very beginning of a piece of music, using a \with, or a \context block within \layout, is recommended:

```
<<
  \new Voice = melody \relative c' {
    \grace { c16( d e f }
    g1) f
  }
  \new Lyrics \with { includeGraceNotes = ##t }
  \lyricsto melody {
    Ah __ fa
  }
>>
```

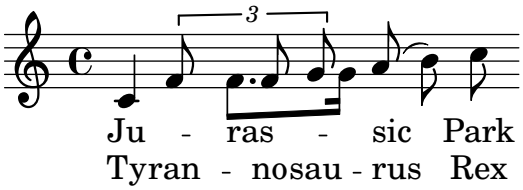


Switching to an alternative melody

More complex variations in setting lyrics to music are possible. The melody to which the lyrics are being set can be changed from within the lyrics by setting the associatedVoice property:

```
<<
  \relative \new Voice = "lahlah" {
    \set Staff.autoBeaming = ##f
    c'4
    <<
      \new Voice = "alternative" {
        \voiceOne
        \tuplet 3/2 {
          % show associations clearly.
          \override NoteColumn.force-hshift = -3
          f8 f g
        }
      }
      {
        \voiceTwo
        f8.[ g16]
        \oneVoice
      } >>
    a8( b) c
  }
  \new Lyrics \lyricsto "lahlah" {
    Ju -- ras -- sic Park
  }
  \new Lyrics \lyricsto "lahlah" {
    % Tricky: need to set associatedVoice
    % one syllable too soon!
    \set associatedVoice = "alternative" % applies to "ran"
    Ty --
```

```
    ran --
    no --
    \set associatedVoice = "lahlah" % applies to "rus"
    sau -- rus Rex
  } >>
```



The text for the first stanza is set to the melody called 'lahlah' in the usual way, but the second stanza is set initially to the `lahlah` context and is then switched to the `alternative` melody for the syllables 'ran' to 'sau' by the lines:

```
\set associatedVoice = "alternative" % applies to "ran"
Ty --
ran --
no --
\set associatedVoice = "lahlah" % applies to "rus"
sau -- rus Rex
```

Here, `alternative` is the name of the `Voice` context containing the triplet.

Note the placement of the `\set associatedVoice` command – it appears to be one syllable too early, but this is correct.

> **Note:** The `\set associatedVoice` command must be placed one syllable *before* the one at which the switch to the new voice is to occur. In other words, changing the associated Voice happens one syllable later than expected. This is for technical reasons, and it is not a bug.

9.3.6 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 several different ways to force line breaks when using `\markup`. For inputting a whole string you may use `\string-lines` with manually inserted `\n` or automatic line breaks as entered or `\wordwrap-string`. If inner formatting code is used a combination of `\line` and `\column` is recommended.

```
melody = \relative {
  e' d c d | e e e e |
  d d e d | c1 | }

text = \lyricmode {
  \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
  >>
}

\markup \column \string-lines
  "Verse 2. \n Everywhere that Mary went \n The lamb was sure to go."
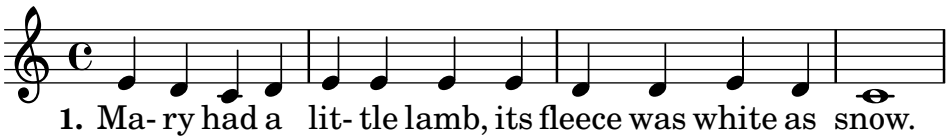
\markup \column \string-lines
  "Verse 3.
   All the children laughed and played,
   To see a lamb at school."

\markup \column {
  \line \italic { Verse 4. }
  \line { And so the teacher turned it out, }
  \line { But still it lingered near. }
}

\markup \wordwrap-string "
  Verse 5.

  Mary took it home again,

  It was against the rule."
```



Verse 2.
Everywhere that Mary went
The lamb was sure to go.

Verse 3.
All the children laughed and played,
To see a lamb at school.

Verse 4.
And so the teacher turned it out,
But still it lingered near.

Verse 5.
Mary took it home again,
It was against the rule.

9.3.7 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.
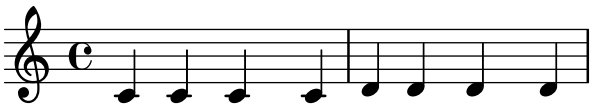
```
melody = \relative {
  c'4 c c c | d d d d
}

text = \lyricmode {
  \stanza "1." This is verse one.
  It has two lines.
}

\score {
  <<
    \new Voice = "one" { \melody }
    \new Lyrics \lyricsto "one" \text
  >>
  \layout { }
}

\markup {
  \fill-line {
    % moves the column off the left margin;
    % can be removed if space on the page is tight
    \hspace #0.1
    \column {
      \line { \bold "2."
        \column {
          "This is verse two."
          "It has two lines."
        }
      }
      % adds vertical spacing between verses
      \combine \null \vspace #0.1
      \line { \bold "3."
        \column {
          "This is verse three."
          "It has two lines."
        }
      }
    }
    % adds horizontal spacing between columns
    \hspace #0.1
    \column {
      \line { \bold "4."
        \column {
          "This is verse four."
          "It has two lines."
        }
      }
      % adds vertical spacing between verses
      \combine \null \vspace #0.1
      \line { \bold "5."
        \column {
          "This is verse five."
```

```
            "It has two lines."
          }
        }
      }
      % gives some extra space on the right margin;
      % can be removed if page space is tight
      \hspace #0.1
    }
  }
```



1. This is verse one. It has two lines.

2. This is verse two.
It has two lines.

3. This is verse three.
It has two lines.

4. This is verse four.
It has two lines.

5. This is verse five.
It has two lines.

See also

Internals Reference: Section “LyricText” in *Internals Reference*, Section “StanzaNumber” in *Internals Reference*.

9.4 Songs

9.4.1 References for songs

Songs are usually written on three staves with the melody for the singer on the top staff and two staves of piano accompaniment at the bottom. The lyrics of the first stanza are printed immediately underneath the top staff. If there are just a small number of further stanzas these can be printed immediately under the first one, but if there are more stanzas than can be easily accommodated there the second and subsequent stanzas are printed after the music as stand-alone text.

All the notational elements needed to write songs are fully described elsewhere:

- For constructing the staff layout, see Section 6.1 [Displaying staves], page 242.
- For writing piano music, see Chapter 10 [Keyboard and other multi-staff instruments], page 413.
- For writing the lyrics to a melody line, see Section 9.1 [Common notation for vocal music], page 345.
- For placing the lyrics, see Section 9.2.2 [Placing lyrics vertically], page 360.
- For entering stanzas, see Section 9.3 [Stanzas], page 377.
- Songs are frequently printed with the chording indicated by chord names above the staves. This is described in Section 15.2 [Displaying chords], page 513.
- To print fret diagrams of the chords for guitar accompaniment or accompaniment by other fretted instruments, see “Fret diagram markups” in Section 12.1 [Common notation for fretted strings], page 431.

See also

Learning Manual: Section “Songs” in *Learning Manual*.

Notation Reference: Section 9.1 [Common notation for vocal music], page 345, Section 15.2 [Displaying chords], page 513, Section 6.1 [Displaying staves], page 242, Chapter 10 [Keyboard and other multi-staff instruments], page 413, Section 9.2.2 [Placing lyrics vertically], page 360, Section 9.3 [Stanzas], page 377.

Snippets: Section "Vocal music" in *Snippets*.

9.4.2 Lead sheets

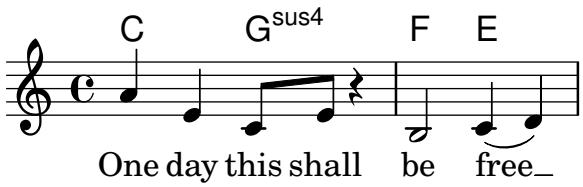
Lead sheets may be printed by combining vocal parts and 'chord mode'; this syntax is explained in Chapter 15 [Chord notation], page 508.

Selected snippets

Simple lead sheet

When put together, chord names, a melody, and lyrics form a lead sheet.

```
<<
  \chords { c2 g:sus4 f e }
  \new Staff \relative c'' {
    a4 e c8 e r4
    b2 c4( d)
  }
  \addlyrics { One day this shall be free __ }
>>
```



See also

Notation Reference: Chapter 15 [Chord notation], page 508.

9.5 Choral

This section discusses notation issues that relate most directly to choral music. This includes anthems, part songs, oratorio, etc.

9.5.1 References for choral

Choral music is usually notated on two, three or four staves within a `ChoirStaff` group. Accompaniment, if required, is placed beneath in a `PianoStaff` group, which is usually reduced in size for rehearsal of *a cappella* choral works. The notes for each vocal part are placed in a `Voice` context, with each staff being given either a single vocal part (i.e., one `Voice`) or a pair of vocal parts (i.e., two `Voices`).

Words are placed in `Lyrics` contexts, either underneath each corresponding music staff, or one above and one below the music staff if this contains the music for two parts.

Several common topics in choral music are described fully elsewhere:

- An introduction to creating an SATB vocal score can be found in the Learning Manual, see Section "Four-part SATB vocal score" in *Learning Manual*. There is also a built-in template which simplifies the entry of SATB vocal music, see Section "Built-in templates" in *Learning Manual*.

- Several templates suitable for various styles of choral music can also be found in the Learning Manual, see Section "Vocal ensembles templates" in *Learning Manual*.
- For information about `ChoirStaff` and `PianoStaff` see Section 6.1.2 [Grouping staves], page 244.
- Shape note heads, as used in Sacred Harp and similar notation, are described in Section 1.4.3 [Shape note heads], page 46.
- When two vocal parts share a staff the stems, ties, slurs, etc., of the higher part will be directed up and those of the lower part down. To do this, use `\voiceOne` and `\voiceTwo`. See Section 5.2.1 [Single-staff polyphony], page 222.
- When a vocal part temporarily splits, you should use *Temporary polyphonic passages* (see Section 5.2.1 [Single-staff polyphony], page 222).

Predefined commands

`\oneVoice`, `\voiceOne`, `\voiceTwo`.

See also

Learning Manual: Section "Four-part SATB vocal score" in *Learning Manual*, Section "Vocal ensembles templates" in *Learning Manual*.

Notation Reference: Section 33.7 [Context layout order], page 755, Section 6.1.2 [Grouping staves], page 244, Section 1.4.3 [Shape note heads], page 46, Section 5.2.1 [Single-staff polyphony], page 222.

Snippets: Section "Vocal music" in *Snippets*.

Internals Reference: Section "ChoirStaff" in *Internals Reference*, Section "Lyrics" in *Internals Reference*, Section "PianoStaff" in *Internals Reference*.

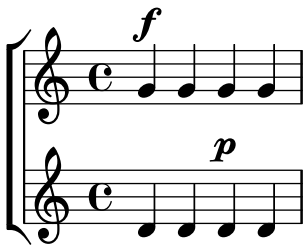
9.5.2 Score layouts for choral

Choral music containing four staves, with or without piano accompaniment, is usually laid out with two systems per page. Depending on the page size, achieving this may require changes to several default settings. The following settings should be considered:

- The global staff size can be modified to change the overall size of the elements of the score. See Section 27.2 [Setting the staff size], page 685.
- The distances between the systems, the staves and the lyrics can all be adjusted independently. See Chapter 29 [Vertical spacing], page 697.
- The dimensions of the vertical layout variables can be displayed as an aid to adjusting the vertical spacing. This and other possibilities for fitting the music onto fewer pages are described in Chapter 31 [Fitting music onto fewer pages], page 729.
- If the number of systems per page changes from one to two it is customary to indicate this with a system separator mark between the two systems. See Section 6.1.4 [Separating systems], page 249.
- For details of other page formatting properties, see Chapter 26 [Page layout], page 671.

Dynamic markings by default are placed below the staff, but in choral music they are usually placed above the staff in order to avoid the lyrics. The predefined command `\dynamicUp` does this for the dynamic markings in a single `Voice` context. If there are many `Voice` contexts this predefined command would have to be placed in every one. Alternatively its expanded form can be used to place all dynamic markings in the entire score above their respective staves, as shown here:

```
\score {
  \new ChoirStaff <<
    \new Staff {
      \new Voice {
        \relative { g'4\f g g g }
      }
    }
    \new Staff {
      \new Voice {
        \relative { d'4 d d\p d }
      }
    }
  >>
  \layout {
    \context {
      \Score
      \override DynamicText.direction = #UP
      \override DynamicLineSpanner.direction = #UP
    }
  }
}
```



Predefined commands

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

See also

Notation Reference: Section 31.2 [Changing spacing], page 730, Section 31.1 [Displaying spacing], page 729, Chapter 31 [Fitting music onto fewer pages], page 729, Chapter 26 [Page layout], page 671, Chapter 27 [Score layout], page 683, Section 6.1.4 [Separating systems], page 249, Section 27.2 [Setting the staff size], page 685, Chapter 28 [Breaks], page 689, Chapter 29 [Vertical spacing], page 697.

Internals Reference: Section “VerticalAxisGroup” in *Internals Reference*, Section “StaffGrouper” in *Internals Reference*.
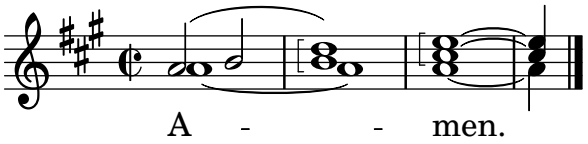
Selected snippets

Using a bracket to clarify divisi

The `\nonArpeggiato` command can be used to indicate the division of voices where there are no stems to provide the information. This is often seen in choral music.

```
\include "english.ly"

\score {
  \relative c'' {
    \key a \major
    \time 2/2
    <<
      \new Voice = "upper" <<
        {
          \voiceOne
          a2( b2
          <b d>1\nonArpeggiato)
          <cs e>\nonArpeggiato ~
          <cs e>4
          \fine
        }
        \addlyrics { \lyricmode { A -- men. } }
      >>
      \new Voice = "lower" {
        \voiceTwo
        a1 ~
        a
        a ~
        a4
        \fine
      }
    >>
  }
}
```



See also

Notation Reference: Section 3.3 [Expressive marks as lines], page 177.

9.6 Opera and stage musicals

The music, lyrics and dialogue to opera and stage musicals are usually set out in one or more of the following forms:

- A *Conductors' Score* containing the full orchestral and vocal parts, together with libretto cues if there are spoken passages.
- *Orchestral Parts* containing the music for the individual instruments of the orchestra or band.
- A *Vocal Score* containing all vocal parts with piano accompaniment. The accompaniment is usually an orchestral reduction, and if so the name of the original orchestral instrument is often indicated. Vocal scores sometimes includes stage directions and libretto cues.
- A *Vocal Book* containing just the vocal parts (no accompaniment), sometimes combined with the libretto.

- A *Libretto* containing the extended passages of spoken dialogue usually found in musicals, together with the words to the sung parts. Stage directions are usually included. LilyPond can be used to typeset libretti but as they contain no music alternative methods may be preferable.

The sections in the LilyPond documentation which cover the topics needed to create scores in the styles commonly found in opera and musicals are indicated in the References below. This is followed by sections covering those techniques which are peculiar to typesetting opera and musical scores.

9.6.1 References for opera and stage musicals

In addition to vocal and stage ensembles, most of the following notions may apply to nearly any orchestral and ensemble music:

- A conductors' score contains many grouped staves and lyrics. Ways of grouping staves is shown in Section 6.1.2 [Grouping staves], page 244. To nest groups of staves see Section 6.1.3 [Nested staff groups], page 248.
- The printing of empty staves in conductors' scores and vocal scores is often suppressed. To create such a "Frenched score" see Section 6.2.3 [Hiding staves], page 257.
- Writing orchestral parts is covered in Section 6.3 [Writing parts], page 262. Other sections in the Specialist notation chapter may be relevant, depending on the orchestration used. Many instruments are transposing instruments, see Section 1.3.4 [Instrument transpositions], page 30.
- If the number of systems per page changes from page to page it is customary to separate the systems with a system separator mark. See Section 6.1.4 [Separating systems], page 249.
- For details of other page formatting properties, see Chapter 26 [Page layout], page 671.
- Dialogue cues, stage directions and footnotes can be inserted, see Section 21.4 [Creating footnotes], page 613, and Chapter 8 [Text], page 308. Extensive stage directions can also be added with a section of stand-alone markups between two `\score` blocks, see Section 8.1.6 [Separate text], page 318.

See also

Musical Glossary: Section "Frenched score" in *Music Glossary*, Section "Frenched staves" in *Music Glossary*, Section "transposing instrument" in *Music Glossary*.

Notation Reference: Section 21.4 [Creating footnotes], page 613, Section 6.1.2 [Grouping staves], page 244, Section 6.2.3 [Hiding staves], page 257, Section 1.3.4 [Instrument transpositions], page 30, Section 6.1.3 [Nested staff groups], page 248, Chapter 26 [Page layout], page 671, Section 6.1.4 [Separating systems], page 249, Section 1.2.2 [Transpose], page 13, Section 6.3 [Writing parts], page 262, Section 8.1 [Writing text], page 308.

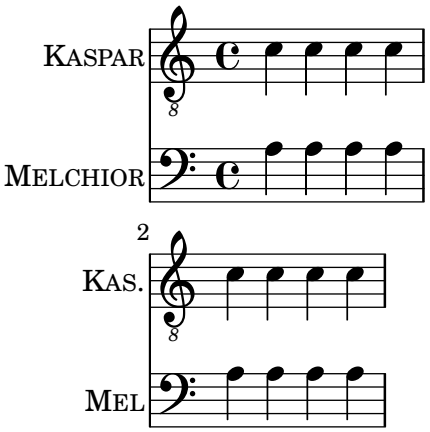
Snippets: Section "Vocal music" in *Snippets*.

9.6.2 Character names

Character names are usually shown to the left of the staff when the staff is dedicated to that character alone:
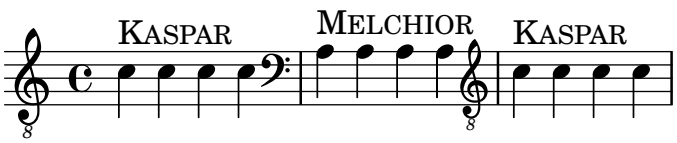
```
\score {
  <<
    \new Staff {
      \set Staff.vocalName = \markup \smallCaps Kaspar
      \set Staff.shortVocalName = \markup \smallCaps Kas.
      \relative {
        \clef "G_8"
```

```
        c'4 c c c
        \break
        c4 c c c
      }
    }
    \new Staff {
      \set Staff.vocalName = \markup \smallCaps Melchior
      \set Staff.shortVocalName = \markup \smallCaps Mel
      \clef "bass"
      \relative {
        a4 a a a
        a4 a a a
      }
    }
  >>
}
```



When two or more characters share a staff the character's name is usually printed above the staff at the start of every section applying to that character. This can be done with markup. Often a specific font is used for this purpose.

```
\relative c' {
  \clef "G_8"
  c4^\markup \fontsize #1 \smallCaps Kaspar
  c c c
  \clef "bass"
  a4^\markup \fontsize #1 \smallCaps Melchior
  a a a
  \clef "G_8"
  c4^\markup \fontsize #1 \smallCaps Kaspar
  c c c
}
```
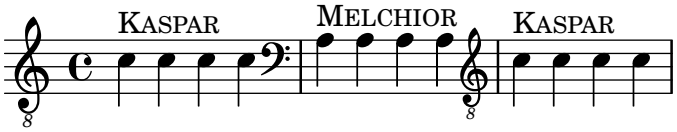


Alternatively, if there are many character changes, it may be easier to set up variables to hold the definitions for each character so that the switch of characters can be indicated easily and concisely.

```
kaspar = {
  \clef "G_8"
  \set Staff.shortVocalName = "Kas."
  \set Staff.midiInstrument = "voice oohs"
  <>^\markup \smallCaps "Kaspar"
}

melchior = {
  \clef "bass"
  \set Staff.shortVocalName = "Mel."
  \set Staff.midiInstrument = "choir aahs"
  <>^\markup \smallCaps "Melchior"
}

\relative c' {
  \kaspar
  c4 c c c
  \melchior
  a4 a a a
  \kaspar
  c4 c c c
}
```



See also

Learning Manual: Section "Organizing pieces with variables" in *Learning Manual*.

Notation Reference: Chapter 8 [Text], page 308, Section A.1 [Text markup commands], page 809.

9.6.3 Musical cues

Musical cues can be inserted in Vocal Scores, Vocal Books and Orchestral Parts to indicate what music in another part immediately precedes an entry. Also, cues are often inserted in the piano reduction in Vocal Scores to indicate what each orchestral instrument is playing. This aids the conductor when a full Conductors' Score is not available.

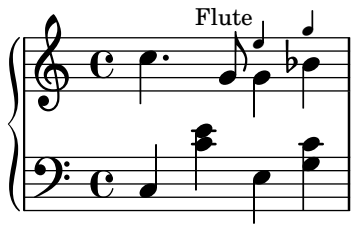
The basic mechanism for inserting cues is fully explained in the main text, see Section 6.3.2 [Quoting other voices], page 266, and Section 6.3.3 [Formatting cue notes], page 269. But when many cues have to be inserted, for example, as an aid to a conductor in a vocal score, the instrument name must be positioned carefully just before and close to the start of the cue notes. The following example shows how this is done.

```
flute = \relative {
  s4 s4 e'' g
}
\addQuote "flute" { \flute }

pianoRH = \relative {
  c''4. g8
  % position name of cue-ing instrument just before the cue notes,
  % and above the staff
```

```
  <>^\markup { \right-align { \tiny "Flute" } }
  \cueDuring "flute" #UP { g4 bes4 }
}
pianoLH = \relative { c4 <c' e> e, <g c> }

\score {
  \new PianoStaff <<
    \new Staff {
      \pianoRH
    }
    \new Staff {
      \clef "bass"
      \pianoLH
    }
  >>
}
```



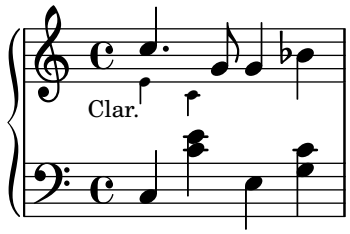
If a transposing instrument is being quoted the instrument part should specify its key so the conversion of its cue notes will be done automatically. The example below shows this transposition for a B-flat clarinet. The notes in this example are low on the staff so DOWN is specified in \cueDuring (so the stems are down) and the instrument name is positioned below the staff.

```
clarinet = \relative c' {
  \transposition bes
  fis4 d d c
}
\addQuote "clarinet" { \clarinet }

pianoRH = \relative c'' {
  \transposition c'
  % position name of cue-ing instrument below the staff
  <>_\markup { \right-align { \tiny "Clar." } }
  \cueDuring "clarinet" #DOWN { c4. g8 }
  g4 bes4
}
pianoLH = \relative { c4 <c' e> e, <g c> }

\score {
  <<
    \new PianoStaff <<
      \new Staff {
        \new Voice {
          \pianoRH
        }
      }
```

```
      \new Staff {
        \clef "bass"
        \pianoLH
      }
    >>
  >>
}
```



From these two examples it is clear that inserting many cues in a Vocal Score would be tedious, and the notes of the piano part would become obscured. However, as the following snippet shows, it is possible to define a music function to reduce the amount of typing and to make the piano notes clearer.

Selected snippets

Adding orchestral cues to a vocal score

This snippet shows one approach to simplify adding many orchestral cues to the piano reduction in a vocal score. The music function `\cueWhile` takes four arguments: the music from which the cue is to be taken, as defined by `\addQuote`, the name to be inserted before the cue notes, then either `UP` or `DOWN` to specify either `\voiceOne` with the name above the staff or `\voiceTwo` with the name below the staff, and finally the piano music in parallel with which the cue notes are to appear. The name of the cued instrument is positioned to the left of the cued notes. Many passages can be cued, but they cannot overlap each other in time.

```
cueWhile =
#(define-music-function
   (instrument name dir music)
   (string? string? ly:dir? ly:music?)
   #{
     \cueDuring $instrument #dir {
       \once \override TextScript.self-alignment-X = #RIGHT
       \once \override TextScript.direction = $dir
       <>-\markup { \tiny #name }
       $music
     }
   #})

flute = \relative c'' {
  \transposition c'
  s4 s4 e g
}
\addQuote "flute" { \flute }
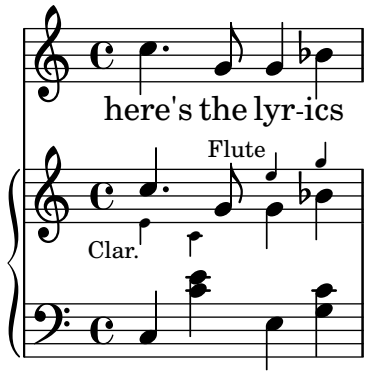
clarinet = \relative c' {
  \transposition bes
  fis4 d d c
```

```
}
\addQuote "clarinet" { \clarinet }

singer = \relative c'' { c4. g8 g4 bes4 }
words = \lyricmode { here's the lyr -- ics }

pianoRH = \relative c'' {
  \transposition c'
  \cueWhile "clarinet" "Clar." #DOWN { c4. g8 }
  \cueWhile "flute" "Flute" #UP { g4 bes4 }
}
pianoLH = \relative c { c4 <c' e> e, <g c> }

\score {
  <<
    \new Staff {
      \new Voice = "singer" {
        \singer
      }
    }
    \new Lyrics {
      \lyricsto "singer"
      \words
    }
    \new PianoStaff <<
      \new Staff {
        \new Voice {
          \pianoRH
        }
      }
      \new Staff {
        \clef "bass"
        \pianoLH
      }
    >>
  >>
}
```



See also

Musical Glossary: Section “cue-notes” in *Music Glossary*.

Notation Reference: Section 36.9 [Aligning objects], page 791, Section 36.1 [Direction and placement], page 774, Section 6.3.3 [Formatting cue notes], page 269, Section 6.3.2 [Quoting other voices], page 266, Section 22.3 [Using music functions], page 641.

Snippets: Section "Vocal music" in *Snippets*.

Internals Reference: Section "CueVoice" in *Internals Reference*.

Known issues and warnings

`\cueDuring` automatically inserts a `CueVoice` context and all cue notes are placed in that context. This means it is not possible to have two overlapping sequences of cue notes by this technique. Overlapping sequences could be entered by explicitly declaring separate `CueVoice` contexts and using `\quoteDuring` to extract and insert the cue notes.

9.6.4 Spoken music
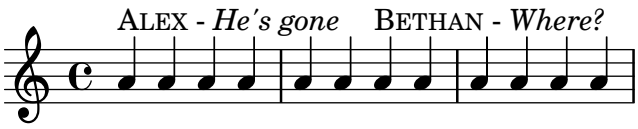
Such effects as 'parlato' or 'Sprechgesang' require performers to speak without pitch but still with rhythm; these are notated by cross note heads, as demonstrated in Section 1.4.1 [Special note heads], page 43.

9.6.5 Dialogue over music

Dialogue over music is usually printed over the staves in an italic font, with the start of each phrase keyed in to a particular music moment.

For short interjections a simple markup suffices.

```
\relative {
  a'4^\markup { \smallCaps { Alex - } \italic { He's gone } } a a a
  a4 a a^\markup { \smallCaps { Bethan - } \italic Where? } a
  a4 a a a
}
```



For longer phrases it may be necessary to expand the music to make the words fit neatly. There is no provision in LilyPond to do this fully automatically, and some manual intervention to layout the page will be necessary.

For long phrases or for passages with a lot of closely packed dialogue, using a Lyrics context will give better results. The Lyrics context should not be associated with a music Voice; instead each section of dialogue should be given an explicit duration. If there is a gap in the dialogue, the final word should be separated from the rest and the duration split between them so that the underlying music spaces out smoothly.

If the dialogue extends for more than one line it will be necessary to manually insert `\breaks` and adjust the placing of the dialogue to avoid running into the right margin. The final word of the last measure on a line should also be separated out, as above.

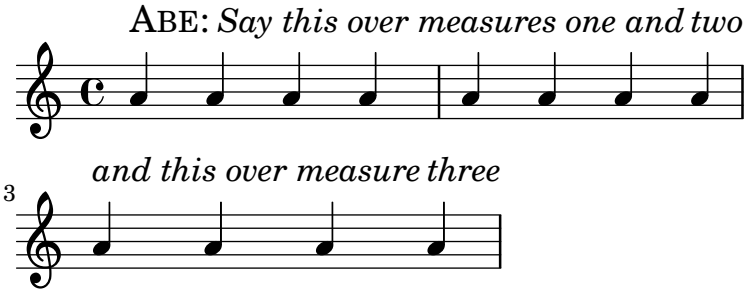
Here is an example illustrating how this might be done.

```
music = \relative {
  \*3 { a'4 a a a }
}

dialogue = \lyricmode {
  \markup {
    \fontsize #1 \upright \smallCaps Abe:
```

```
    "Say this over measures one and"
  }4*7
  "two"4 |
  \break
  "and this over measure"4*3
  "three"4 |
}

\score {
  <<
    \new Lyrics \with {
      \override LyricText.font-shape = #'italic
      \override LyricText.self-alignment-X = #LEFT
    }
    { \dialogue }
    \new Staff {
      \new Voice { \music }
    }
  >>
}
```



See also

Notation Reference: Section 9.1.5 [Manual syllable durations], page 352, Chapter 8 [Text], page 308.

Internal Reference: Section "LyricText" in *Internals Reference*.

9.7 Chants psalms and hymns

The music and words for chants, psalms and hymns usually follow a well-established format in any particular church. Although the formats may differ from church to church the typesetting problems which arise are broadly similar, and are covered in this section.

9.7.1 References for chants and psalms

Typesetting Gregorian chant in various styles of ancient notation is described in Chapter 17 [Ancient notation], page 538.

See also

Notation reference: Chapter 17 [Ancient notation], page 538.

Snippets: Section "Vocal music" in *Snippets*.

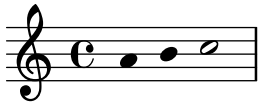
9.7.2 Setting a chant

Modern chant settings use modern notation with varying numbers of elements taken from ancient notation. Some of the elements and methods to consider are shown here.

Chants often use quarter notes without stems to indicate the pitch, with the rhythm being taken from the spoken rhythm of the words.
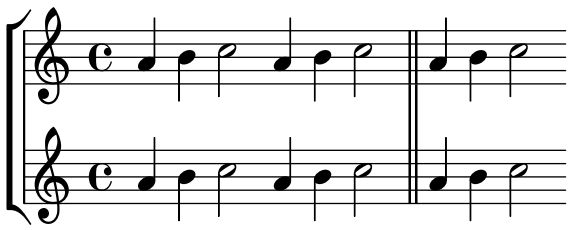
```
stemOff = { \hide Staff.Stem }

\relative c' {
  \stemOff
  a'4 b c2 |
}
```



Chants often omit measure bar lines or use shortened or dotted bar lines to indicate pauses in the music. To set a chant with no musical meter, see Section 2.3.5 [Unmetered music], page 93. To retain all the effects of a time signature but disable automatic measure bar lines, set `measureBarType` to `'()`. Without regular bar lines, you might need to take steps to control horizontal spacing; see Section 28.1 [Line breaking], page 689.
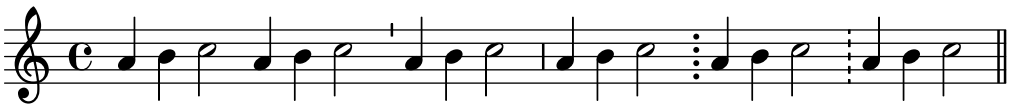
```
\score {
  \new StaffGroup <<
    \new Staff {
      \relative {
        a'4 b c2 |
        a4 b c2 | \section
        a4 b c2 |
      }
    }
    \new Staff {
      \relative {
        a'4 b c2 |
        a4 b c2 | \section
        a4 b c2 |
      }
    }
  >>
  \layout {
    \context {
      \Staff
      measureBarType = #'()
      forbidBreakBetweenBarLines = ##f
    }
  }
}
```



Measure bar lines can also be modified on a staff-by-staff basis; see Section 33.4 [Modifying context plug-ins], page 745.
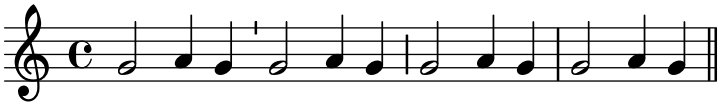
Rests or pauses in chants can be indicated by modified bar lines.

```
\relative a' {
  a4
  \cadenzaOn
  b c2
  a4 b c2
  \bar "'"
  a4 b c2
  \bar ","
  a4 b c2
  \bar ";"
  a4 b c2
  \bar "!"
  a4 b c2
  \bar "||"
}
```
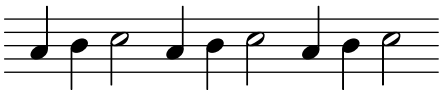


Alternatively, the notation used in Gregorian chant for pauses or rests is sometimes used even though the rest of the notation is modern.

```
\score {
  \relative {
    g'2 a4 g
    \divisioMinima
    g2 a4 g
    \divisioMaior
    g2 a4 g
    \divisioMaxima
    g2 a4 g
    \finalis
  }
  \layout {
    \context {
      \Staff
      \remove Caesura_engraver
      \consists Divisio_engraver
      \EnableGregorianDivisiones
      caesuraType = #'((breath . chantquarterbar))
      measureBarType = #'()
      forbidBreakBetweenBarLines = ##f
    }
  }
}
```



Chants usually omit the time signature and often omit the clef too.

```
\score {
  \new Staff {
    \relative {
      a'4 b c2 |
      a4 b c2 |
      a4 b c2 |
    }
  }
  \layout {
    \context {
      \Staff
      \remove Time_signature_engraver
      \remove Clef_engraver
      measureBarType = #'()
      forbidBreakBetweenBarLines = ##f
    }
  }
}
```



Chants for psalms in the Anglican tradition are usually either *single*, with 7 bars of music, or *double*, with two lots of 7 bars. Each group of 7 bars is divided into two halves, corresponding to the two halves of each verse, usually separated by a double bar line. Only whole and half notes are used. The 1st bar in each half always contains a single chord of whole notes. This is the "reciting note". Chants are usually centered on the page.

```
SopranoMusic = \relative {
  g'1 | c2 b | a1 |
  a1 | d2 c | c b | c1 |
}

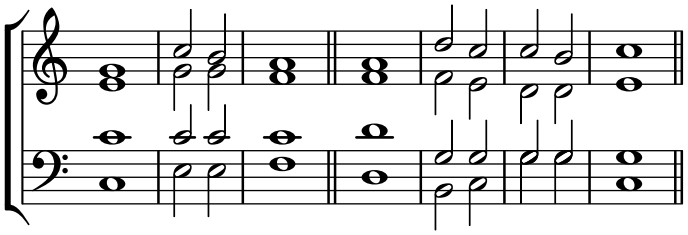
AltoMusic = \relative {
  e'1 | g2 g | f1 |
  f1 | f2 e | d d | e1 |
}

TenorMusic = \relative {
  c'1 | c2 c | c1 |
  d1 | g,2 g | g g | g1 |
}

BassMusic =  \relative {
  c1 | e2 e | f1 |
  d1 | b2 c | g' g | c,1 |
}

global = {
  \time 2/2
  \skip 1*3 \section
  \skip 1*4 \fine
}
```

```
% Use markup to center the chant on the page
\markup {
  \fill-line {
    \score {  % centered
      <<
        \new ChoirStaff <<
          \new Staff <<
            \global
            \clef "treble"
            \new Voice = "Soprano" <<
              \voiceOne
              \SopranoMusic
            >>
            \new Voice = "Alto" <<
              \voiceTwo
              \AltoMusic
            >>
          >>
          \new Staff <<
            \clef "bass"
            \global
            \new Voice = "Tenor" <<
              \voiceOne
              \TenorMusic
            >>
            \new Voice = "Bass" <<
              \voiceTwo
              \BassMusic
            >>
          >>
        >>
      >>
      \layout {
        \context {
          \Score
          \override SpacingSpanner.base-shortest-duration =
            \musicLength 2
          fineBarType = "||"
        }
        \context {
          \Staff
          \remove Time_signature_engraver
        }
      }
    }  % End score
  }
}  % End markup
```



Some other approaches to setting such a chant are shown in the first of the following snippets.
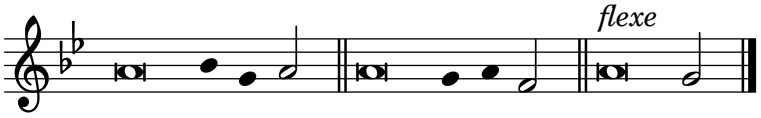
Selected snippets

Chant or psalm notation

This form of notation is used for psalm chant, where verses are not always of the same length.

```
stemOff = \hide Staff.Stem
stemOn  = \undo \stemOff

\score {
  \new Staff \with { \remove "Time_signature_engraver" }
  {
    \key g \minor
    \cadenzaOn
    \stemOff a'\breve bes'4 g'4
    \stemOn a'2 \section
    \stemOff a'\breve g'4 a'4
    \stemOn f'2 \section
    \stemOff a'\breve^\markup { \italic flexe }
    \stemOn g'2 \fine
  }
}
```



Canticles and other liturgical texts may be set more freely, and may use notational elements from ancient music. Often the words are shown underneath and aligned with the notes. If so, the notes are spaced in accordance with the syllables rather than the notes' durations.

Ancient notation template – modern transcription of Gregorian music
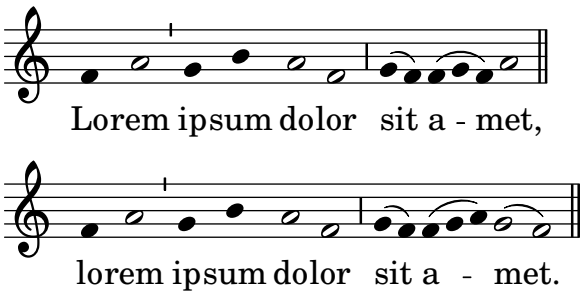
This example demonstrates how to do modern transcription of Gregorian music. Gregorian music has no measure, no stems; it uses only half and quarter note heads, and special marks, indicating rests of different length.

```
chant = \relative c' {
  \set Score.timing = ##f
  f4 a2 \divisioMinima
  g4 b a2 f2 \divisioMaior
  g4( f) f( g f) a2 \finalis \break
  f4 a2 \divisioMinima
  g4 b a2 f2 \divisioMaior
  g4( f) f( g a) g2( f) \finalis
```

```
}

verba = \lyricmode {
  Lo -- rem ip -- sum do -- lor sit a -- met,
  lo -- rem ip -- sum do -- lor sit a -- met.
}

\score {
  \new GregorianTranscriptionStaff <<
    \new GregorianTranscriptionVoice = "melody" \chant
    \new GregorianTranscriptionLyrics = "one" \lyricsto melody \verba
  >>
}
```



See also

Learning Manual: Section “Visibility and color of objects” in *Learning Manual*, Section “Vocal ensembles templates” in *Learning Manual*.

Notation Reference: Chapter 17 [Ancient notation], page 538, Section 2.5.1 [Bar lines], page 121, Section 33.4 [Modifying context plug-ins], page 745, Section 17.4 [Typesetting Gregorian chant], page 554, Section 2.3.5 [Unmetered music], page 93, Section 36.7 [Visibility of objects], page 784.

9.7.3 Pointing a psalm

The words to an Anglican psalm are usually printed in separate verses centered underneath the chant.

Single chants (with 7 bars) are repeated for every verse. Double chants (with 14 bars) are repeated for every pair of verses. Marks are inserted in the words to show how they should be fitted to the chant. Each verse is divided into two halves. A colon is usually used to indicate this division. This corresponds to the double bar line in the music. The words before the colon are sung to the first three bars of music; the words after the colon are sung to the last four bars.

Single bar lines (or in some psalters an inverted comma or similar symbol) are inserted between words to indicate where the bar lines in the music fall. In markup mode a single bar line can be entered with the bar check symbol, `|`.

```
\markup {
  \fill-line {
    \column {
      \line { O come let us sing | unto the | Lord : let }
      \line { us heartily rejoice in the | strength of | our }
      \line { sal- | -vation. }
    }
  }
}
```

O come let us sing | unto the | Lord : let
us heartily rejoice in the | strength of | our
sal- | -vation.

Other symbols may require glyphs from the `fetaMusic` fonts. For details, see Section 8.3 [Fonts], page 337.

```
tick = \markup {
  \raise #1 \fontsize #-5 \musicglyph "scripts.rvarcomma"
}
\markup {
  \fill-line {
    \column {
      \line { O come let us sing \tick unto the \tick Lord : let }
      \line { us heartily rejoice \bold \italic in the \tick strength of \tick our }
      \line { sal \tick vation. }
    }
  }
}
```

O come let us sing ' unto the ' Lord : let
us heartily rejoice in the ' strength of ' our
sal ' vation.

Where there is one whole note in a bar all the words corresponding to that bar are recited on that one note in speech rhythm. Where there are two notes in a bar there will usually be only one or two corresponding syllables. If there are more that two syllables a dot is usually inserted to indicate where the change in note occurs.

```
dot = \markup {
  \raise #0.7 \musicglyph "dots.dot"
}
tick = \markup {
  \raise #1 \fontsize #-5 \musicglyph "scripts.rvarcomma"
}
\markup {
  \fill-line {
    \column {
      \line { O come let us sing \tick unto \dot the \tick Lord : let }
      \line { us heartily rejoice \bold \italic in the \tick strength of \tick our }
      \line { sal \tick vation. }
    }
  }
}
```

O come let us sing ' unto • the ' Lord : let
us heartily rejoice in the ' strength of ' our
sal ' vation.

In some psalters an asterisk is used to indicate a break in a recited section instead of a comma, and stressed or slightly lengthened syllables are indicated in bold text.

```
dot = \markup {
  \raise #0.7 \musicglyph "dots.dot"
}
tick = \markup {
  \raise #1 \fontsize #-5 \musicglyph "scripts.rvarcomma"
}
\markup {
  \fill-line {
    \column {
      \line { Today if ye will hear his voice * }
      \line { \concat { \bold hard en } |
              not your | hearts : as in the pro- }
      \line { vocation * and as in the \bold day of tempt- | }
      \line { -ation | in the | wilderness. }
    }
  }
}
```

Today if ye will hear his voice *
harden | not your | hearts : as in the pro-
vocation * and as in the **day** of tempt- |
-ation | in the | wilderness.

In other psalters an accent is placed over the syllable to indicate stress.

```
tick = \markup {
  \raise #2 \fontsize #-5 \musicglyph "scripts.rvarcomma"
}
\markup {
  \fill-line {
    \column {
      \line { O come let us \concat { si \combine \tick ng } |
              unto the | Lord : let }
      \line { us heartily \concat { rejo \combine \tick ice }
              in the | strength of | our }
      \line { sal- | -vation. }
    }
  }
}
```

O come let us sing′ | unto the | Lord : let
us heartily rejoice′ in the | strength of | our
sal- | -vation.

The use of markup to center text, and arrange lines in columns is described in Section 8.2 [Formatting text], page 320.

Most of these elements are shown in one or other of the two verses in the template, see Section "Psalms" in *Learning Manual*.

See also

Learning Manual: Section "Psalms" in *Learning Manual*, Section "Vocal ensembles templates" in *Learning Manual*.

Notation Reference: Section 8.3 [Fonts], page 337, Section 8.2 [Formatting text], page 320.

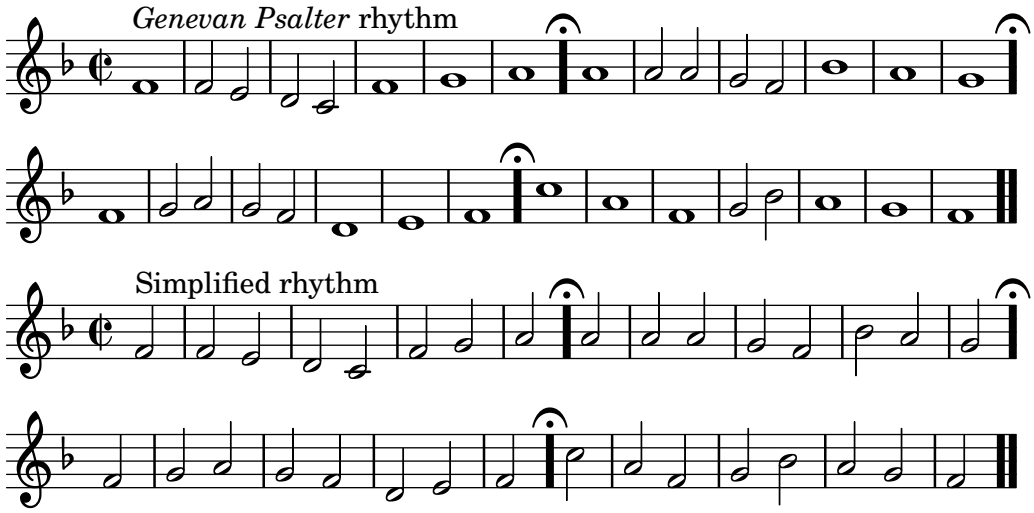
9.7.4 Phrase bar lines in hymn tunes

The `\caesura` command can be configured to create phrase bar lines that interact well with other automatic bar lines (see Section 2.5.2 [Automatic bar lines], page 131).

In the following examples, the source code for the tune in `old-hundredth-example.ly` uses `\caesura` between poetic lines and `\fine` at the end. There are no `\fermata` or `\bar` commands; those symbols appear in the output because of the caesura configuration.

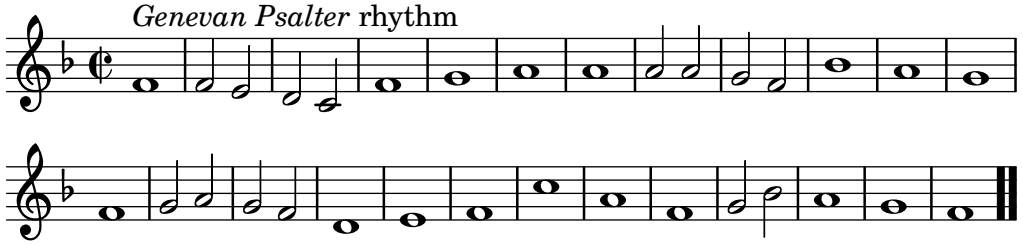
The Boston Handel and Haydn Society Collection of Church Music (1830) has a thick bar after each phrase. For this tune, it also has fermatas over the bar lines:
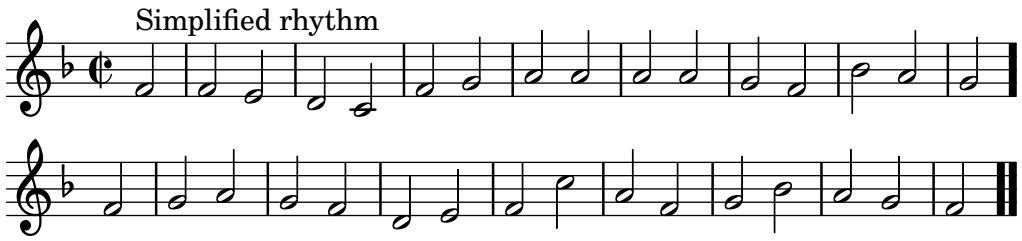
```
\layout {
  \context {
    \Score
    caesuraType = #'((bar-line . ".")
                     (scripts . (fermata)))
    fineBarType = ".."
  }
}
\include "old-hundredth-example.ly"
```



J.S. James' *Original Sacred Harp* (1911) has a thick bar when a line is broken in mid-measure at the end of a phrase.

```
\layout {
  \context {
    \Score
    caesuraType = #'((underlying-bar-line . "x-."))
    fineBarType = ".."
  }
}
\include "old-hundredth-example.ly"
```





9.7.5 Partial measures in hymn tunes

Hymn tunes frequently start and end every line of music with partial measures so that each line of music corresponds exactly with a line of text. This requires a `\partial` command at the start of the music and a bar line at the end of each line.

Hymn template

This code shows one way of setting out a hymn tune where each line starts and ends with a partial measure. It also shows how to add the verses as stand-alone text under the music.

```
Timeline = {
  \time 4/4
  \tempo 4=96
  \partial 2
  s2 | s1 | s2 \breathe s2 | s1 | s2 \caesura \break
  s2 | s1 | s2 \breathe s2 | s1 | s2 \fine
}

SopranoMusic = \relative g' {
  g4 g | g g g g | g g g g | g g g g | g2
  g4 g | g g g g | g g g g | g g g g | g2
}

AltoMusic = \relative c' {
  d4 d | d d d d | d d d d | d d d d | d2
  d4 d | d d d d | d d d d | d d d d | d2
}

TenorMusic = \relative a {
  b4 b | b b b b | b b b b | b b b b | b2
  b4 b | b b b b | b b b b | b b b b | b2
}

BassMusic = \relative g {
  g4 g | g g g g | g g g g | g g g g | g2
  g4 g | g g g g | g g g g | g g g g | g2
}

global = {
 \key g \major
}

\score {  % Start score
  \new PianoStaff <<  % Start pianostaff
    \new Staff <<  % Start Staff = RH
      \global
```

```
      \clef "treble"
      \new Voice = "Soprano" <<  % Start Voice = "Soprano"
        \Timeline
        \voiceOne
        \SopranoMusic
      >>  % End Voice = "Soprano"
      \new Voice = "Alto" <<  % Start Voice = "Alto"
        \Timeline
        \voiceTwo
        \AltoMusic
      >>  % End Voice = "Alto"
    >>  % End Staff = RH

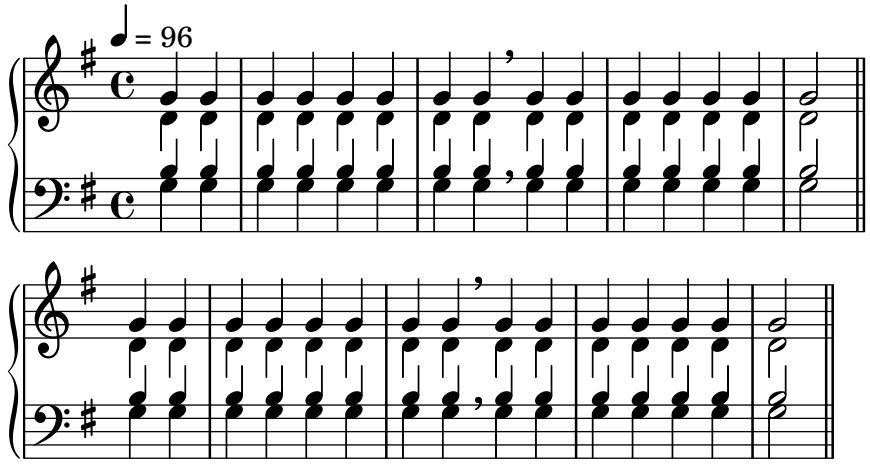
    \new Staff <<  % Start Staff = LH
      \global
      \clef "bass"
      \new Voice = "Tenor" <<  % Start Voice = "Tenor"
        \Timeline
        \voiceOne
        \TenorMusic
      >>  % End Voice = "Tenor"
      \new Voice = "Bass" <<  % Start Voice = "Bass"
        \Timeline
        \voiceTwo
        \BassMusic
      >>  % End Voice = "Bass"
    >>  % End Staff = LH
  >>  % End pianostaff
}  % End score

\markup \fill-line {
  \left-column {
    "This is line one of the first verse"
    "This is line two of the same"
    \null
    "And here's line one of the second verse"
    "And the next line of the same"
  }
}

\layout {
  \context {
    \Score
    caesuraType = #'((bar-line . "||"))
    fineBarType = "||"
  }
}

\paper {  % Start paper block
  indent = 0         % don't indent first system
  line-width = 130   % shorten line length to suit music
  tagline = ##f      % Don't print tag line, can be removed
```

```
}  % End paper block
```



This is line one of the first verse
This is line two of the same

And here's line one of the second verse
And the next line of the same
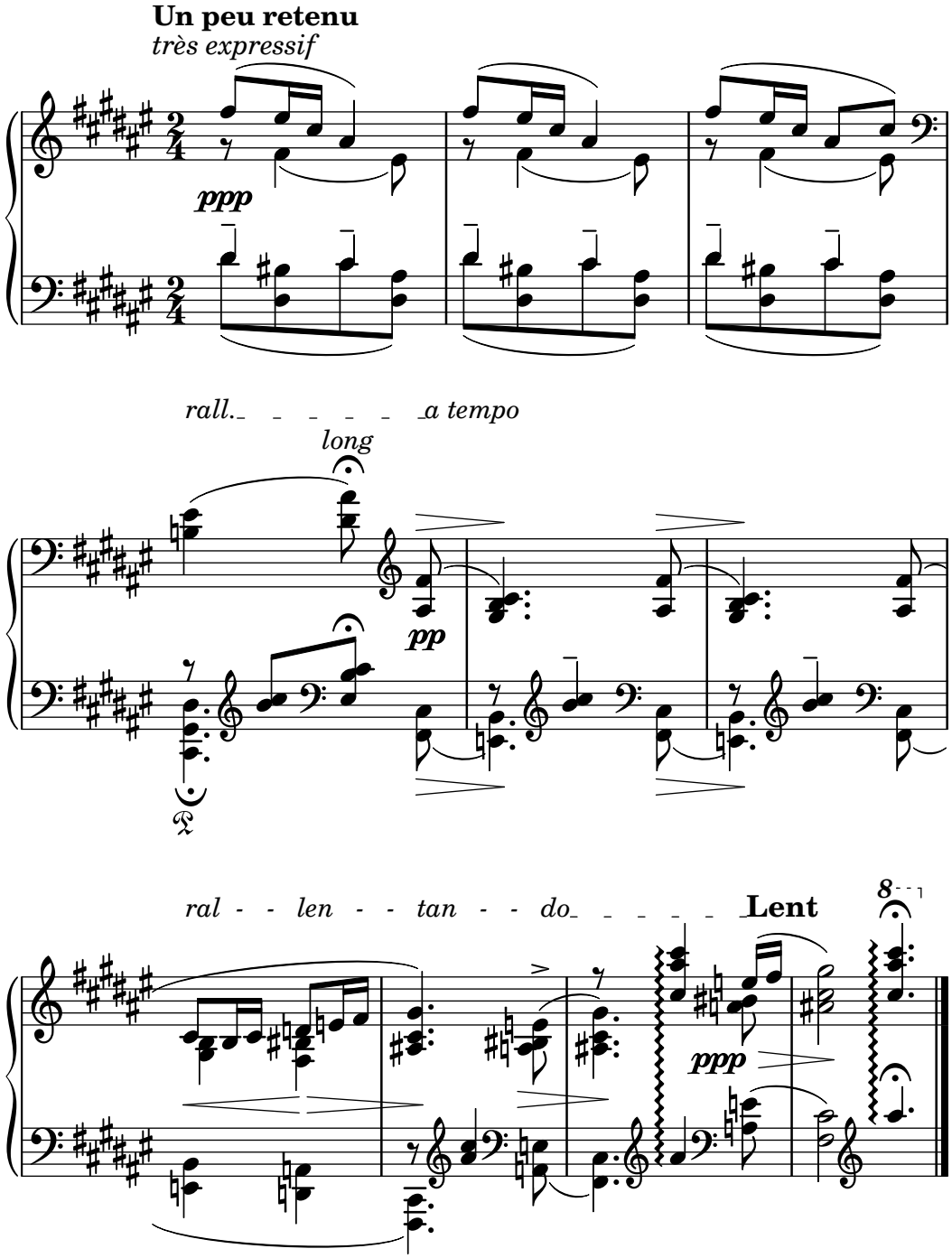
9.8 Ancient vocal music

Ancient vocal music is supported, as explained in Chapter 17 [Ancient notation], page 538.

See also

Notation Reference: Chapter 17 [Ancient notation], page 538.

10 Keyboard and other multi-staff instruments



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.

10.1 Common notation for keyboards

This section discusses notation issues that may arise for most keyboard instruments.

10.1.1 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 Section 5.2.3 [Collision resolution], page 227.
- Keyboard music can be written in parallel, as described in Section 5.2.6 [Writing music in parallel], page 238.
- Dynamics may be placed in a `Dynamics` context, between the two `Staff` contexts to align the dynamic marks on a horizontal line centered between the staves; see Section 3.1.2 [Dynamics], page 160.
- Fingerings are indicated with Section 7.1.2 [Fingering instructions], page 282.
- Organ pedal indications are inserted as articulations, see Section 10.3.1 [Organ pedal marks], page 423, and Section B.13 [List of articulations], page 928.
- Vertical grid lines can be shown with Section 7.2.3 [Grid lines], page 302.
- Keyboard music often contains *Laissez vibrer* ties as well as ties on arpeggios and tremolos, described in Section 2.1.4 [Ties], page 63.
- Placing arpeggios across multiple voices and staves is covered in Section 3.3.2 [Arpeggio], page 182.
- Tremolo marks are described in Section 4.2.2 [Tremolo repeats], page 212.
- 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: Section 6.1.2 [Grouping staves], page 244, Section 6.3.1 [Instrument names], page 262, Section 5.2.3 [Collision resolution], page 227, Section 5.2.6 [Writing music in parallel], page 238, Section 7.1.2 [Fingering instructions], page 282, Section B.13 [List of articulations], page 928, Section 7.2.3 [Grid lines], page 302, Section 2.1.4 [Ties], page 63, Section 3.3.2 [Arpeggio], page 182, Section 4.2.2 [Tremolo repeats], page 212.

Internals Reference: Section “PianoStaff” in *Internals Reference*.

Snippets: Section “Keyboard and other multi-staff instruments” in *Snippets*.

10.1.2 Changing staff manually

Voices can be switched between staves manually, using the command

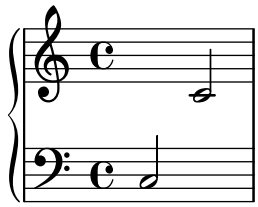
```
\change Staff = staffname
```

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 33.3 [Keeping contexts alive], page 742, or

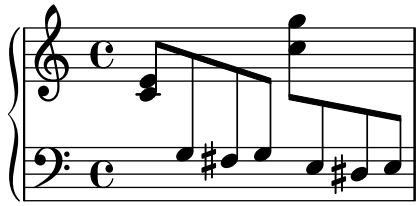
explicitly instantiated, for example by using the empty chord, <>, see Section 5.1.1 [Chorded notes], page 216.

```
\new PianoStaff <<
  \new Staff = "up" {
    % enforce creation of all contexts at this point of time
    <>
    \change Staff = "down" c2
    \change Staff = "up" c'2
  }
  \new Staff = "down" {
    \clef bass
    % keep staff alive
    s1
  }
>>
```



Cross-staff notes are beamed automatically:

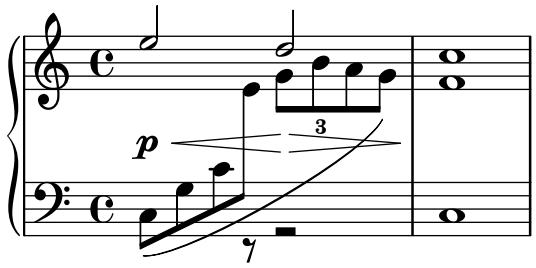
```
\new PianoStaff <<
  \new Staff = "up" {
    <e' c'>8
    \change Staff = "down"
    g8 fis g
    \change Staff = "up"
    <g'' c''>8
    \change Staff = "down"
    e8 dis e
    \change Staff = "up"
  }
  \new Staff = "down" {
    \clef bass
    % keep staff alive
    s1
  }
>>
```



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*.

Overlapping notation can result when voices cross staves:

```
\new PianoStaff <<
  \new Staff = "up" {
    \voiceOne
    % Make space for fingering in the cross-staff voice
    \once\override DynamicLineSpanner.staff-padding = 4
    e''2\p\< d''\>
    c''1\!
  }
  \new Staff = "down" <<
    {
      \clef bass
      s4. e,8\rest g,2\rest
      c1
    } \\ {
      c8\( g c'
      \change Staff = "up"
      e' g' b'-3 a' g'\)
      f'1
    }
  >>
>>
```



The stem and slur overlap the intervening line of dynamics because automatic collision resolution is suspended for beams, slurs and other spanners that connect notes on different staves, as well as for stems and articulations if their placement is affected by a cross-staff spanner. The resulting collisions must be resolved manually, where necessary, using the methods in Section "Fixing overlapping notation" in *Learning Manual*.

See also

Learning Manual: Section "Fixing overlapping notation" in *Learning Manual*.

Notation Reference: Section 7.1.9 [Stems], page 297, Section 2.4.1 [Automatic beams], page 102, Section 33.3 [Keeping contexts alive], page 742.

Snippets: Section "Keyboard and other multi-staff instruments" in *Snippets*.

Internals Reference: Section "Beam" in *Internals Reference*, Section "ContextChange" in *Internals Reference*.

Known issues and warnings

Beam collision avoidance does not work for automatic beams that end right before a change in staff. In this case use manual beams.
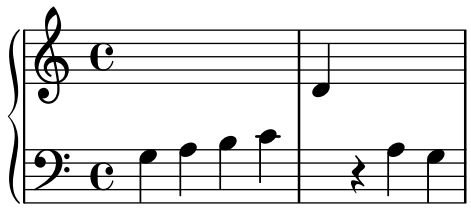
10.1.3 Changing staff automatically

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 auto-changer switches on the basis of the pitch (middle C is the turning point), and it looks ahead skipping over rests to switch in advance.
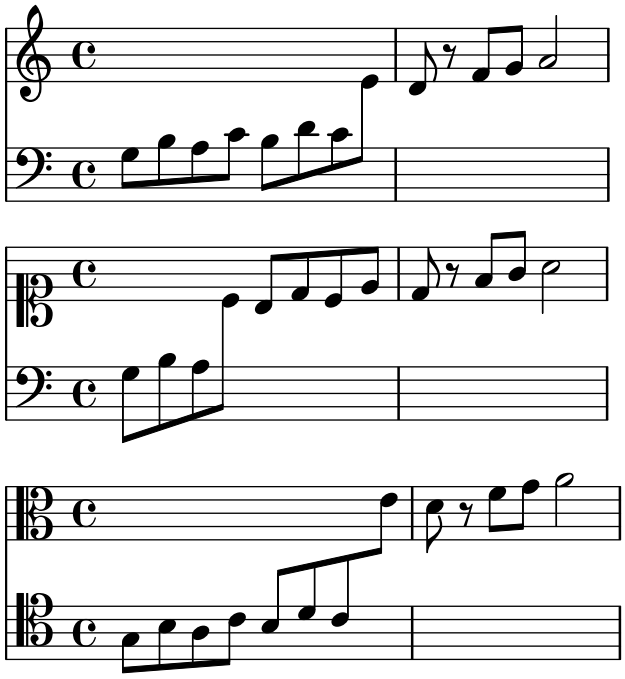
```
\new PianoStaff {
  \autoChange {
    g4 a b c'
    d'4 r a g
  }
}
```



It is possible to specify other pitches for the turning point. If the staves are not instantiated explicitly, other clefs may be used.

```
music = {
  g8 b a c' b8 d' c'8 e'
  d'8 r f' g' a'2
}

\autoChange d' \music
\autoChange b \with { \clef soprano } \music
\autoChange d' \with { \clef alto } \with { \clef tenor } \music
```



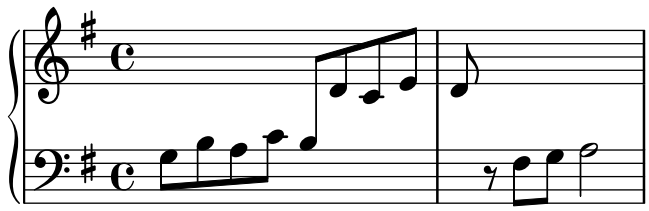
A `\relative` section that is outside of `\autoChange` has no effect on the pitches of the music, so if necessary, put `\relative` inside `\autoChange`.

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 {
        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: Section 10.1.2 [Changing staff manually], page 414.

Snippets: Section "Keyboard and other multi-staff instruments" in *Snippets*.

Known issues and warnings

The staff switches may not end up in optimal places. For high quality output, staff switches should be specified manually.

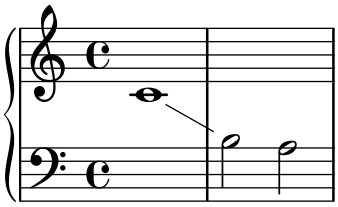
Chords will not be split across the staves; they will be assigned to a staff based on the first note named in the chord construct.

10.1.4 Staff-change lines

Whenever a voice switches to another staff, a line connecting the notes can be printed automatically:

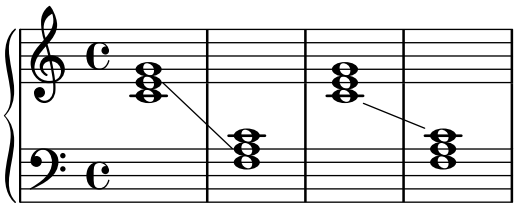
```
\new PianoStaff <<
  \new Staff = "one" {
    \showStaffSwitch
    c'1
    \change Staff = "two"
    b2 a
```

```
  }
  \new Staff = "two" {
    \clef bass
    s1*2
  }
>>
```



A staff-change line between chords connects the chords' "last notes" as written in the source file; this can be used to quickly adjust the line's vertical start and end positions.

```
\new PianoStaff <<
  \new Staff = "one" {
    <c' e' g'>1
    \showStaffSwitch
    \change Staff = "two"
    <a c' f>1
    \hideStaffSwitch
    \change Staff = "one"
    <e' g' c'>1
    \showStaffSwitch
    \change Staff = "two"
    <f a c'>1
  }
  \new Staff = "two" {
    \clef bass
    s1*4
  }
>>
```



Predefined commands

`\showStaffSwitch`, `\hideStaffSwitch`.

See also

Snippets: Section "Keyboard and other multi-staff instruments" in *Snippets*.

Internals Reference: Section "Note_head_line_engraver" in *Internals Reference*, Section "VoiceFollower" in *Internals Reference*.

Selected snippets

Cross-staff stems

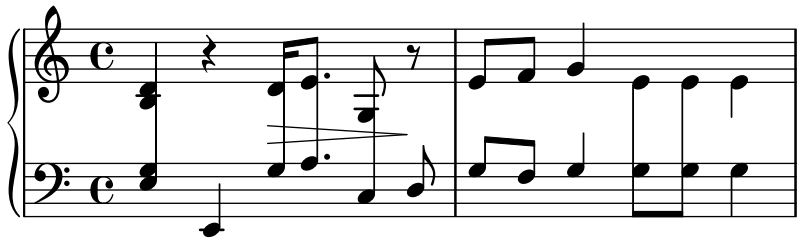
This snippet shows how to use `Span_stem_engraver` and `\crossStaff` to connect stems across staves automatically.

The stem lengths need not be specified, as the variable distance between noteheads and staves is calculated automatically. However, it is important that `\crossStaff` is applied to the correct voice or staff (i.e., on the opposite side of where a beam is or would be positioned) to get the desired effect.

```
\layout {
  \context {
    \PianoStaff
    \consists "Span_stem_engraver"
  }
}

\new PianoStaff <<
  \new Staff {
    <b d'>4 r d'16\> e'8. g8 r\! |
    e'8 f' g'4
      \voiceTwo
      % Down to lower staff
      \crossStaff { e'8 e'8 } e'4 |
  }

  \new Staff {
    \clef bass
    \voiceOne
    % Up to upper staff
    \crossStaff { <e g>4 e, g16 a8. c8 } d |
    g8 f g4 \voiceTwo g8 g g4 |
  }
>>
```



Indicating cross-staff chords with a bracket

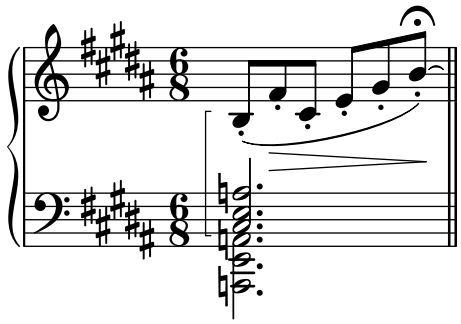
A non-arpeggiato bracket can indicate that notes on two different staves are to be played with the same hand. In order to do this, the `PianoStaff` must be set to accept cross-staff brackets.

The following example typesets measure 65 of Debussy's prelude *Les collines d'Anacapri*.

```
\new PianoStaff <<
  \once \set PianoStaff.connectChordBrackets = ##t

  \new Staff \relative c' {
    \key b \major
    \time 6/8
    b8-.(\nonArpeggiato fis'-.\> cis-.
      e-. gis-. b-.)\!\fermata^\laissezVibrer
    \section
  }

  \new Staff \relative c' {
    \clef bass
    \key b \major
    << { <a e cis>2.\nonArpeggiato } \\
       { <a, e a,>2. } >>
    \section
  }
>>
```



See also

Snippets: Section “Keyboard and other multi-staff instruments” in *Snippets*.

Internals Reference: Section “Stem” in *Internals Reference*.
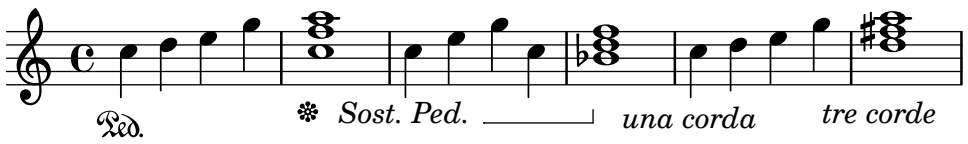
10.2 Piano

This section discusses notation issues that relate most directly to the piano.

10.2.1 Piano pedals

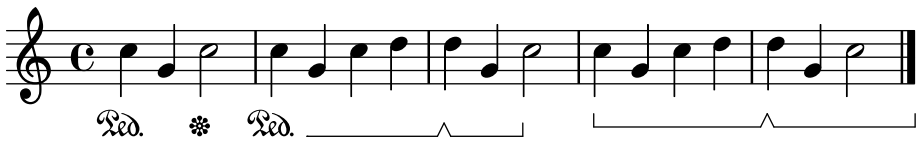
Pianos generally have three pedals that alter the way sound is produced: *sustain*, *sostenuto* (*sos.*), and *una corda* (*U.C.*). Sustain pedals are also found on vibraphones and celestas.

```
\relative {
  c''4\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.

```
\relative {
  c''4\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. Pedaling to the final bar line is indicated by omitting the final pedal off command.

Pedal indications may be placed in a `Dynamics` context, which aligns them on a horizontal line.

See also

Notation Reference: Section 2.1.4 [Ties], page 63.

Snippets: Section “Keyboard and other multi-staff instruments” 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*.

10.3 Organ

This section discusses notation issues that relate to the (church) organ.

10.3.1 Organ pedal marks

Mostly in organ method books you can find marks that indicate how to use the toe and heel of the left and right foot while playing the pedal. Depending on the style of the book, different glyphs are used for such marks. In almost all cases, however, the pedal marks above the staff are for the right foot, and the marks below the staff are for the left foot.

The main commands to print such marks are `\rtoe`, `\ltoe`, `\rheel`, and `\lheel`. Use `toeHeelStyle` context property to select the pedal mark style. Available options are as follows.

`default` The default at start-up, mainly for backward compatibility.

`standard` The most common scheme in use today; the shapes are identical above and below the staff.

`reversed` The glyph 'tips' point to the staff.

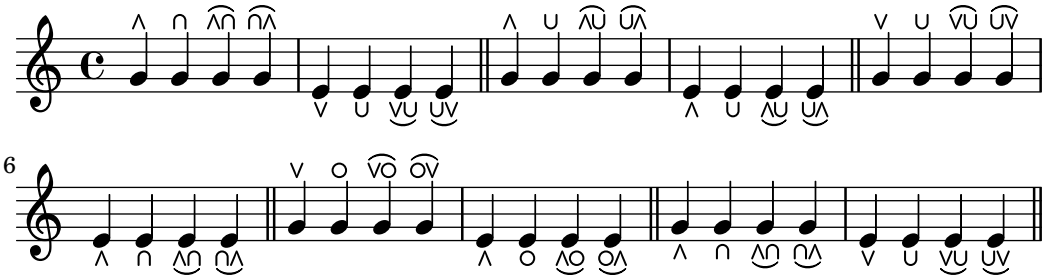
`circleheels` Use a circle glyph for the heel.

`below` This is for the rare cases where the pedal is notated in the same staff as the left hand, which makes it necessary that marks for both the left and right foot are below the staff.

For toe-heel and heel-toe substitutions the commands `\rtoeheel`, `\ltoeheel`, `\rheeltoe`, and `\lheeltoe` are provided.

```
music = { g'4_\rtoe g'\rheel g'\rtoeheel g'\rheeltoe |
          e'4\ltoe e'\lheel e'\ltoeheel e'\lheeltoe \bar "||" }

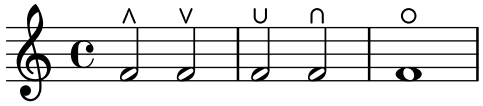
{
  \music              % #'default
  \set toeHeelStyle = #'standard \music
  \set toeHeelStyle = #'reversed \music
  \set toeHeelStyle = #'circleheels \music
  \set toeHeelStyle = #'below \music
}
```



`\rtoe` and siblings look similar to other articulation commands like `\flageolet` or `\prall`, and they are indeed `Script` grobs. However, they ignore direction changes (i.e., '^', '_', and `\tweak direction` are ignored). This is due to the `Toe_heel_engraver`, which is enabled by default. The idea behind this decision is separation of content and representation, letting a style control the position of the marks.

If the need arises to position pedal marks individually above or below the staff, use the standard articulation commands `\toe`, `\vartoe`, `\heel`, `\varheel`, and `\heelcircle`.

```
{
  f'2\toe f'2\vartoe |
  f'2\heel f'2\varheel |
  f'1\heelcircle
}
```



10.4 Accordion

This section discusses notation that is unique to the accordion.

10.4.1 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.

A complete list of all available accordion registers can be found in Section A.1.7 [Accordion registers], page 869.

Selected snippets

Accordion register symbols

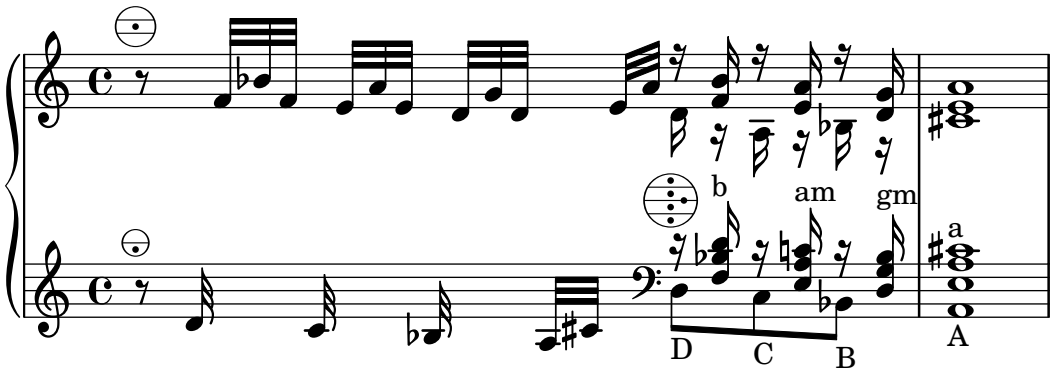
Accordion register symbols are available as `\markup` as well as as standalone music events (as register changes tend to occur between actual music events). Bass registers are not overly standardized. The available commands can be found in 'Discant symbols' in the Notation Reference (`https://lilypond.org/doc/v2.24/Documentation/notation/accordion#discant-symbols`).

```
#(use-modules (lily accreg))

\new PianoStaff <<
  \new Staff \relative {
    \clef treble
    \discant "10"
    r8 s32 f'[ bes f] s e[ a e] s d[ g d] s16 e32[ a]
    <<
      { r16 <f bes> r <e a> r <d g> }
      \\
      { d r a r bes r }
    >> |
    <cis e a>1
  }

  \new Staff \relative {
    \clef treble
    \freeBass "1"
    r8 d'32 s16. c32 s16. bes32 s16. a32[ cis] s16
    \clef bass \stdBass "Master"
    <<
      { r16 <f, bes d>^"b" r <e a c>^"am" r <d g bes>^"gm" |
```

```
      <e a cis>1^"a" }
      \\
      { d8_"D" c_"C" bes_"B" | a1_"A" }
    >>
  }
>>
```



See also

Snippets: Section "Keyboard and other multi-staff instruments" in *Snippets*.

10.5 Harp

This section discusses notation issues that are unique to the harp.

10.5.1 References for harps

Some common characteristics of harp music are covered elsewhere:

- The glissando is the most characteristic harp technique, Section 3.3.1 [Glissando], page 177.
- A *bisbigliando* is written as a tremolo Section 4.2.2 [Tremolo repeats], page 212.
- Natural harmonics are covered under Section 11.1.3 [Harmonics], page 428.
- For directional arpeggios and non-arpeggios, see Section 3.3.2 [Arpeggio], page 182.
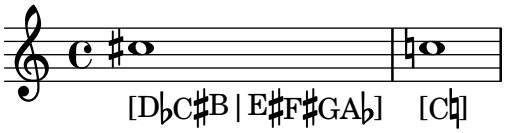
See also

Notation Reference: Section 4.2.2 [Tremolo repeats], page 212, Section 3.3.1 [Glissando], page 177, Section 3.3.2 [Arpeggio], page 182, Section 11.1.3 [Harmonics], page 428.

10.5.2 Harp pedals

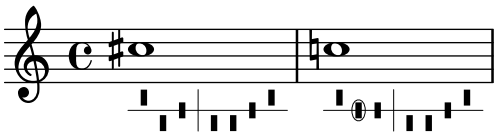
Harps have seven strings per octave that may be sounded at the natural, flattened, or sharpened pitch. In lever harps, each string is adjusted individually, but in pedal harps every string with the same pitch name is controlled by a single pedal. From the player's left to right, the pedals are D, C, and B on the left and E, F, G, and A on the right. The position of the pedals may be indicated with text marks:

```
\textLengthOn
cis''1_\markup \concat \vcenter {
  [D \flat C \sharp B|E \sharp F \sharp G A \flat] }
c''!1_\markup \concat \vcenter {
  [ C \natural ] }
```



or pedal diagrams:

```
\textLengthOn
cis''1_\markup { \harp-pedal "^v-|vv-^" }
c''!1_\markup { \harp-pedal "^o--|vv-^" }
```

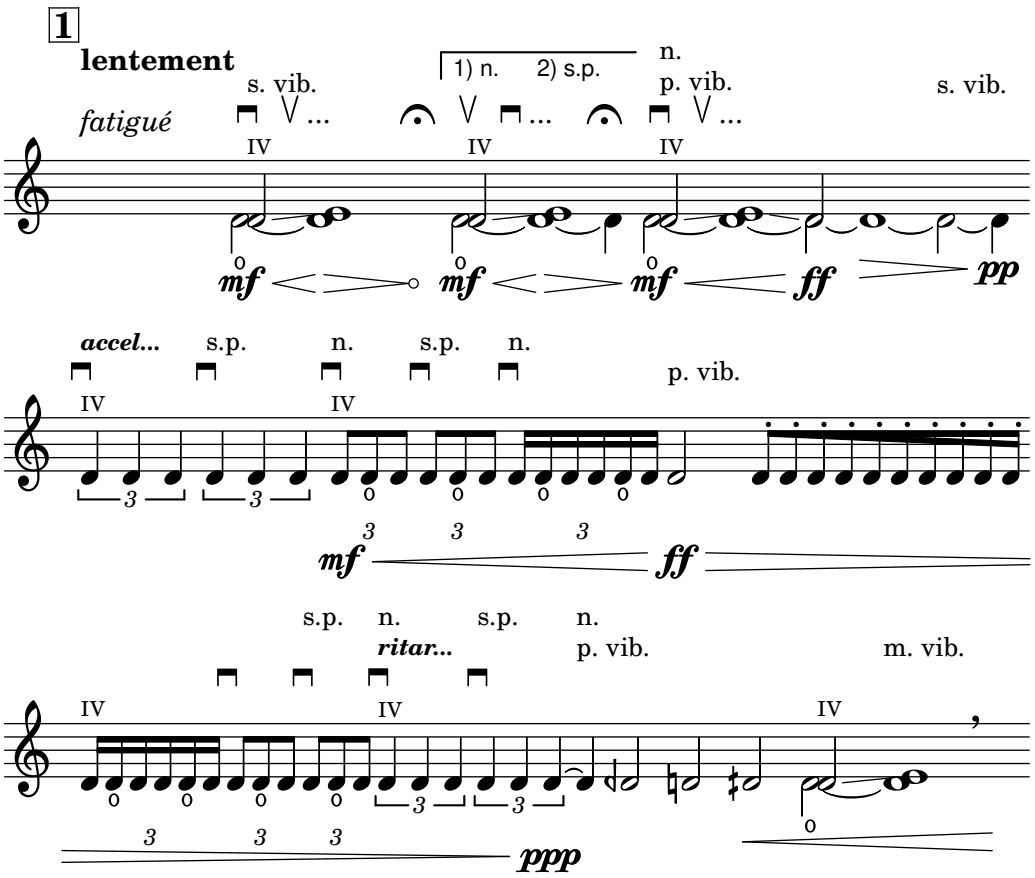


The `\harp-pedal` command accepts a string of characters, where `^` is the highest pedal position (flattened pitch), `-` is the middle pedal position (natural pitch), `v` is the lowest pedal position (sharpened pitch), and `|` is the divider. A prefixed `o` will circle the following pedal symbol.

See also

Notation Reference: Section 8.1.2 [Text scripts], page 310, Section A.1.6 [Instrument-specific markup], page 864.

11 Unfretted string instruments



This section provides information and references which are helpful when writing for unfretted string instruments, principally orchestral strings.

11.1 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.

11.1.1 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 Section 8.1.2 [Text scripts], page 310.
- Fingerings, including the thumb indication, are described in Section 7.1.2 [Fingering instructions], page 282.
- String numbers may be added (generally in roman numbers for bowed instruments) as explained in Section 12.1.2 [String number indications], page 432.
- Double stopping is normally indicated by writing a chord, see Section 5.1.1 [Chorded notes], page 216. Directives for playing chords may be added, see Section 3.3.2 [Arpeggio], page 182.
- Templates for string quartets can be found in Section “String quartet templates” in *Learning Manual*. Others are shown in the snippets.

See also

Learning Manual: Section “String quartet templates” in *Learning Manual*.

Notation Reference: Section 8.1.2 [Text scripts], page 310, Section 7.1.2 [Fingering instructions], page 282, Section 5.1.1 [Chorded notes], page 216, Section 3.3.2 [Arpeggio], page 182.
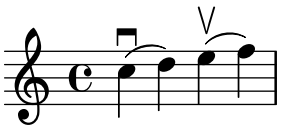
Snippets: Section “Unfretted string instruments” in *Snippets*.

11.1.2 Bowing indications

Bowing indications are created as articulations, which are described in Section 3.1.1 [Articulations and ornamentations], page 156.

The bowing commands, `\upbow` and `\downbow`, are used with slurs as follows:
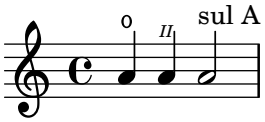
```
\relative { c''4(\downbow d) e(\upbow f) }
```



Roman numerals can be used for string numbers (rather than the default circled Arabic numbers), as explained in Section 12.1.2 [String number indications], page 432.

Alternatively, string indications may be printed using markup commands; articulation scripts may also indicate open strings.

```
a'4 \open
\romanStringNumbers
a'\2
a'2^\markup { \small "sul A" }
```



Predefined commands

`\downbow`, `\upbow`, `\open`, `\romanStringNumbers`.

See also
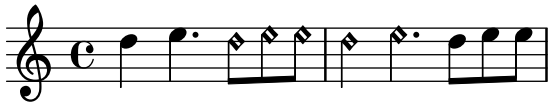
Notation Reference: Section 3.1.1 [Articulations and ornamentations], page 156, Section 12.1.2 [String number indications], page 432, Section 3.2.1 [Slurs], page 171.

11.1.3 Harmonics

Natural harmonics
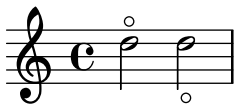
Natural harmonics can be notated in several ways. A diamond-shaped note head generally means to touch the string where you would stop the note if it were not a diamond.

```
\relative d'' {
  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:

```
d''2^\flageolet d''_\flageolet
```
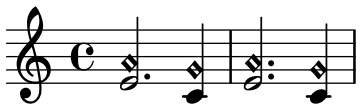


Artificial harmonics

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.

```
\relative e' {
  <e a\harmonic>2.  <c g'\harmonic>4
  \set harmonicDots = ##t
  <e a\harmonic>2.  <c g'\harmonic>4
}
```



See also

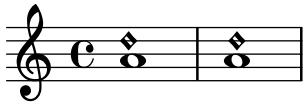
Music Glossary: Section “harmonics” in *Music Glossary*.

Notation Reference: Section 1.4.1 [Special note heads], page 43, Section 11.1.1 [References for unfretted strings], page 427.

Known issues and warnings

If you want to center a harmonic on a whole note, make it part of a chord.

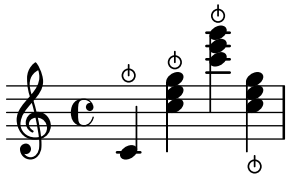
```
{
  << { d''1\harmonic } \\ { a'1 } >>
  <a' d''\harmonic>1
}
```
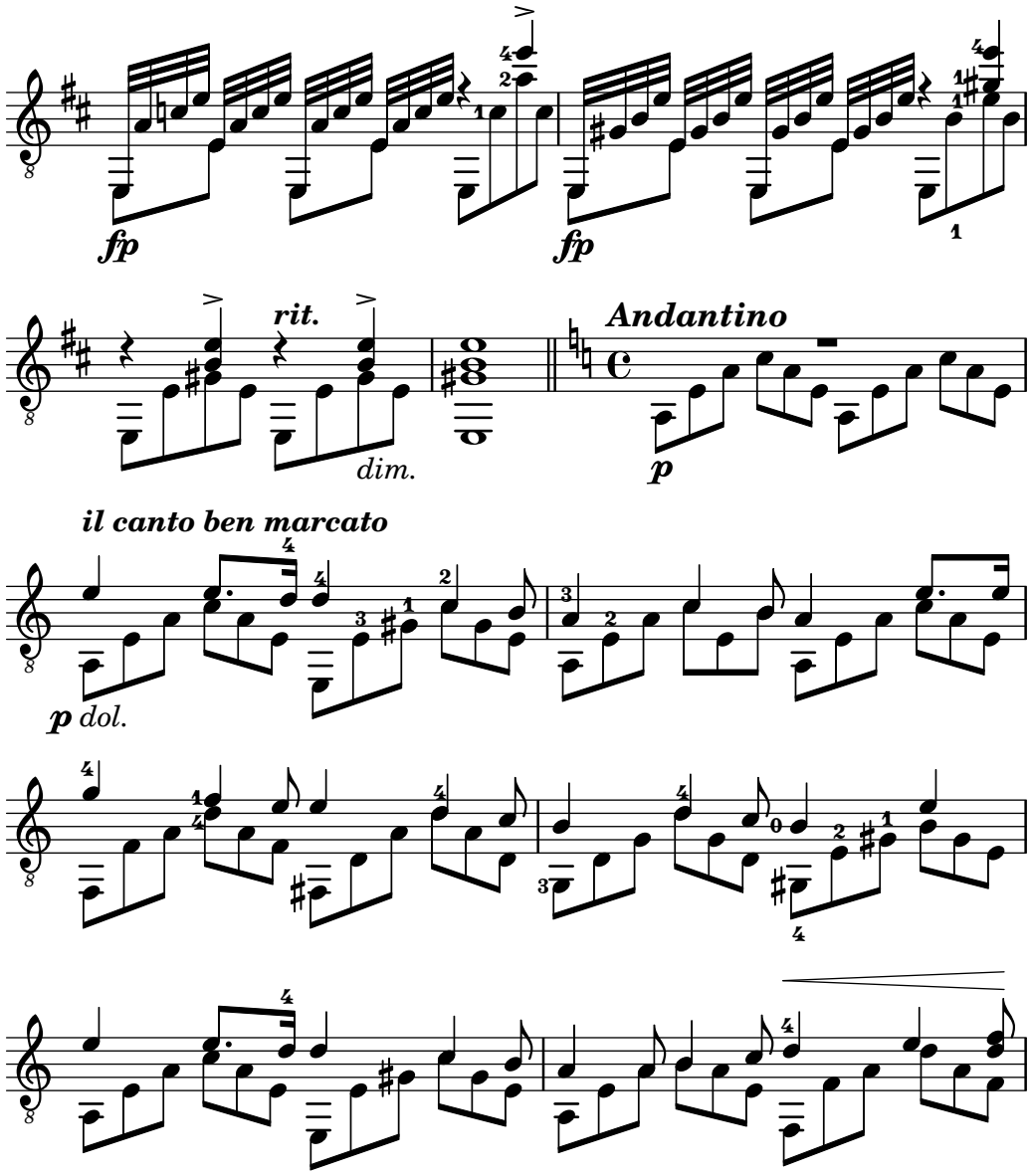


11.1.4 Snap (Bartók) pizzicato

A *snap pizzicato* (also known as “Bartok pizz”) is a type of pizzicato where the string is deliberately plucked upwards (rather than sideways) such that it hits the fingerboard.

```
\relative {
  c'4\snappizzicato
  <c' e g>4\snappizzicato
  <c' e g>4^\snappizzicato
  <c, e g>4_\snappizzicato
}
```



12 Fretted string instruments



This section discusses several aspects of music notation that are unique to fretted string instruments.

12.1 Common notation for fretted strings

This section discusses common notation that is unique to fretted string instruments.

12.1.1 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 Section 7.1.2 [Fingering instructions], page 282.

- Instructions for *Laissez vibrer* ties as well as ties on arpeggios and tremolos can be found in Section 2.1.4 [Ties], page 63.
- Instructions for handling multiple voices can be found in Section 5.2.3 [Collision resolution], page 227.
- Instructions for indicating harmonics can be found in Section 11.1.3 [Harmonics], page 428.

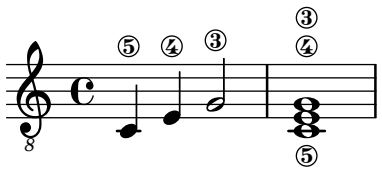
See also

Notation Reference: Section 7.1.2 [Fingering instructions], page 282, Section 2.1.4 [Ties], page 63, Section 5.2.3 [Collision resolution], page 227, Section 6.3.1 [Instrument names], page 262, Section 5.2.6 [Writing music in parallel], page 238, Section 3.3.2 [Arpeggio], page 182, Section B.13 [List of articulations], page 928, Section 1.3.1 [Clef], page 19, Section 1.3.4 [Instrument transpositions], page 30.

12.1.2 String number indications

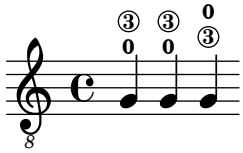
The string on which a note should be played may be indicated by appending `\number` to a note.

```
\clef "treble_8"
c4\5 e\4 g2\3
<c\5 e\4 g\3>1
```



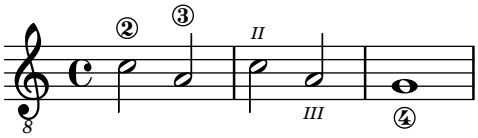
When fingerings and string indications are used together, their vertical placement can be controlled with the `script-priority` property, see [Controlling the vertical ordering of scripts], page 158, and Section B.18 [Default values for `script-priority`], page 938; the ordering in the source code has no influence.

```
\clef "treble_8"
g4\3-0
g-0\3
g-\tweak script-priority 200 -0 \3
```



String numbers may also, as is customary with unfretted strings, be printed in Roman numerals and placed below the staff rather than above.

```
\clef "treble_8"
c'2\2
a\3
\romanStringNumbers
c'\2
\set stringNumberOrientations = #'(down)
a\3
\arabicStringNumbers
g1\4
```



Most behaviors of string number indications (namely, the `StringNumber` object), including their placement, may be set in the same way as fingerings: see Section 7.1.2 [Fingering instructions], page 282.

Predefined commands

`\arabicStringNumbers`, `\romanStringNumbers`.

See also

Notation Reference: Section 7.1.2 [Fingering instructions], page 282.

Snippets: Section “Fretted string instruments” in *Snippets*.

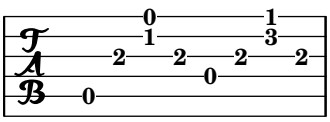
Internals Reference: Section “StringNumber” in *Internals Reference*, Section “Fingering” in *Internals Reference*.

12.1.3 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 letter-like symbols in historical intavolatura). The staff lines in tablature indicate the string on which the note is to be played, and a number placed on a staff line indicates the fret at which the corresponding string is to be pressed. Notes that are to be played simultaneously are vertically aligned.

By default, string 1 is the highest string, and corresponds to the top line on the `TabStaff`. The tuning of the `TabStaff` strings 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 {
  a,8 a' <c e> a
  d,8 a' <d f> a
}
```

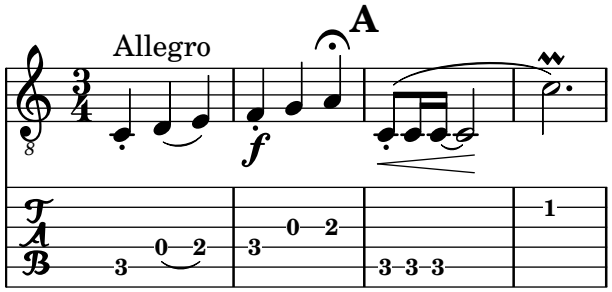


Default tablatures do not contain any symbols for tone duration nor any other musical symbols such as expressive marks, for example.

```
symbols = {
  \time 3/4
  c4-.^"Allegro" d( e)
  f4-.\f g a^\fermata
  \mark \default
  c8_.\<\( c16 c~ 2\!
  c'2.\prall\)
}

\score {
  <<
```

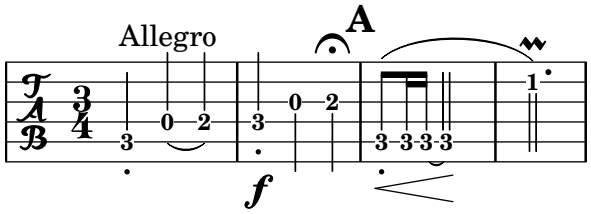
```
    \new Staff { \clef "G_8" \symbols }
    \new TabStaff { \symbols }
  >>
}
```



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.

```
symbols = {
  \time 3/4
  c4-.^"Allegro" d( e)
  f4-.\f g a^\fermata
  \mark \default
  c8_.\<\( c16 c~ 2\!
  c'2.\prall\)
}

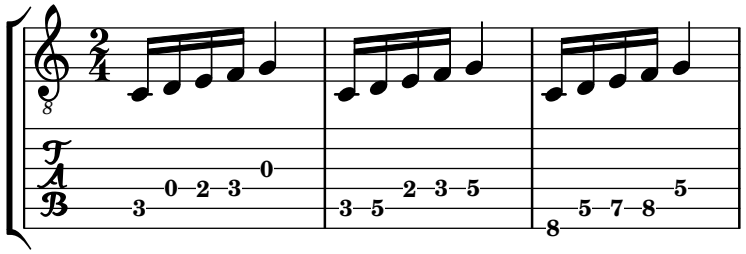
\score {
  \new TabStaff {
    \tabFullNotation
    \symbols
  }
}
```



By default pitches are assigned to the lowest playing position on the fretboard (first position). Open strings are automatically preferred. If you would like a certain pitch to be played on a specific string you can add a string number indication to the pitch name. If you don't want to have string number indications appear in traditional notation, you can override the respective stencil. Usually it will be more comfortable to define the playing position by using the value of `minimumFret`. The default value for `minimumFret` is 0.

Even when `minimumFret` is set, open strings are used whenever possible. This behavior can be changed by setting `restrainOpenStrings` to `#t`.

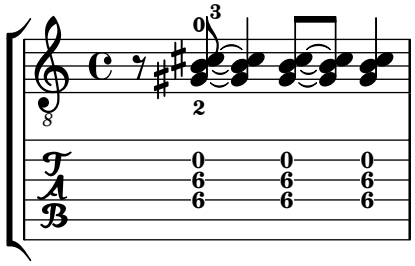
```
\layout { \omit Voice.StringNumber }
\new StaffGroup <<
   \new Staff \relative {
     \clef "treble_8"
     \time 2/4
     c16 d e f g4
     c,16\5 d\5 e\4 f\4 g4\4
     c,16 d e f g4
   }
   \new TabStaff \relative {
     c16 d e f g4
     c,16\5 d\5 e\4 f\4 g4\4
     \set TabStaff.minimumFret = 5
     \set TabStaff.restrainOpenStrings = ##t
     c,16 d e f g4
   }
>>
```



Chord constructs can be repeated by the chord repetition symbol 'q', (see Section 5.1.2 [Chord repetition], page 218, for more). In combination with tabulatures, its behavior of removing string and fingering numbers alongside with other events may lead to unwanted results, in particular different fret positions. The command `\tabChordRepeats` keeps the fingering consistent across repetitions. In the following example, the default fingering for this chord (without fingering indications) would be 'gis' on 4th string, 'b' on 3rd string, and 'cis' on 2nd string. As we use `b-0` in the input, 'b' is on the second string, and 'cis' moves to the 3rd string. `\tabChordRepeats` allows to keep the same fingering in the following q chords:

```
guitar = \relative {
  r8 <gis-2 cis-3 b-0>~ q4 q8~ 8 q4
}

\new StaffGroup <<
  \new Staff {
    \clef "treble_8"
    \guitar
  }
  \new TabStaff {
    \tabChordRepeats \guitar
  }
>>
```

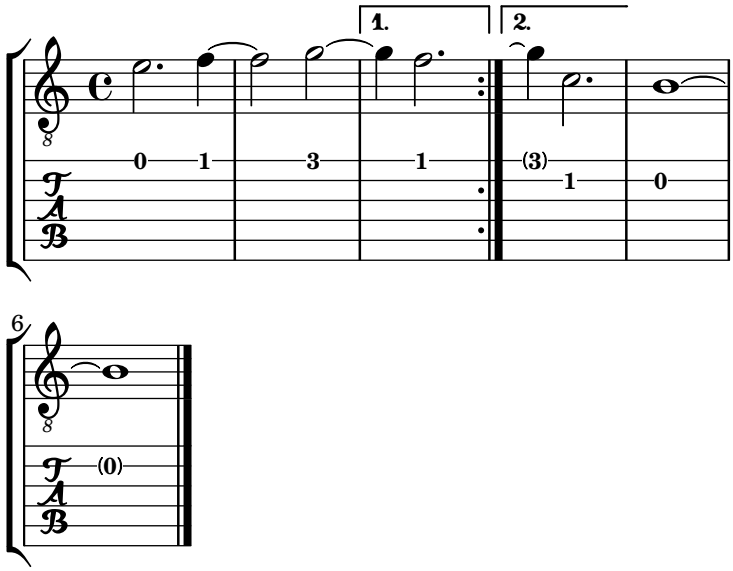


Ties over a line break are parenthesized by default. The same holds for the second alternative of a repeat.

```
ties = \relative {
  \repeat volta 2 {
    e'2. f4~
    2 g2~
  }
  \alternative {
     \volta 1 { g4 f2. }
     \volta 2 { g4\repeatTie c,2. }
  }
  b1~
  \break
  b1
  \bar "|."
}

\score {
  <<
    \new StaffGroup  <<
      \new Staff {
        \clef "treble_8"
        \ties
      }
      \new TabStaff {
        \ties
      }
    >>
  >>
  \layout {
```

```
  indent = 0
  ragged-right = ##t
  }
}
```

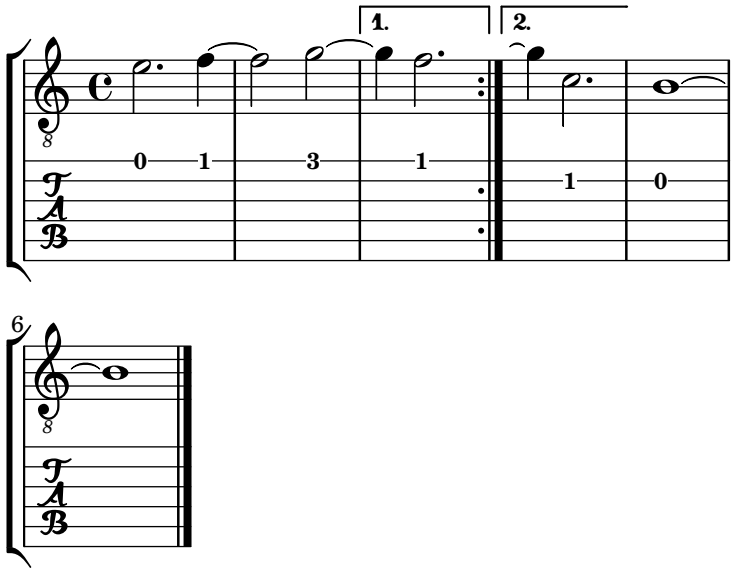


The command \hideSplitTiedTabNotes cancels the behavior of engraving fret numbers in parentheses:

```
ties = \relative {
  \repeat volta 2 {
    e'2. f4~
    2 g2~ }
  \alternative {
    \volta 1 { g4 f2. }
    \volta 2 { g4\repeatTie c,2. }
  }
  b1~
  \break
  b1
  \bar "|."
}

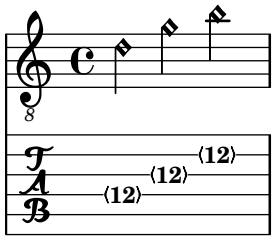
\score {
  <<
    \new StaffGroup  <<
      \new Staff {
        \clef "treble_8"
        \ties
      }
      \new TabStaff {
      \hideSplitTiedTabNotes
        \ties
      }
    >>
  >>
  \layout {
```

```
  indent = 0
  ragged-right = ##t
  }
}
```



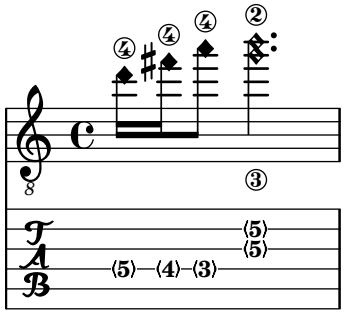
Harmonic indications can be added to tablature notation as sounding pitches:

```
\layout { \omit Voice.StringNumber }
firstHarmonic = {
  d'4\4\harmonic
  g'4\3\harmonic
  b'2\2\harmonic
}
\score {
  <<
    \new Staff {
      \clef "treble_8"
      \firstHarmonic
    }
    \new TabStaff { \firstHarmonic }
  >>
}
```



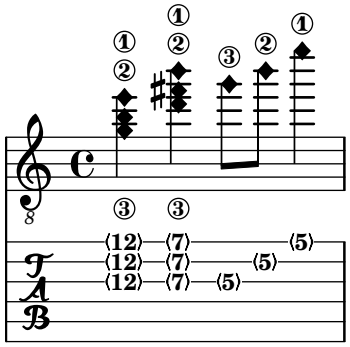
Note that the command \harmonic must always be attached to single notes (possibly inside of a chord) instead of whole chords. It only makes sense for open-string harmonics in the 12th fret. All other harmonics should be calculated by LilyPond. This can be achieved by indicating the fret where a finger of the fretting hand should touch a string.

```
fretHarmonics = {
  \harmonicByFret 5 d16\4
  \harmonicByFret 4 d16\4
  \harmonicByFret 3 d8\4
  \harmonicByFret 5 <g\3 b\2>2.
}
\score {
  <<
    \new Staff {
      \clef "treble_8"
      \fretHarmonics
    }
    \new TabStaff { \fretHarmonics }
  >>
}
```



Alternatively, harmonics can be computed by defining the ratio of string lengths above and below the harmonic fingering.

```
ratioHarmonics = {
  \harmonicByRatio #1/2 <g\3 b\2 e'\1>4
  \harmonicByRatio #1/3 <g\3 b\2 e'\1>4
  \harmonicByRatio #1/4 { g8\3 b8\2 e'4\1 }
}
\score {
  <<
    \new Staff {
      \clef "treble_8"
      \ratioHarmonics
    }
    \new TabStaff { \ratioHarmonics }
  >>
}
```



String bendings can be added to tablature notation. A bending is introduced by appending \^ to the note or chord to be bent; it terminates automatically at the next note or chord. Available are the following styles: the default prints a curve with an arrow head up or down, 'hold a dashed horizontal line, 'pre-bend a vertical line with an arrow head, and 'pre-bend-hold a vertical line with an arrow head continued by a dashed line.

The commands \bendHold, \preBend, and \preBendHold are shortcuts for setting the bending style.

```
bend-styles = {
  <>^"default"
  f'4\^ g'4\^ f'2

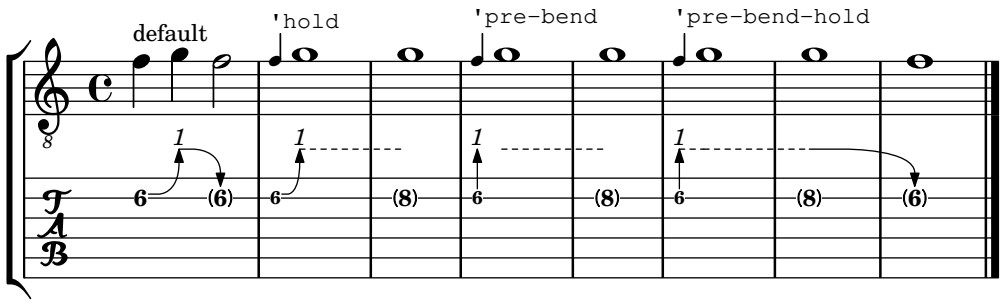
  <>^\markup \typewriter "'hold"
  \grace f'4\^ g'1\bendHold \^ g'1

  <>^\markup \typewriter "'pre-bend"
  \grace f'4\preBend \^ g'1\bendHold \^ g'1

  <>^\markup \typewriter "'pre-bend-hold"
  \grace f'4\preBendHold \^ g'1\bendHold \^ g'1\^ f'

  \bar "|."
}

\score {
  \new StaffGroup
  <<
    \new Staff {
      \override TextScript.font-size = -2
      \clef "G_8"
      \bend-styles
    }
    \new TabStaff \bend-styles
  >>
  \layout {
    \context {
      \Voice
      \omit StringNumber
    }
    \context {
      \TabStaff
      minimumFret = 5
    }
  }
}
```



Open strings are usually not bent. To have them bent as well set the property `bend-me` to `#t`. To exclude other notes from being bent set it to `#f`.

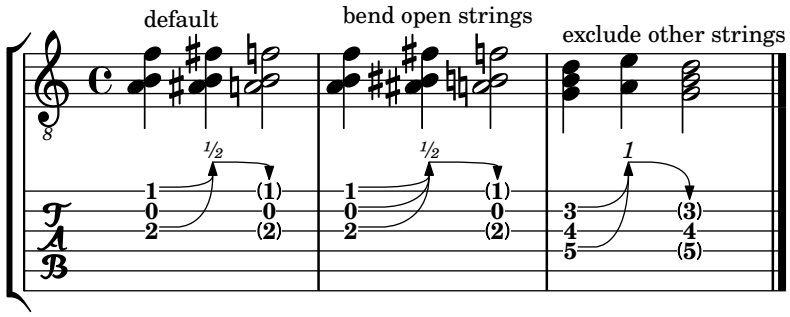
```
mus = {
  <>^"default"
  <a b f'>4\^
  <ais b fis'>\^
  <a b f'>2

  <>^"bend open strings"
  <a \tweak bend-me ##t b f'>4\^
  <ais \tweak bend-me ##t bis fis'>\^
  <a b f'>2

  <>^"exclude other strings"
  <g \tweak bend-me ##f b\3 d'>4\^
  <a e'\2 >\^
  <g \tweak bend-me ##f b\3 d'>2

  \bar "|."
}

\score {
  \new StaffGroup
  <<
    \new Staff {
      \override TextScript.font-size = -2
      \clef "G_8"
      \mus
    }
    \new TabStaff \mus
  >>
  \layout {
    \context {
      \Voice
      \omit StringNumber
    }
  }
}
```



For consecutive bendings the starting bend may need to have an appropriate setting for `details.successive-level`. For convenience there is the function `bendStartLevel`, taking an integer.

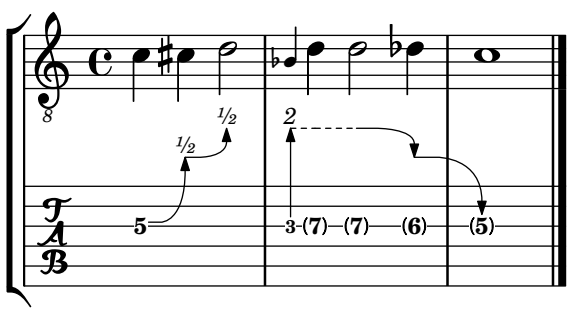
```
printNext = -\tweak details.target-visibility ##t \etc

mus = {
  c'4\3\^ cis'\3 \^ d'2\3

  \grace bes4\3\preBendHold \bendStartLevel 2 \printNext \^
  d'4\3\bendHold \^ d'2\3\^ des'4\3 \^ c'1\3

  \bar "|."
}

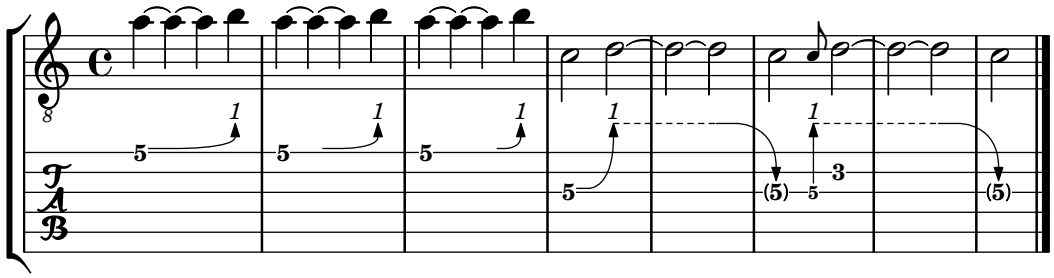
\score {
  \new StaffGroup
  <<
    \new Staff {
      \override TextScript.font-size = -2
      \clef "G_8"
      \mus
    }
    \new TabStaff \mus
  >>
  \layout {
    \context {
      \Voice
      \omit StringNumber
    }
  }
}
```



Per default the BendSpanner ends at the following note or chord even if it is tied to the starting note or chord. A single NoteColumn may be skipped by using \skipNC. A group of NoteColumns can be skipped by using \skipNCs at the beginning and \endSkipNCs at the end.

```
bends-with-ties-and-skips = {
  a'4~\^ \skipNC a'4~ \skipNC a'4 b'4
  a'4~ a'4~\^ \skipNC a'4 b'4
  a'4~ a'4~ a'4\^ b'4
  c'2\^ d'~ \bendHold \^ \skipNC d'~ d'\^ c'
  \grace { c'8-\preBendHold \^ }
  \skipNCs d'2~ d'2~ \endSkipNCs d'\^ c'2
  \bar "|."
}

\score {
  \new StaffGroup
  <<
    \new Staff {
      \clef "G_8"
      \bends-with-ties-and-skips
    }
    \new TabVoice \bends-with-ties-and-skips
  >>
  \layout {
    \context {
      \Voice
      \omit StringNumber
    }
    \context {
      \TabStaff
      minimumFret = 3
      restrainOpenStrings = ##t
    }
  }
}
```



Predefined commands

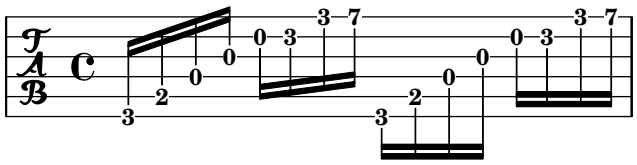
\skipNCs, \skipNC, \endSkipNCs.

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.concaveness = 10000
    g,,16 b d g b d g b
  }
}
```



Polyphony in tablature

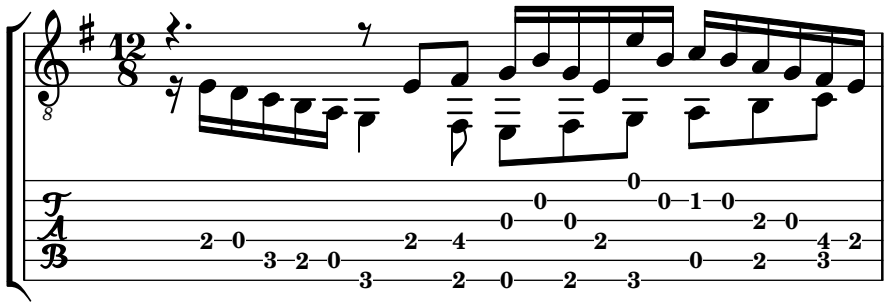
Polyphony is created the same way in a TabStaff as in a regular staff.

```
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"
      \new Voice = "upper" \upper
      \new Voice = "lower" \lower
    >>

    \new TabStaff = "guitar tab" <<
      \new TabVoice = "upper" \upper
      \new TabVoice = "lower" \lower
    >>
  >>
}
```



Open-string harmonics in tablature

This snippet demonstrates open-string harmonics.

```
openStringHarmonics = {
  \textSpannerDown
  \override TextSpanner.staff-padding = 3
  \override TextSpanner.dash-fraction = 0.3
  \override TextSpanner.dash-period = 1

  % first harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "1st harm. "
  \harmonicByFret 12 e,2\6\startTextSpan
  \harmonicByRatio #1/2 e,\6\stopTextSpan

  % second harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "2nd harm. "
  \harmonicByFret 7 e,\6\startTextSpan
  \harmonicByRatio #1/3 e,\6
  \harmonicByFret 19 e,\6
  \harmonicByRatio #2/3 e,\6\stopTextSpan
  %\harmonicByFret 19 < e,\6 a,\5 d\4 >
  %\harmonicByRatio #2/3 < e,\6 a,\5 d\4 >

  % third harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "3rd harm. "
  \harmonicByFret 5 e,\6\startTextSpan
  \harmonicByRatio #1/4 e,\6
  \harmonicByFret 24 e,\6
  \harmonicByRatio #3/4 e,\6\stopTextSpan
  \break

  % fourth harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "4th harm. "
  \harmonicByFret 4 e,\6\startTextSpan
  \harmonicByRatio #1/5 e,\6
  \harmonicByFret 9 e,\6
  \harmonicByRatio #2/5 e,\6
  \harmonicByFret 16 e,\6
  \harmonicByRatio #3/5 e,\6\stopTextSpan
```

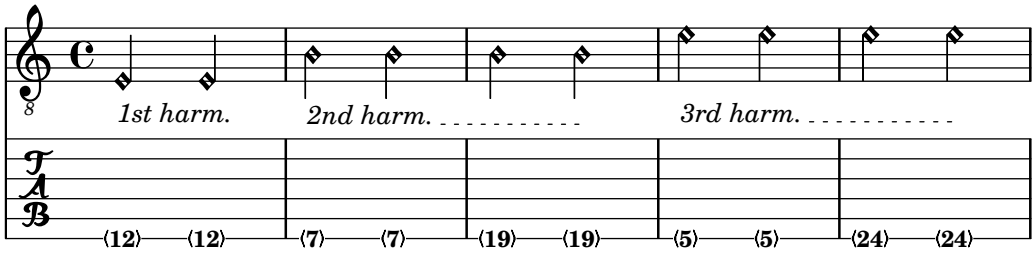
```
  % fifth harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "5th harm. "
  \harmonicByFret 3 e,\6\startTextSpan
  \harmonicByRatio #1/6 e,\6\stopTextSpan
  \break

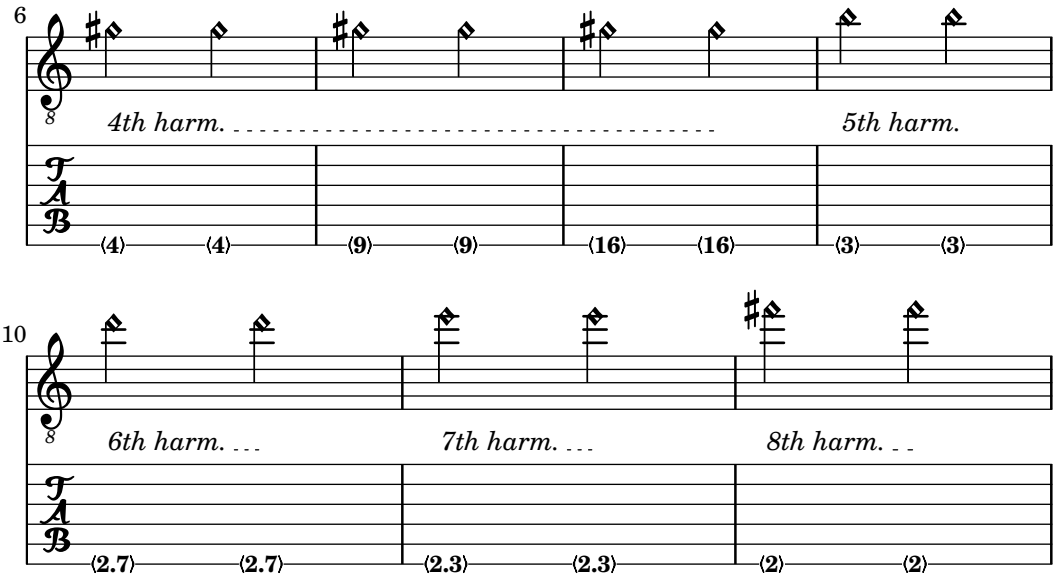
  % sixth harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "6th harm. "
  \harmonicByFret 2.7 e,\6\startTextSpan
  \harmonicByRatio #1/7 e,\6\stopTextSpan

  % seventh harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "7th harm. "
  \harmonicByFret 2.3 e,\6\startTextSpan
  \harmonicByRatio #1/8 e,\6\stopTextSpan

  % eighth harmonic
  \override TextSpanner.bound-details.left.text =
    \markup\small "8th harm. "
  \harmonicByFret 2 e,\6\startTextSpan
  \harmonicByRatio #1/9 e,\6\stopTextSpan
}

\score {
  <<
    \new Staff \with { \omit StringNumber } {
      \new Voice {
        \clef "treble_8"
        \openStringHarmonics
      }
    }
    \new TabStaff {
      \new TabVoice {
        \openStringHarmonics
      }
    }
  >>
}
```





Fretted-string harmonics in tablature

The following demonstrates fretted-string harmonics in a tablature.

```
pinchedHarmonics = {
  \textSpannerDown
  \override TextSpanner.bound-details.left.text =
    \markup {\halign #-0.5 \teeny "PH" }
  \override TextSpanner.style = #'dashed-line
  \override TextSpanner.dash-period = 0.6
  \override TextSpanner.bound-details.right.attach-dir = 1
  \override TextSpanner.bound-details.right.text =
    \markup { \draw-line #'(0 . 1) }
  \override TextSpanner.bound-details.right.padding = -0.5
}

harmonics = {
  % artificial harmonics (AH)
  \textLengthOn
  <\parenthesize b b'\harmonic>4_\markup { \teeny "AH 16" }
  <\parenthesize g g'\harmonic>4_\markup { \teeny "AH 17" }
  <\parenthesize d' d''\harmonic>2_\markup { \teeny "AH 19" }

  % pinched harmonics (PH)
  \pinchedHarmonics
  <a'\harmonic>2\startTextSpan
  <d''\harmonic>4
  <e'\harmonic>4\stopTextSpan

  % tapped harmonics (TH)
  <\parenthesize g\4 g'\harmonic>4_\markup { \teeny "TH 17" }
  <\parenthesize a\4 a'\harmonic>4_\markup { \teeny "TH 19" }
  <\parenthesize c'\3 c''\harmonic>2_\markup { \teeny "TH 17" }

  % touch harmonics (TCH)
  a4( <e''\harmonic>2. )_\markup { \teeny "TCH" }
```

```
}

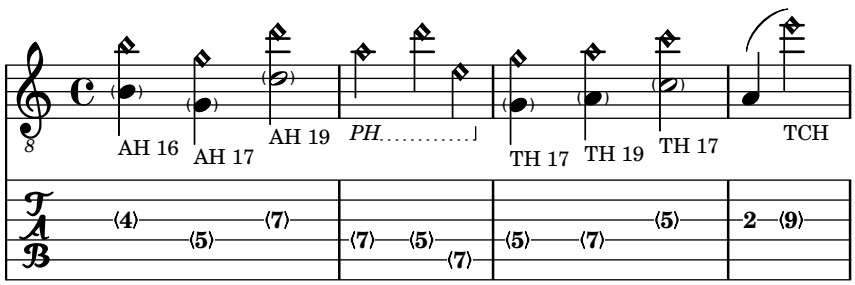
frettedStrings = {
  % artificial harmonics (AH)
  \harmonicByFret 4 g4\3
  \harmonicByFret 5 d4\4
  \harmonicByFret 7 g2\3

  % pinched harmonics (PH)
  \harmonicByFret 7 d2\4
  \harmonicByFret 5 d4\4
  \harmonicByFret 7 a4\5

  % tapped harmonics (TH)
  \harmonicByFret 5 d4\4
  \harmonicByFret 7 d4\4
  \harmonicByFret 5 g2\3

  % touch harmonics (TCH)
  a4 \harmonicByFret 9 g2.\3
}

\score {
  <<
    \new Staff
    \with { \omit StringNumber } {
      \new Voice {
        \clef "treble_8"
        \harmonics
      }
    }
    \new TabStaff {
      \new TabVoice {
        \frettedStrings
      }
    }
  >>
}
```



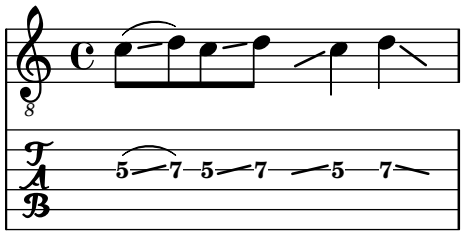
Slides in tablature

Slides can be typeset in both `Staff` and `TabStaff` contexts.

```
slides = {
  c'8\3(\glissando d'8\3)
  c'8\3\glissando d'8\3
  \hideNotes
  \grace { g16\glissando }
  \unHideNotes
  c'4\3
  \afterGrace d'4\3\glissando {
  \stemDown \hideNotes
  g16 }
  \unHideNotes
}

\score {
  <<
    \new Staff { \clef "treble_8" \slides }
    \new TabStaff { \slides }
  >>

  \layout {
    \context {
      \Score
      \override Glissando.minimum-length = 4
      \override Glissando.springs-and-rods =
                  #ly:spanner::set-spacing-rods
      \override Glissando.thickness = 2
      \omit StringNumber
      % or:
      %\override StringNumber.stencil = ##f
    }
  }
}
```



Chord glissando in tablature

Slides for chords are indicated by default in both `Staff` and `TabStaff`.

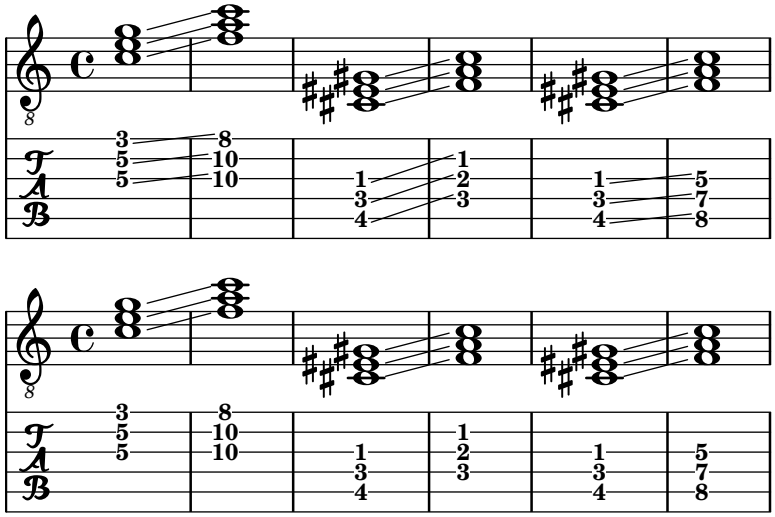
String numbers may be necessary for `TabStaff` because automatic string calculations are different for chords and for single notes.

```
myMusic = \relative c' {
  <c e g>1 \glissando <f a c>
  <cis, eis gis>1 \glissando <f a c>
  <cis eis gis>1 \glissando <f a c\3>
}

\score {
```

```
  <<
    \new Staff {
      \clef "treble_8"
      \omit StringNumber
      \myMusic
    }
    \new TabStaff \myMusic
  >>
}

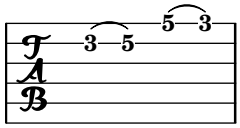
\score {
  <<
    \new Staff {
      \clef "treble_8"
      \omit StringNumber
      \myMusic
    }
    \new TabStaff \with { \override Glissando.style = #'none } {
      \myMusic
    }
  >>
}
```



Hammer-on and pull-off

Hammer-on and pull-off can be obtained using slurs.

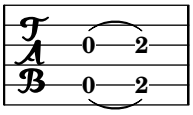
```
\new TabStaff {
  \relative c' {
    d4( e\2)
    a( g)
  }
}
```



Hammer-on and pull-off using voices

The arc of hammer-on and pull-off is upwards in voices one and three and downwards in voices two and four:

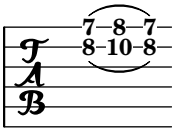
```
\new TabStaff {
  \relative c' {
    << { \voiceOne g2( a) }
    \\ { \voiceTwo a,( b) }
    >> \oneVoice
  }
}
```



Hammer-on and pull-off using chords

When using hammer-on or pull-off with chorded notes, only a single arc is drawn. However "double arcs" are possible by setting the `doubleSlurs` property to `#t`.

```
\new TabStaff {
  \relative c' {
    % chord hammer-on and pull-off
    \set doubleSlurs = ##t
    <g' b>8( <a c> <g b>)
  }
}
```



See also

Notation Reference: Section 5.1.2 [Chord repetition], page 218, Section 3.3.1 [Glissando], page 177, Section 11.1.3 [Harmonics], page 428, Section 7.1.9 [Stems], page 297, Section 4.1.1 [Written-out repeats], page 189.

Snippets: Section "Fretted string instruments" in *Snippets*.

Internals Reference: Section "TabNoteHead" in *Internals Reference*, Section "TabStaff" in *Internals Reference*, Section "TabVoice" in *Internals Reference*, Section "Beam" in *Internals Reference*.

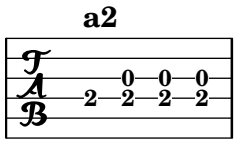
Known issues and warnings

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 }
  >>
>>
```



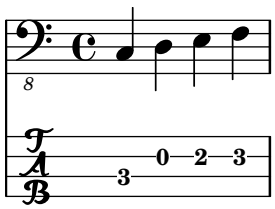
Guitar special effects are limited to harmonics and slides.

12.1.4 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, ukulele, violin, viola, cello, and double bass. LilyPond automatically sets the correct transposition for predefined tunings. The following example is for bass guitar, which sounds an octave lower than written.

```
<<
  \new Voice \with {
    \omit StringNumber
  } {
    \clef "bass_8"
    \relative {
      c,4 d e f
    }
  }
  \new TabStaff \with {
    stringTunings = #bass-tuning
  } {
    \relative {
      c,4 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 `ly/string-tunings-init.ly`.

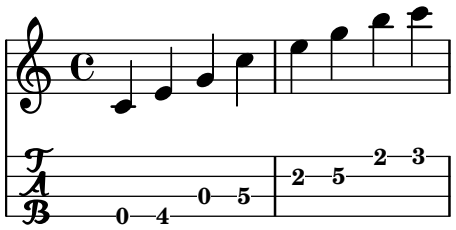
Any desired string tuning can be created. The `\stringTuning` function can be used to define a string tuning which can be used to set `stringTunings` for the current context.

Its argument is a chord construct defining the pitches of each string in the tuning. The chord construct must be in absolute octave mode, see Section 1.1.1 [Absolute octave entry], page 3.

The string with the highest number (generally the lowest string) must come first in the chord. 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 Staff.stringTunings = \stringTuning <c' g' d'' a''>
    \mynotes
  }
>>
```



The `stringTunings` property is also used by `FretBoards` to calculate automatic fret diagrams.

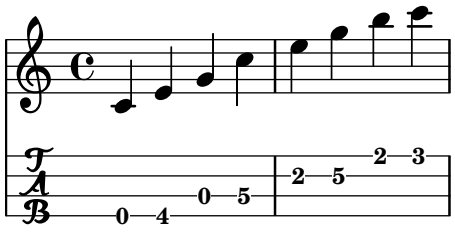
String tunings are used as part of the hash key for predefined fret diagrams (see Section 12.1.6 [Predefined fret diagrams], page 466).

The previous example could also be written as follows:

```
custom-tuning = \stringTuning <c' g' d'' a''>

mynotes = {
  c'4 e' g' c'' |
  e''4 g'' b'' c'''
}

<<
  \new Staff {
    \clef treble
    \mynotes
  }
  \new TabStaff {
    \set TabStaff.stringTunings = #custom-tuning
    \mynotes
  }
>>
```



Internally, 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 a LilyPond pitch object. Pitch objects are created with the Scheme function `ly:make-pitch` (see Section "Scheme functions" in *Internals Reference*).

`\stringTuning` creates such an object from chord input.

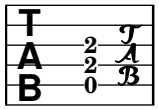
LilyPond automatically calculates the number of lines in the `TabStaff` and the number of strings in an automatically calculated `FretBoard` as the number of elements in `stringTunings`.

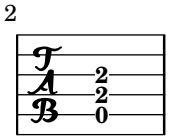
To let all `TabStaff` contexts use the same custom tuning by default, you can use

```
\layout {
  \context {
    \TabStaff
    stringTunings = \stringTuning <c' g' d'' a''>
  }
}
```

A modern tab clef can also be used.

```
\new TabStaff {
  \clef moderntab
  <a, e a>1
  \break
  \clef tab
  <a, e a>1
}
```





The modern tab clef supports tablatures from 4 to 7 strings.

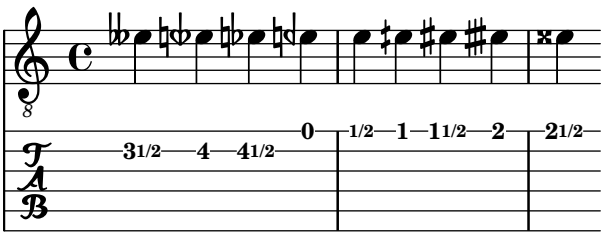
`TabStaff` may support microtones like quarter tones, which can be played using bendings. `supportNonIntegerFret = ##t` needs to be set in `Score` context. However, microtones are not supported in `FretBoards`.

```
\layout {
  \context {
    \Score
    supportNonIntegerFret = ##t
  }
}

custom-tuning = \stringTuning <e, a, d ges beh eeh'>

mus = \relative {
  eeses'4
  eeseh
  ees
  eeh
  e
  eih
  eis
  eisih
  eisis
}

<<
  \new Staff << \clef "G_8" \mus >>
  \new TabStaff \with { stringTunings = \custom-tuning } \mus
>>
```



See also

Notation Reference: Section 1.1.1 [Absolute octave entry], page 3, Section 12.1.6 [Predefined fret diagrams], page 466.

Installed Files: `ly/string-tunings-init.ly`, `scm/tablature.scm`.

Snippets: Section "Fretted string instruments" in *Snippets*.

Internals Reference: Section "Tab_note_heads_engraver" in *Internals Reference*, Section "Scheme functions" 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.

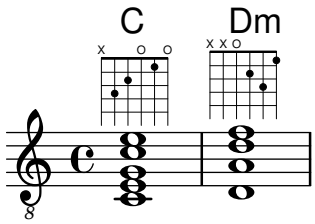
12.1.5 Fret diagram markups

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 syntax of the

different markup strings used to define fret diagrams are found at Section A.1.6 [Instrument-specific markup], page 864.

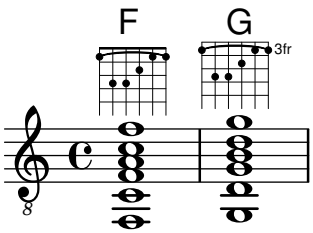
The standard fret diagram markup string indicates the string number and the fret number for each dot to be placed on the string. In addition, open and unplayed (muted) strings can be indicated.

```
<<
  \new ChordNames {
    \chordmode {
      c1 d:m
    }
  }
  \new 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'>1^\markup {
      \fret-diagram "6-x;5-x;4-o;3-2;2-3;1-1;"
    }
  }
>>
```



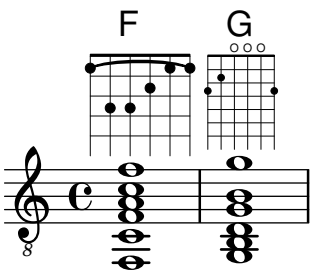
Barré indications can be added to the diagram from the fret diagram markup string.

```
<<
  \new ChordNames {
     \chordmode {
       f1 g
     }
  }
  \new 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'>1^\markup {
      \fret-diagram "c:6-1-3;6-3;5-5;4-5;3-4;2-3;1-3;"
    }
  }
>>
```



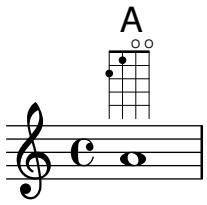
The size of the fret diagram, and the number of frets in the diagram can be changed in the fret diagram markup string.

```
<<
  \new ChordNames {
     \chordmode {
       f1 g
     }
  }
  \new 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'>1^\markup {
      \fret-diagram "h:6;6-3;5-2;4-o;3-o;2-o;1-3;"
    }
  }
>>
```



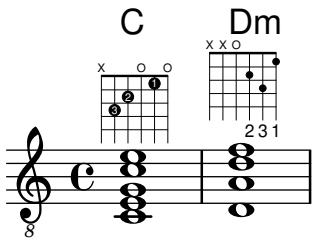
The number of strings in a fret diagram can be changed to accommodate different instruments such as banjos and ukuleles with the fret diagram markup string.

```
<<
  \new ChordNames {
    \chordmode {
      a1
    }
  }
  \new Staff {
    % An 'A' chord for ukulele
    a'1^\markup {
      \fret-diagram "w:4;4-2-2;3-1-1;2-o;1-o;"
    }
  }
>>
```



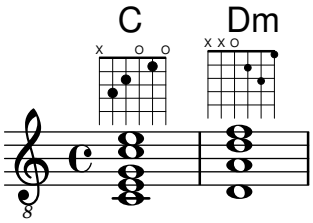
Fingering indications can be added, and the location of fingering labels can be controlled by the fret diagram markup string.

```
<<
  \new ChordNames {
    \chordmode {
      c1 d:m
    }
  }
  \new 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'>1^\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.

```
<<
  \new ChordNames {
    \chordmode {
      c1 d:m
    }
  }
  \new 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'>1^\markup {
      \fret-diagram "p:0.2;6-x;5-x;4-o;3-2;2-3;1-1;"
    }
  }
>>
```



Fret-diagrams may be printed left-handed

```
\markup
  \center-column {
    "C"
    "(left-handed)"
     \override #`(fret-diagram-details . ((handedness . ,LEFT)))
     \fret-diagram "6-x;5-3-3;4-2-2;3-o;2-1;1-o;"
  }
```

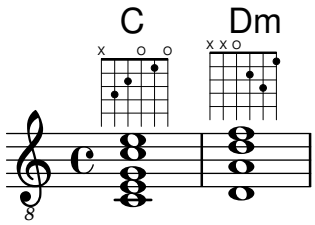
C
(left-handed)



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.

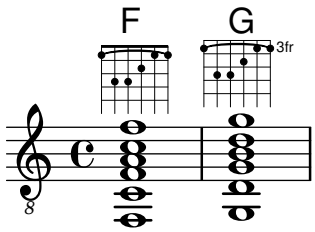
```
<<
  \new ChordNames {
    \chordmode {
      c1 d:m
    }
  }
  \new Staff {
    \clef "treble_8"
    <c e g c' e'>1^\markup {
```

```
      \fret-diagram-terse "x;3;2;o;1;o;"
    }
    <d a d' f'>1^\markup {
      \fret-diagram-terse "x;x;o;2;3;1;"
    }
  }
>>
```



Barré indicators can be included in the `\fret-diagram-terse` markup string.

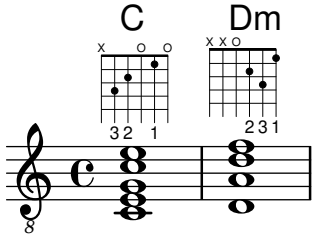
```
<<
  \new ChordNames {
    \chordmode {
      f1 g
    }
  }
  \new Staff {
    \clef "treble_8"
    <f, c f a c' f'>1^\markup {
      \fret-diagram-terse "1-(;3;3;2;1;1-);"
    }
    <g, d g b d' g'>1^\markup {
      \fret-diagram-terse "3-(;5;5;4;3;3-);"
    }
  }
>>
```



Fingering indications can be included in the `\fret-diagram-terse` markup string.

```
<<
  \new ChordNames {
    \chordmode {
      c1 d:m
    }
  }
  \new 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'>1^\markup {
      \fret-diagram-terse "x;x;o;2-2;3-3;1-1;"
    }
  }
>>
```

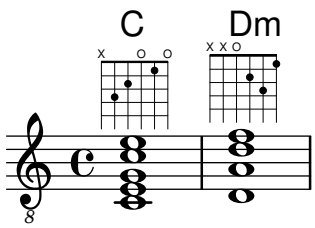


Other fret diagram properties must be adjusted using `\override` when using the `\fret-diagram-terse` markup.

Only one indication per string can be included in a `\fret-diagram-terse` markup. To have multiple indications per string use a fret diagram or `\fret-diagram-verbose` 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.

```
<<
    \new ChordNames {
      \chordmode {
        c1 d:m
      }
    }
  \new 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'>1^\markup {
      \fret-diagram-verbose #'(
        (mute 6)
        (mute 5)
        (open 4)
        (place-fret 3 2)
        (place-fret 2 3)
        (place-fret 1 1)
      )
    }
  }
>>
```



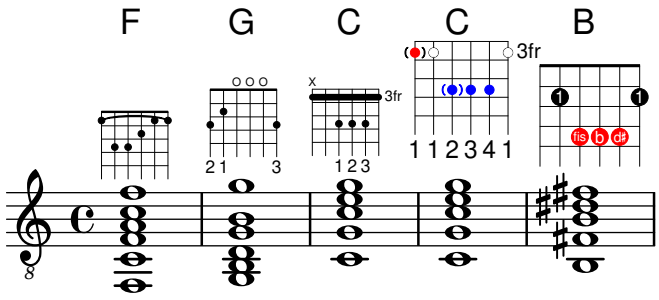
Fingering indications and barrés can be included in a `\fret-diagram-verbose` markup string. Unique to the `\fret-diagram-verbose` interface is a capo indication that can be placed on the fret diagram. The capo indication is a thick bar that covers all strings. The fret with the capo will be the lowest fret in the fret diagram.

Fingering indication dots can be colored as well as parenthesized; the parenthesis's color can also be altered independently.

Markups can be placed into the dots as well.

```
<<
    \new ChordNames {
      \chordmode {
        f1 g c c b
      }
    }
  \new 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'>1^\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 g c' e' g'>1^\markup {
      \fret-diagram-verbose #'(
        (capo 3)
        (mute 6)
        (place-fret 4 5 1)
        (place-fret 3 5 2)
```

```
        (place-fret 2 5 3)
      )
    }
    \override Voice.TextScript.size = 1.4
    <c g c' e' g'>1^\markup {
      \fret-diagram-verbose #'(
        (place-fret 6 3 1 red parenthesized default-paren-color)
        (place-fret 5 3 1 inverted)
        (place-fret 4 5 2 blue parenthesized)
        (place-fret 3 5 3 blue)
        (place-fret 2 5 4 blue)
        (place-fret 1 3 1 inverted)
      )
    }
    \override Voice.TextScript.size = 1.5
    <b, fis b dis' fis'>1^\markup
      \override #'(fret-diagram-details . ((finger-code . in-dot)))
      \fret-diagram-verbose #`(
        (place-fret 5 2 1)
        (place-fret 4 4 "fis" red)
        (place-fret 3 4 "b" red)
        (place-fret
          2 4
          ,#{ \markup
                \concat {
                  \vcenter "d"
                  \fontsize #-5
                  \musicglyph "accidentals.sharp"} #}
          red)
        (place-fret 1 2 1)
      )
  }
>>
```



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*; see Section "Fret diagrams explained and developed" in *Snippets* for an exhaustive example. For a fret diagram markup, the interface properties belong to `Voice.TextScript`.

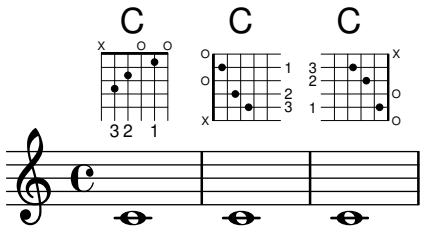
Selected snippets

Changing fret orientations

Fret diagrams can be oriented in three ways. By default the top string or fret in the different orientations will be aligned.

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

<<
  \chords {
    c1
    c1
    c1
  }
  \new FretBoards \chordmode {
    c1
    \override FretBoard.fret-diagram-details.orientation =
      #'landscape
    c1
    \override FretBoard.fret-diagram-details.orientation =
      #'opposing-landscape
    c1
  }
  \new Voice {
    c'1
    c'1
    c'
  }
>>
```



Customizing markup fret diagrams

Fret diagram properties can be modified by setting the `fret-diagram-details` property. 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
```

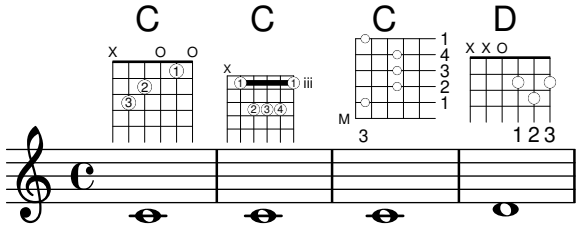
```
   % terse style
c'1^\markup { \fret-diagram-terse "x;3-3;2-2;o;1-1;o;" }

%% 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.1.6 [Instrument-specific markup], page 864.

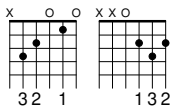
Snippets: Section “Fretted string instruments” in *Snippets*.

Internals Reference: Section “fret-diagram-interface” in *Internals Reference*.

12.1.6 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"
\new 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`.

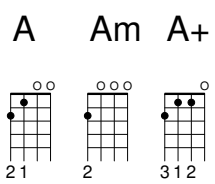
Fret diagrams for the ukulele are contained in the file `predefined-ukulele-fretboards.ly`.

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

myChords = \chordmode { a1 a:m a:aug }

\new ChordNames {
  \myChords
}

\new FretBoards {
  \set Staff.stringTunings = #ukulele-tuning
  \myChords
}
```



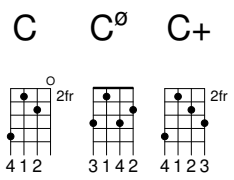
Fret diagrams for the mandolin are contained in the file `predefined-mandolin-fretboards.ly`.

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

myChords = \chordmode { c1 c:m7.5- c:aug }

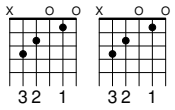
\new ChordNames {
  \myChords
}

\new FretBoards {
  \set Staff.stringTunings = #mandolin-tuning
  \myChords
}
```



Chord pitches can be entered either as simultaneous music or using chord mode (see Section 15.1.1 [Chord mode overview], page 508).

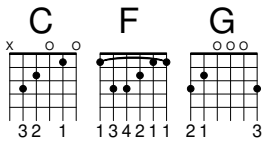
```
\include "predefined-guitar-fretboards.ly"
\new FretBoards {
  \chordmode { c1 }
  <c' e' g'>1
}
```



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
}

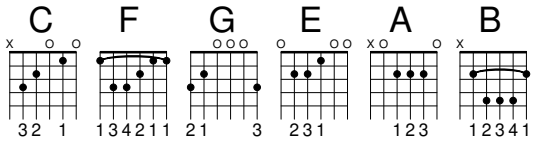
<<
  \new ChordNames {
    \mychords
  }
  \new FretBoards {
    \mychords
  }
>>
```



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 }
}
<<
  \new ChordNames {
    \mychordlist
  }
  \new FretBoards {
    \mychordlist
  }
>>
```



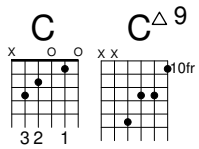
The predefined fret diagram table for guitar contains eight chords (major, minor, augmented, diminished, dominant seventh, major seventh, minor seventh, dominant ninth) for each of 17 keys.

The predefined fret diagram table for ukulele contains these chords plus an additional three chords (major sixth, suspended second, and suspended fourth).

See Section B.4 [Predefined fretboard diagrams], page 894, for a complete list of the predefined fret diagrams. If there is no entry in the table for a chord, the `FretBoards` engraver calculates a fret diagram using the automatic fret diagram functionality, see Section 12.1.7 [Automatic fret diagrams], page 477.

```
\include "predefined-guitar-fretboards.ly"
mychords = \chordmode {
  c1 c:maj9
}

<<
  \new ChordNames {
    \mychords
  }
  \new FretBoards {
    \mychords
  }
>>
```



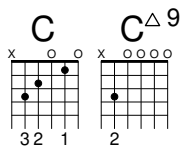
Fret diagrams can be added to the fret diagram table. To add a diagram, you must specify the hash table for the diagram, the chord for the diagram, the tuning to be used, and a definition for the diagram. Normally, the hash table will be *default-fret-table*. The diagram definition can be either a `\fret-diagram-terse` definition string or a `\fret-diagram-verbose` marking list.

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

\storePredefinedDiagram #default-fret-table
                        \chordmode { c:maj9 }
                        #guitar-tuning
                        "x;3-2;o;o;o;o;"

mychords = \chordmode {
  c1 c:maj9
}

<<
  \new ChordNames {
    \mychords
  }
  \new FretBoards {
    \mychords
  }
>>
```



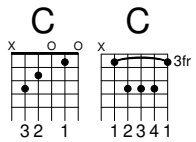
Different fret diagrams for the same chord name can be stored using different octaves of pitches. The different octave should be at least two octaves above or below the default octave, because the octaves above and below the default octave are used for transposing fretboards.

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

\storePredefinedDiagram #default-fret-table
                        \chordmode { c'' }
                        #guitar-tuning
                        #(offset-fret 2
                           (chord-shape 'bes guitar-tuning))

mychords = \chordmode {
  c1 c''
}

<<
  \new ChordNames {
    \mychords
  }
  \new 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;"

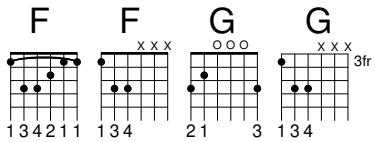
% add some new chords based on the power chord shape

\storePredefinedDiagram #default-fret-table
                        \chordmode { f'' }
                        #guitar-tuning
                        #(chord-shape 'powerf guitar-tuning)
\storePredefinedDiagram #default-fret-table
                        \chordmode { g'' }
                        #guitar-tuning
                        #(offset-fret 2
                           (chord-shape 'powerf guitar-tuning))

mychords = \chordmode {
  f1 f'' g g''
```

```
}

<<
  \new ChordNames {
    \mychords
  }
  \new 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*; see Section "Fret diagrams explained and developed" in *Snippets* for an exhaustive example. For a predefined fret diagram, the interface properties belong to `FretBoards.FretBoard`.

Selected snippets

Customizing fretboard fret diagrams

Fret diagram properties can be modified by setting the `fret-diagram-details` property. For `FretBoard` fret diagrams, overrides are applied to the `FretBoards.FretBoard` object. Like `Voice`, `FretBoards` is a bottom-level context, and therefore can be omitted in property overrides.

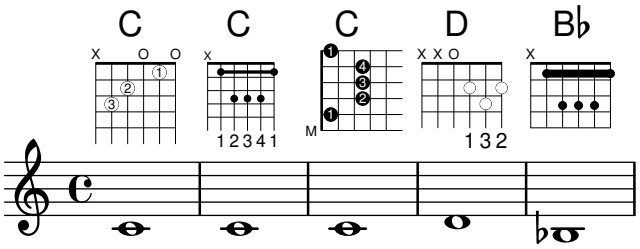
```
\include "predefined-guitar-fretboards.ly"

\storePredefinedDiagram #default-fret-table \chordmode { c' }
                        #guitar-tuning
                        "x;1-1-(;3-2;3-3;3-4;1-1-);"

% shorthand
oo = #(define-music-function
       (grob-path value)
       (list? scheme?)
       #{ \once \override $grob-path = #value #})

<<
  \new ChordNames {
    \chordmode { c1 | c | c | d | bes }
  }
  \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
      \oo FretBoard.size #1.0
```

```
      \oo FretBoard.fret-diagram-details.barre-type #'straight
      \oo FretBoard.fret-diagram-details.dot-color #'black
      \oo FretBoard.fret-diagram-details.finger-code #'below-string
      c'
      \oo FretBoard.fret-diagram-details.barre-type #'none
      \oo FretBoard.fret-diagram-details.number-type #'arabic
      \oo FretBoard.fret-diagram-details.orientation #'landscape
      \oo FretBoard.fret-diagram-details.mute-string "M"
      \oo FretBoard.fret-diagram-details.label-dir #LEFT
      \oo FretBoard.fret-diagram-details.dot-color #'black
      c'
      \oo FretBoard.fret-diagram-details.finger-code #'below-string
      \oo FretBoard.fret-diagram-details.dot-radius #0.35
      \oo FretBoard.fret-diagram-details.dot-position #0.5
      \oo FretBoard.fret-diagram-details.fret-count #3
      d
      \oo FretBoard.fret-diagram-details.barre-type #'straight
      \oo FretBoard.fret-diagram-details.finger-code #'none
      \oo FretBoard.fret-diagram-details.dot-radius #0.25
      \oo FretBoard.fret-diagram-details.dot-color #'black
      \oo FretBoard.fret-diagram-details.string-overhang #0.
      \oo FretBoard.fret-diagram-details.barre-thickness #2.
      bes
    }
  }
  \new Voice {
    c'1 | c' | c' | d' | bes
  }
>>
```



Setting up predefined fretboards for other instruments

Predefined fret diagrams can be added for new instruments in addition to the standard diagrams 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 be \included into each of your compositions.
```

```
cuatroTuning = #`(,(ly:make-pitch 0 6 0)
                  ,(ly:make-pitch 1 3 SHARP)
                  ,(ly:make-pitch 1 1 0)
                  ,(ly:make-pitch 0 5 0))

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 #default-fret-table \dSix
                        #cuatroTuning
                        "o;o;o;o;"
\storePredefinedDiagram #default-fret-table \dMajor
                        #cuatroTuning
                        "o;o;o;3-3;"
\storePredefinedDiagram #default-fret-table \aMajSeven
                        #cuatroTuning
                        "o;2-2;1-1;2-3;"
\storePredefinedDiagram #default-fret-table \dMajSeven
                        #cuatroTuning
                        "o;o;o;1-1;"
\storePredefinedDiagram #default-fret-table \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"
```

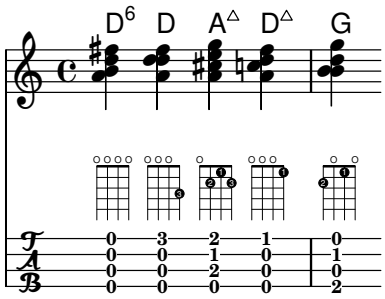
```
      }
      \relative c'' {
        \primeros
      }
    }

    \new FretBoards {
      \set Staff.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 =
        \musicLength 16
    }
  }
  \midi { }
}
```



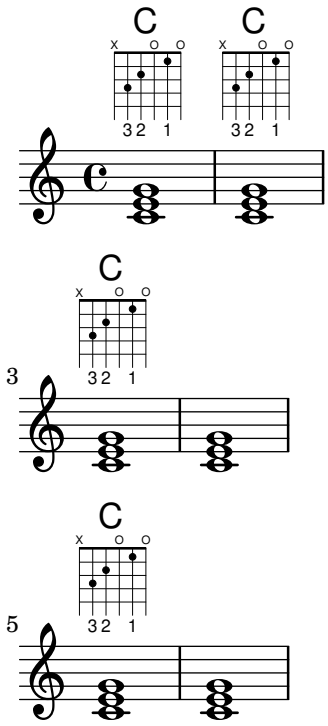
Chord changes for fretboards

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
}
```

```
<<
  \new ChordNames { \myChords }
  \new FretBoards { \myChords }
  \new Staff { \myChords }
>>
```



Fretboards alternate tables

Alternate fretboard tables can be created. These would be used in order to have alternate fretboards for a given chord. In order to use an alternate fretboard table, the table must first be created. Fretboards are then added to the table.

The created fretboard table can be blank, or it can be copied from an existing table. The table to be used in displaying predefined fretboards is selected by the property `\predefinedDiagramTable`.

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

% Make a blank new fretboard table.
#(define custom-fretboard-table-one
   (make-fretboard-table))

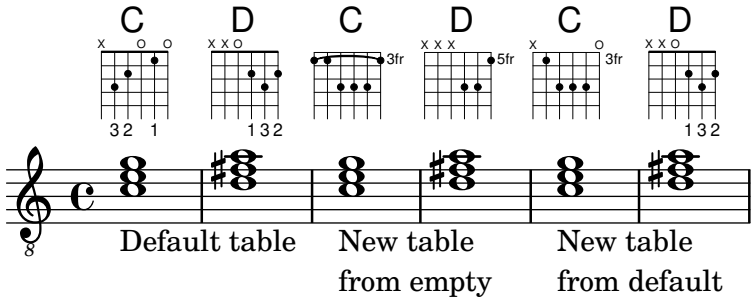
% Make a new fretboard table as a copy of `default-fret-table`.
#(define custom-fretboard-table-two
   (make-fretboard-table default-fret-table))

% Add a chord to `custom-fretboard-table-one`.
\storePredefinedDiagram #custom-fretboard-table-one
                        \chordmode {c}
                        #guitar-tuning
                        "3-(;3;5;5;5;3-);"

% Add a chord to `custom-fretboard-table-two`.
```

```
\storePredefinedDiagram #custom-fretboard-table-two
                        \chordmode {c}
                        #guitar-tuning
                        "x;3;5;5;5;o;"

<<
  \chords {
    c1 | d1 |
    c1 | d1 |
    c1 | d1 |
  }
  \new FretBoards {
    \chordmode {
      \set predefinedDiagramTable = #default-fret-table
      c1 | d1 |
      \set predefinedDiagramTable = #custom-fretboard-table-one
      c1 | d1 |
      \set predefinedDiagramTable = #custom-fretboard-table-two
      c1 | d1 |
    }
  }
  \new Staff {
    \clef "treble_8"
    <<
      \chordmode {
        c1 | d1 |
        c1 | d1 |
        c1 | d1 |
      }
      {
        s1_\markup "Default table" | s1 |
        s1_\markup \column { "New table" "from empty" } | s1 |
        s1_\markup \column { "New table" "from default" } | s1 |
      }
    >>
  }
>>
```



See also

Notation Reference: Section 12.1.4 [Custom tablatures], page 452, Section 12.1.7 [Automatic fret diagrams], page 477, Section 15.1.1 [Chord mode overview], page 508, Section B.4 [Predefined fretboard diagrams], page 894.

Installed Files: `ly/predefined-guitar-fretboards.ly`, `ly/predefined-guitar-ninth-fretboards.ly`, `ly/predefined-ukulele-fretboards.ly`, `ly/predefined-mandolin-fretboards.ly`.

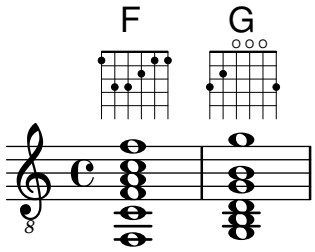
Snippets: Section "Fretted string instruments" in *Snippets*.

Internals Reference: Section "fret-diagram-interface" in *Internals Reference*.

12.1.7 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.

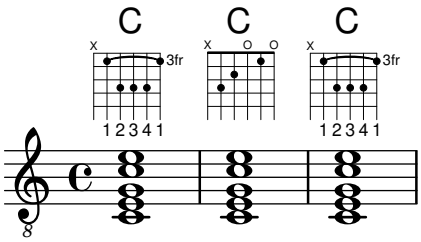
```
<<
  \new ChordNames {
    \chordmode {
      f1 g
    }
  }
  \new FretBoards {
    <f, c f a c' f'>1
    <g,\6 b, d g b g'>1
  }
  \new Staff {
    \clef "treble_8"
    <f, c f a c' f'>1
    <g, b, d g b g'>1
  }
>>
```



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 #default-fret-table
                        <c e g c' e'>
                        #guitar-tuning
                        "x;3-1-(;5-2;5-3;5-4;3-1-1-);"
<<
  \new ChordNames {
    \chordmode {
      c1 c c
    }
  }
  \new FretBoards {
    <c e g c' e'>1
```

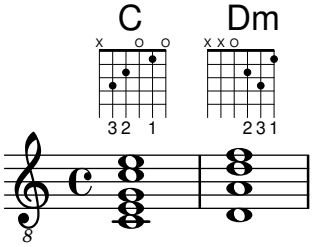
```
    \predefinedFretboardsOff
    <c e g c' e'>1
    \predefinedFretboardsOn
    <c e g c' e'>1
  }
  \new Staff {
    \clef "treble_8"
    <c e g c' e'>1
    <c e g c' e'>1
    <c e g c' e'>1
  }
>>
```



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.

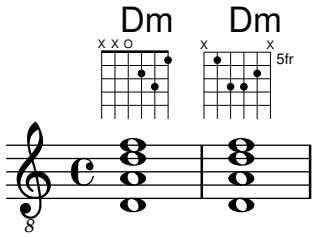
Fingerings can be added to FretBoard fret diagrams.

```
<<
  \new ChordNames {
    \chordmode {
      c1 d:m
    }
  }
  \new FretBoards {
    <c-3 e-2 g c'-1 e'>1
    <d a-2 d'-3 f'-1>1
  }
  \new Staff {
    \clef "treble_8"
    <c e g c' e'>1
    <d a d' f'>1
  }
>>
```



The minimum fret to be used in calculating strings and frets for the FretBoard context can be set with the minimumFret property.

```
<<
  \new ChordNames {
    \chordmode {
      d1:m d:m
    }
  }
  \new FretBoards {
    <d a d' f'>1
    \set FretBoards.minimumFret = 5
    <d a d' f'>1
  }
  \new Staff {
    \clef "treble_8"
    <d a d' f'>1
    <d a d' f'>1
  }
>>
```



The strings and frets for the `FretBoards` context depend on the `stringTunings` property, which has the same meaning as in the `TabStaff` context. See Section 12.1.4 [Custom tablatures], page 452, 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*; see Section "Fret diagrams explained and developed" in *Snippets* for an exhaustive example. For a `FretBoards` fret diagram, the interface properties belong to `FretBoards.FretBoard`.

Predefined commands

`\predefinedFretboardsOff`, `\predefinedFretboardsOn`.

See also

Notation Reference: Section 12.1.4 [Custom tablatures], page 452.

Snippets: Section "Fretted string instruments" in *Snippets*.

Internals Reference: Section "fret-diagram-interface" in *Internals Reference*.

Known issues and warnings

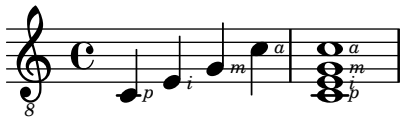
Automatic fretboard calculations do not work properly for instruments with non-monotonic tunings.

12.1.8 Right-hand fingerings

Right-hand fingerings *p-i-m-a* must be entered using `\rightHandFinger` followed by a number.

> **Note:** If the number is entered in Scheme notation, remember to append a space before following it with a closing > or similar.

```
\clef "treble_8"
c4\rightHandFinger 1
e\rightHandFinger 2
g\rightHandFinger 3
c'\rightHandFinger 4
<c\rightHandFinger 1 e\rightHandFinger 2
 g\rightHandFinger 3 c'\rightHandFinger 4 >1
```



For convenience, \rightHandFinger may be abbreviated to something shorter, for example \RH, by adding the appropriate definition at the source file's top level:

```
RH = \rightHandFinger \etc
```

Most behaviors of right-hand fingerings (namely, the StrokeFinger object) may be set in the same way as ordinary fingerings: see Section 7.1.2 [Fingering instructions], page 282.

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.

```
#(define RH rightHandFinger)

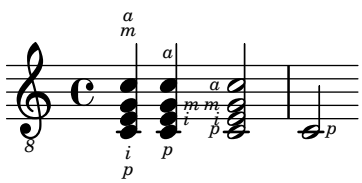
\relative c {
  \clef "treble_8"

  \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

  \set strokeFingerOrientations = #'(right)
  c\RH 1
}
```

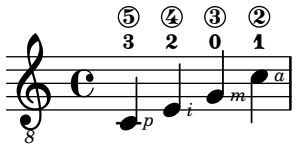


Fingerings, string indications, and right-hand fingerings

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

Notation Reference: Section 7.1.2 [Fingering instructions], page 282.

Snippets: Section “Fretted string instruments” in *Snippets*.

Internals Reference: Section “StrokeFinger” in *Internals Reference*.
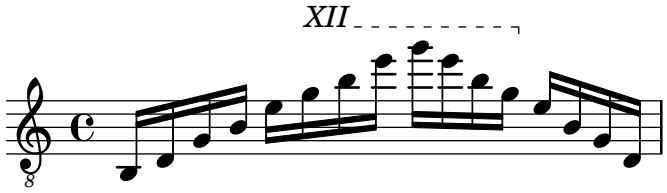
12.2 Guitar

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.

12.2.1 Indicating position and barring

This example demonstrates how to include guitar position and barring indications using a barré line.

```
\relative {
  \clef "treble_8"
  b,16 d g b e
  \once \override TextSpanner.bound-details.left.text =
    \markup {"XII" \hspace #0.4 }
  \once \override TextSpanner.bound-details.right.text =
    \markup \draw-line #'(0 . -.5)
  \once \override TextSpanner.bound-details.right.padding = -0.65
  \once \override TextSpanner.dash-fraction = 0.4
  \once \override TextSpanner.dash-period = 1.3
  g16\startTextSpan
  b16 e g e b g\stopTextSpan
  e16 b g d
}
```



A more sophisticated solution can be found in this guitar barré snippet (`https://wiki.lilypond.community/wiki/Guitar_barre`).

See also
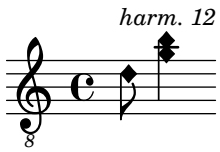
Notation Reference: Section 8.1.3 [Text spanners], page 312.

Snippets: Section "Fretted string instruments" in *Snippets*, Section "Expressive marks" in *Snippets*.

12.2.2 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 {
  \clef "treble_8"
  \override NoteHead.style = #'harmonic-mixed
  d'8^\markup { \italic \fontsize #-2 "harm. 12" } <g b>4
}
```



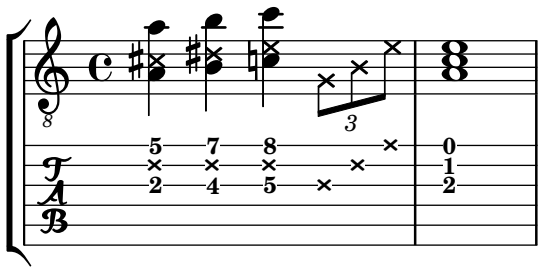
Dampened notes (also called *dead notes*) are supported within normal and tablature staves. In the following example, such notes are shown with pitches in the normal staff, indicating the frets where they are dampened. An alternative notation is to use empty strings instead. In tablature notation, however, it doesn't make a difference.

```
music = \relative c' {
  \omit StringNumber

  <a \deadNote cis a'>4
    <b\3 \deadNote dis b'>
    <c\3 \deadNote e\2 c'>
    \deadNotesOn
    \tuplet 3/2 { g8 b e }
    \deadNotesOff
  <a, c e>1
}

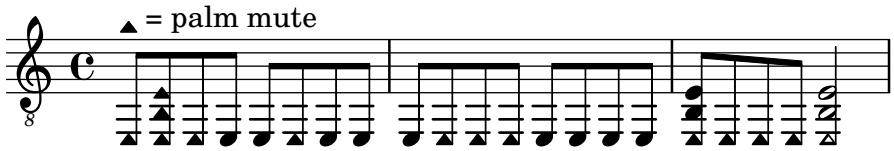
\new StaffGroup <<
  \new Staff {
    \clef "treble_8"
    \music
  }
  \new TabStaff {
    \music
```

```
  }
>>
```



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.

```
\relative c, {
  \clef "G_8"
  \palmMuteOn
    e8^\markup { \musicglyph "noteheads.s2do" = 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 string instruments” in *Snippets*.

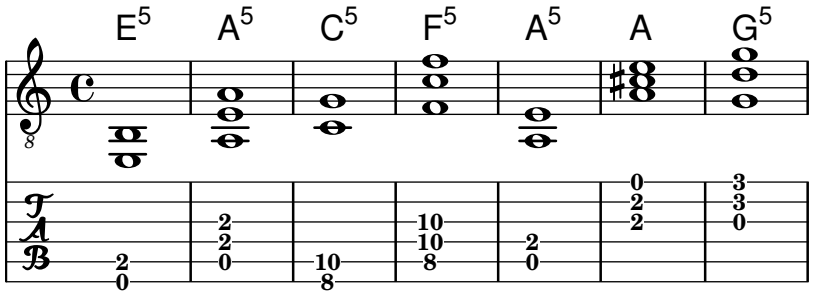
Notation Reference: Section 1.4.1 [Special note heads], page 43, Section B.9 [Note head styles], page 922.

12.2.3 Indicating power chords

Power chords and their symbols can be engraved in chord mode or as chord constructs. As an exception, the fifth is specified in these chord names, whereas it is usually left out in other chords (e.g., major or minor triads).

```
ChordsAndSymbols = {
  \chordmode {
    e,,1:5
    a,,:5.8
    \set TabStaff.restrainOpenStrings = ##t
    \set minimumFret = 8
    c,:5
    f,:5.8
  }
  \set minimumFret = 2
  \set restrainOpenStrings = ##f
  <a, e> <a cis' e'>
```

```
  <g d' g'>
}
\score {
  <<
    \new ChordNames {
    \ChordsAndSymbols
    }
    \new Staff {
      \clef "treble_8"
      \ChordsAndSymbols
    }
    \new TabStaff {
      \ChordsAndSymbols
    }
  >>
}
```



See also

Music Glossary: Section “power chord” in *Music Glossary*.

Notation Reference: Section 15.1.3 [Extended and altered chords], page 510, Section 15.2.1 [Printing chord names], page 513.

Snippets: Section “Fretted string instruments” in *Snippets*.

12.3 Banjo

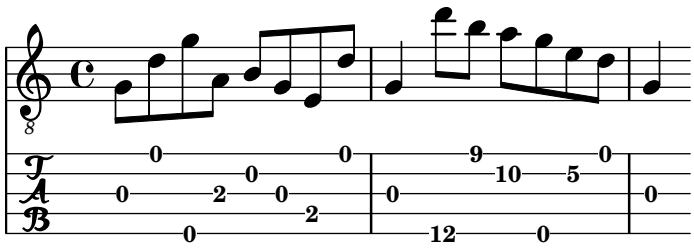
12.3.1 Banjo tablatures

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:

```
music = {
  g8 d' g'\5 a b g e d' |
  g4 d''8\5 b' a'\2 g'\5 e'\2 d' |
  g4
}

<<
  \new Staff \with { \omit StringNumber }
  { \clef "treble_8"  \music }
  \new TabStaff \with {
    tablatureFormat = #fret-number-tablature-format-banjo
    stringTunings = #banjo-open-g-tuning
  }
  { \music }
```

```
>>
```



A number of common tunings for the five-string banjo are predefined: `banjo-open-g-tuning` (gDGBD), `banjo-c-tuning` (gCGBD), `banjo-modal-tuning` (gDGCD), `banjo-open-d-tuning` (aDF#AD), `banjo-open-dm-tuning` (aDFAD), `banjo-double-c-tuning` (gCGCD) and `banjo-double-d-tuning` (aDGDE).

These may be converted to four-string tunings using the `four-string-banjo` function:

```
\set TabStaff.stringTunings = #(four-string-banjo banjo-c-tuning)
```

See also

Installed Files: `ly/string-tunings-init.ly`.

Snippets: Section "Fretted string instruments" in *Snippets*.

12.4 Lute

12.4.1 Lute tablatures

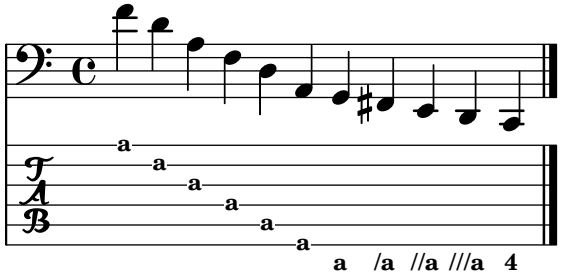
LilyPond supports tablature for lute.

To get additional bass strings use `additionalBassStrings`, where the pitches of those strings are set. They will be printed below lowest line as: a, /a, //a, ///a, 4, 5, etc.

`fret-letter-tablature-format` for `tablatureFormat` should be used, probably also `fretLabels` for further customizing.

```
m = { f'4 d' a f d a, g, fis, e, d, c,  \bar "|." }

\score {
  <<
    \new Staff { \clef bass \cadenzaOn  \m }
    \new TabStaff \m
  >>
  \layout {
    \context {
      \Score
      tablatureFormat = #fret-letter-tablature-format
    }
    \context {
      \TabStaff
      stringTunings = \stringTuning <a, d f a d' f'>
      additionalBassStrings = \stringTuning <c, d, e, fis, g,>
      fretLabels = #'("a" "b" "r" "d" "e" "f" "g" "h" "i" "k")
    }
  }
}
```



Known issues and warnings

Using `FretBoards` with `additionalBassStrings` is not supported and will yield unsatisfying results.

13 Percussion

13.1 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.

13.1.1 References for percussion

- Some percussion may be notated on a rhythmic staff; this is discussed in Section 2.3.8 [Showing melody rhythms], page 100, and Section 6.1.1 [Instantiating new staves], page 242.
- MIDI output is discussed in a separate section; please see Chapter 24 [Creating MIDI output], page 654.

See also

Notation Reference: Section 2.3.8 [Showing melody rhythms], page 100, Section 6.1.1 [Instantiating new staves], page 242, Chapter 24 [Creating MIDI output], page 654.

Snippets: Section "Percussion" in *Snippets*.

13.1.2 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:
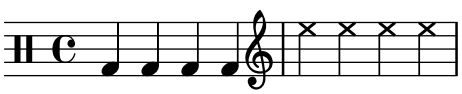
```
\new DrumStaff \drummode {
  hihat4 hh bassdrum bd
}
```



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 B.15 [Percussion notes], page 931.

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 percussion
  bd4 4 4 4
  \clef treble
  hh4 4 4 4
}
```



There are a few issues concerning MIDI support for percussion instruments; for details please see Chapter 24 [Creating MIDI output], page 654.

See also

Notation Reference: Chapter 24 [Creating MIDI output], page 654, Section B.15 [Percussion notes], page 931.
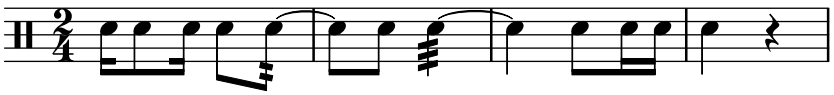
Installed Files: `ly/drumpitch-init.ly`.

Snippets: Section "Percussion" in *Snippets*.

13.1.3 Drum rolls

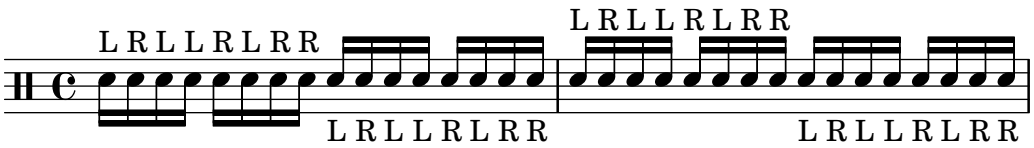
Drum rolls are indicated with three slashes across the stem. For quarter notes or longer the three slashes are shown explicitly, eighth notes are shown with two slashes (the beam being the third), and drum rolls shorter than eighths have one stem slash to supplement the beams. This is achieved with the tremolo notation, as described in Section 4.2.2 [Tremolo repeats], page 212.

```
\drums {
  \time 2/4
  sn16 8 16 8 8:32 ~
  8 8 4:32 ~
  4 8 16 16
  4 r4
}
```



Sticking can be indicated by placing a markup for `"R"` or `"L"` above or below notes, as discussed in Section 36.1 [Direction and placement], page 774. The `staff-padding` property may be overridden to achieve a pleasing baseline.

```
\drums {
  \repeat unfold 2 {
    sn16^"L" 16^"R" 16^"L" 16^"L" 16^"R" 16^"L" 16^"R" 16^"R"
    \stemUp
    sn16_"L" 16_"R" 16_"L" 16_"L" 16_"R" 16_"L" 16_"R" 16_"R"
  }
}
```



See also

Notation Reference: Section 4.2.2 [Tremolo repeats], page 212.

Snippets: Section "Percussion" in *Snippets*.

13.1.4 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

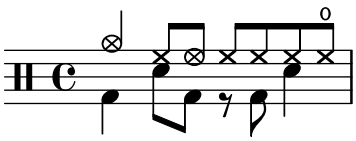
Notation Reference: Chapter 24 [Creating MIDI output], page 654.

Snippets: Section "Percussion" in *Snippets*.

13.1.5 Percussion staves

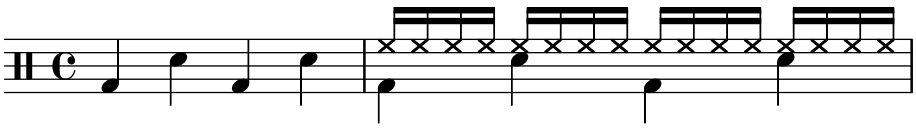
A percussion part for more than one instrument typically uses a multi-line staff where each position in the staff refers to one piece of percussion. To typeset the music, the notes must be interpreted in `DrumStaff` and `DrumVoice` context.

```
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,

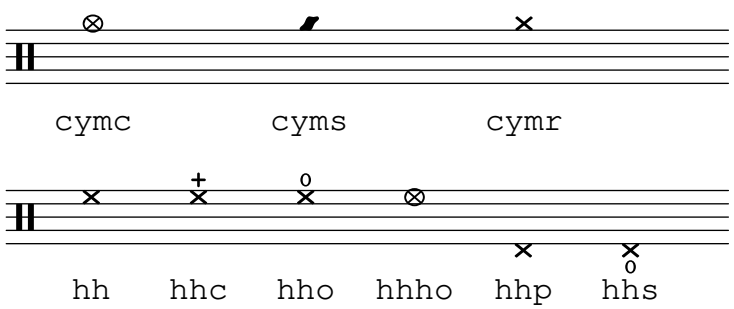
```
\new DrumStaff <<
  \drummode {
    bd4 sn4 bd4 sn4
    << {
      \*16 hh16
    } \\ {
      bd4 sn4 bd4 sn4
    } >>
  }
>>
```

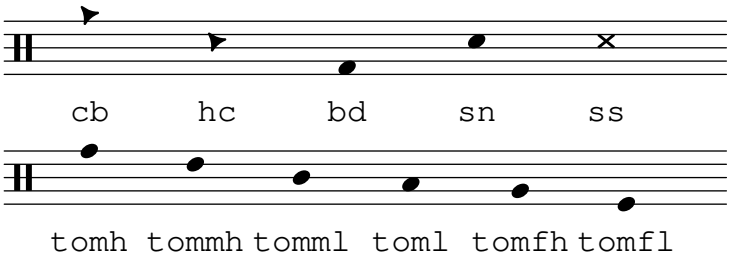


There are also other layout possibilities. To use these, set the property `drumStyleTable` in context `DrumVoice`. The following variables have been predefined:

`drums-style`

This is the default. It typesets a typical drum kit on a five-line staff:

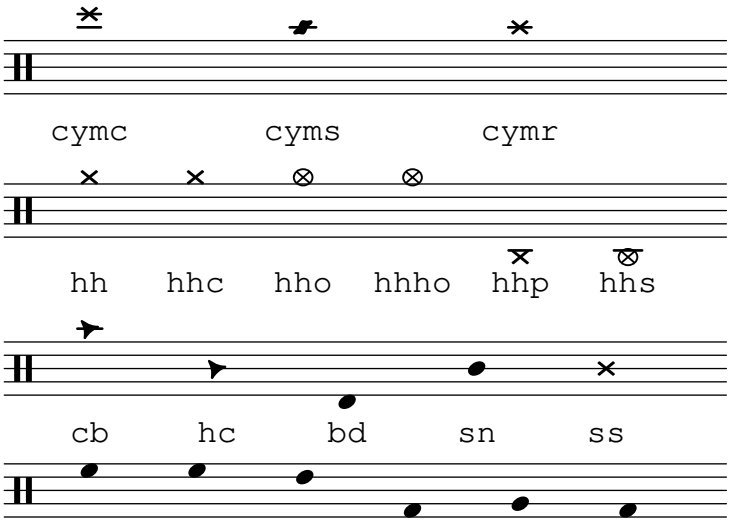




The drum scheme supports six different toms. When there are fewer toms, simply select the toms that produce the desired result. For example, to get toms on the three middle lines you use `tommh`, `tomml`, and `tomfh`.

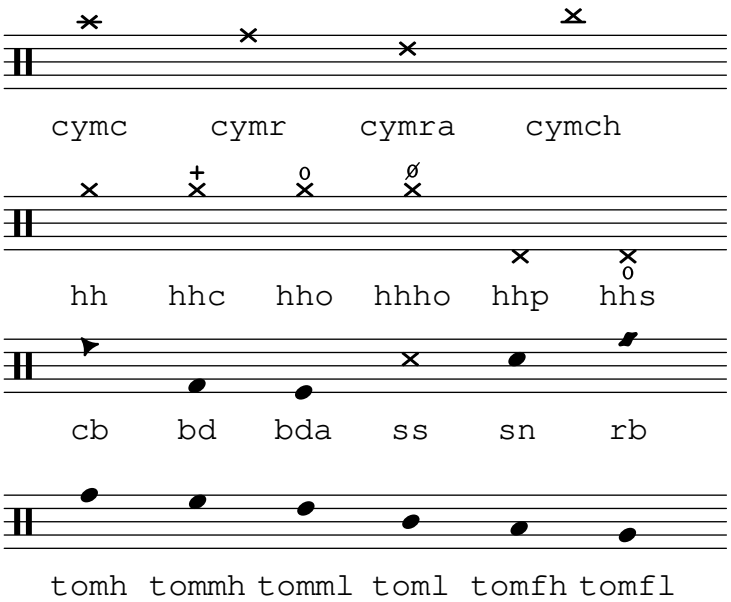
`agostini-drums-style`

Invented by the French percussionist Dante Agostini in 1965, this notation is commonly employed in France but also elsewhere.

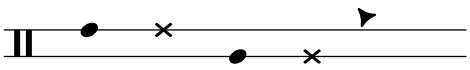


`weinberg-drums-style`

Based on the work of Norman Weinberg, published in his *Guidelines for Drumset Notation*.

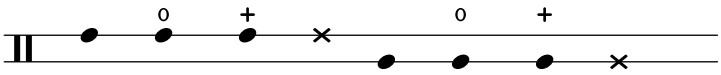


`timbales-style`
: This typesets timbales on a two line staff:



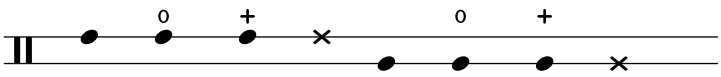
timh ssh timl ssl cb

`congas-style`
: This typesets congas on a two line staff:



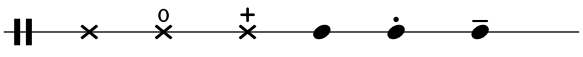
cgh cgho cghm ssh cgl cglo cglm ssl

`bongos-style`
: This typesets bongos on a two line staff:

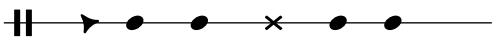


boh boho bohm ssh bol bolo bolm ssl

`percussion-style`
: To typeset all kinds of simple percussion on one-line staves:



tri trio trim gui guis guil



cb cl tamb cab mar hc

Custom percussion styles may also be defined, as explained in Section 13.1.6 [Custom percussion staves], page 491.

See also

Learning Manual: Section “I’m hearing voices” in *Learning Manual*.

Notation Reference: Section 13.1.6 [Custom percussion staves], page 491.

Installed Files: `ly/drumpitch-init.ly`.

Snippets: Section “Percussion” in *Snippets*.

13.1.6 Custom percussion staves

Custom percussion styles may be defined, to which the `drumStyleTable` property may then be set. Existing notations may be redefined as an association list where each entry has to be comprised of four items: a name, the note head style (or ‘`()`’ to indicate the default), an articulation sign if needed (or `#f` if not), and the note head’s position on the staff. That list must then be converted into a Scheme hash table, using the `alist->hash-table` function.

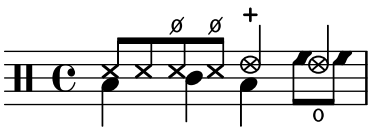
The entry for an articulation can either be a script name symbol or a pair, with the first element the script name symbol and the second a forced-direction indicator for the script.

[In the following example, note the use of the quasi-quotation shorthand (‘`` ` ``’) at the beginning of the Scheme expression instead of the standard quotation shorthand (‘`'`’), which enables the unquote shorthand (‘`,`’) to evaluate the element it precedes.]

```
#(define mydrums `(
         (bassdrum        ()        #f              -1)
         (snare           ()        #f              0)
         (hihat           cross     #f              1)
         (halfopenhihat   cross     halfopen        1)
         (pedalhihat      xcircle   stopped         2)
         (splashhihat     xcircle   (open . ,DOWN)   2)
         (lowtom          diamond   #f              3)))

up = \drummode { hh8 hh hhho hhho hhp4 hhs }
down = \drummode { bd4 sn bd toml8 toml }

\new DrumStaff \with { drumStyleTable = #(alist->hash-table mydrums) }
  <<
    \new DrumVoice { \voiceOne \up }
    \new DrumVoice { \voiceTwo \down }
  >>
```

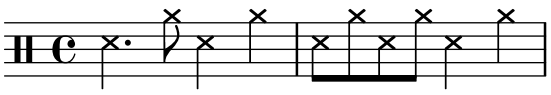


New names may also be added to these custom notations through the drumPitchNames variable, that may be redefined as an association list (or augmented by appending a new list to its existing value, as demonstrated below), but also through its individual entries. This also makes it possible to define aliases: alternate input shorthand for some notations.

```
drumPitchNames =
  #(append
    '((leftsnap . sidestick)
      (rightsnap . ridecymbal))
    drumPitchNames)

drumPitchNames.ls = #'sidestick
drumPitchNames.rs = #'ridecymbal

\drums {
  leftsnap4. rightsnap8 leftsnap4 rightsnap
  ls8 rs ls rs ls4 rs
}
```



In a similar manner, the drumPitchTable property associates a specific pitch (meaning a different instrument sound, as provided by available MIDI sound fonts) to each notation. That property needs to be defined as a hash table, which is again converted from an association list (stored by default as the midiDrumPitches variable). Redefining these associations is achieved as explained above, either by defining an entire association list or through individual entries. The following example demonstrates how to create a whole notation set with its own input syntax, custom notations and corresponding MIDI output.
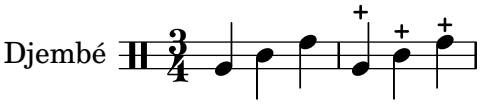
```
drumPitchNames.dbass     = #'dbass
drumPitchNames.dba       = #'dbass  % 'db is in use already
drumPitchNames.dbassmute = #'dbassmute
drumPitchNames.dbm       = #'dbassmute
drumPitchNames.do        = #'dopen
drumPitchNames.dopenmute = #'dopenmute
drumPitchNames.dom       = #'dopenmute
drumPitchNames.dslap     = #'dslap
drumPitchNames.ds        = #'dslap
drumPitchNames.dslapmute = #'dslapmute
drumPitchNames.dsm       = #'dslapmute

#(define djembe-style
  '((dbass       ()   #f        -2)
    (dbassmute   ()   stopped   -2)
    (dopen       ()   #f        0)
    (dopenmute   ()   stopped   0)
    (dslap       ()   #f        2)
    (dslapmute   ()   stopped   2)))

midiDrumPitches.dbass     = g
midiDrumPitches.dbassmute = fis
midiDrumPitches.dopen     = a
midiDrumPitches.dopenmute = gis
midiDrumPitches.dslap     = b
midiDrumPitches.dslapmute = ais

test = \drummode { dba4 do ds dbm dom dsm }

\score {
  \new DrumStaff \with {
    \override StaffSymbol.line-count = 3
    instrumentName = "Djembé "
    drumStyleTable = #(alist->hash-table djembe-style)
    drumPitchTable = #(alist->hash-table midiDrumPitches)
  } {
    \time 3/4
    \test
  }
  \layout {}
  \midi {}
}
```



See also

Installed Files: `ly/drumpitch-init.ly`.

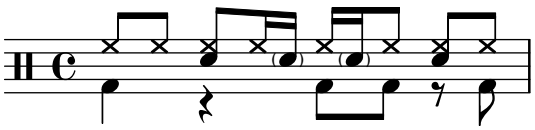
Snippets: Section “Percussion” in *Snippets*.

Internals Reference: Section “DrumStaff” in *Internals Reference*, Section “DrumVoice” in *Internals Reference*.

13.1.7 Ghost notes

Also known as dead, muted, silenced or false notes; ghost notes can be created using the \parenthesize command, see Section 7.1.8 [Parentheses], page 296.

```
\new DrumStaff <<
  \new DrumVoice  = "1" { s1 }
  \new DrumVoice  = "2" { s1 }
  \drummode {
    <<
      {
        hh8[ 8] <hh sn> hh16
        \parenthesize sn hh
        \parenthesize sn hh8 <hh sn> hh
      } \\
      {
        bd4 r4 bd8 8 r8 bd
      }
    >>
  }
>>
```

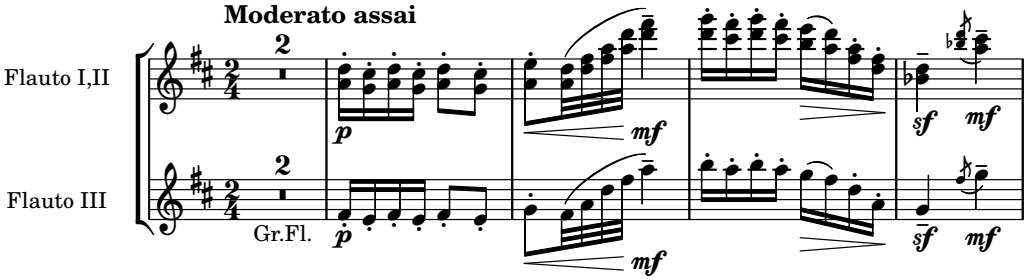


See also

Notation Reference: Section 7.1.8 [Parentheses], page 296.

Snippets: Section “Percussion” in *Snippets*.

14 Wind instruments



This section includes elements of music notation that arise when writing specifically for wind instruments.

14.1 Common notation for wind instruments

This section discusses notation common to most wind instruments.

14.1.1 References for wind instruments

Many notation issues for wind instruments pertain to breathing and tonguing:

- Breathing can be specified by rests or Section 3.2.3 [Breath marks], page 176.
- Legato playing is indicated by Section 3.2.1 [Slurs], page 171.
- Different types of tonguings, ranging from legato to non-legato to staccato are usually shown by articulation marks, sometimes combined with slurs, see Section 3.1.1 [Articulations and ornamentations], page 156, and Section B.13 [List of articulations], page 928.
- Flutter tonguing is usually indicated by placing a tremolo mark and a text markup on the note. See Section 4.2.2 [Tremolo repeats], page 212.

Other aspects of musical notation that can apply to wind instruments:

- Many wind instruments are transposing instruments, see Section 1.3.4 [Instrument transpositions], page 30.
- Slide glissandi are characteristic of the trombone, but other winds may perform keyed or valved glissandi. See Section 3.3.1 [Glissando], page 177.
- Harmonic series glissandi, which are possible on all brass instruments but common for French Horns, are usually written out as Section 2.6.1 [Grace notes], page 148.
- Pitch inflections at the end of a note are discussed in Section 3.2.4 [Falls and doits], page 177.
- Key slaps or valve slaps are often shown by the `cross` style of Section 1.4.1 [Special note heads], page 43.
- Woodwinds can overblow low notes to sound harmonics. These are shown by the `flageolet` articulation. See Section B.13 [List of articulations], page 928.
- 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 Section 3.1.1 [Articulations and ornamentations], page 156, and Section B.13 [List of articulations], page 928.
- Stopped horns are indicated by the `stopped` articulation. See Section 3.1.1 [Articulations and ornamentations], page 156.

See also

Notation Reference: Section 3.2.3 [Breath marks], page 176, Section 3.2.1 [Slurs], page 171, Section 3.1.1 [Articulations and ornamentations], page 156, Section B.13 [List of articulations], page 928, Section 4.2.2 [Tremolo repeats], page 212, Section 1.3.4 [Instrument transpositions], page 30, Section 3.3.1 [Glissando], page 177, Section 2.6.1 [Grace notes], page 148, Section 3.2.4 [Falls and doits], page 177, Section 1.4.1 [Special note heads], page 43.

Snippets: Section "Wind instruments" in *Snippets*.

14.1.2 Fingerings

All wind instruments other than the trombone require the use of several fingers to produce each pitch. Some fingering examples are shown in the snippets below.

Woodwind diagrams can be produced and are described in Section 14.3.1 [Woodwind diagrams], page 500.

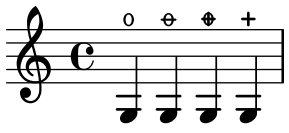
Selected snippets

Fingering symbols for wind instruments

Special symbols can be achieved by combining existing glyphs, which is useful for wind instruments.

```
lineup =
  \tweak outside-staff-padding #0
  \tweak staff-padding #0
  \tweak padding #0.2
  \tweak parent-alignment-X #CENTER
  \tweak self-alignment-X #CENTER
  \etc

\relative c' {
  g\open
  g\lineup ^\markup \combine
              \musicglyph "scripts.open"
              \musicglyph "scripts.tenuto"
  g\lineup ^\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 = #(lambda (g)
    (+ (ly:self-alignment-interface::centered-on-x-parent g)
```

```
        (ly:self-alignment-interface::x-aligned-on-self g)))
}

\new Staff \with {
  \remove "Time_signature_engraver"
  \omit Stem
  \omit Flag
  \consists "Horizontal_bracket_engraver"
} {
  \clef bass
  \set Score.timing = ##f

  f,1*1/4 \glissando

  \clef violin
  gis'1*1/4

  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

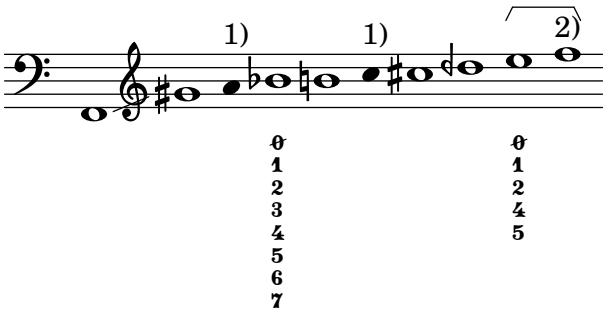
  c''4^\markup "1)"

  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

Notation Reference: Section 14.3.1 [Woodwind diagrams], page 500.

Snippets: Section "Wind instruments" in *Snippets*.

14.2 Bagpipes

This section discusses notation common bagpipes.

14.2.1 Bagpipe definitions

LilyPond contains special definitions for Scottish, Highland Bagpipe music; to use them, add
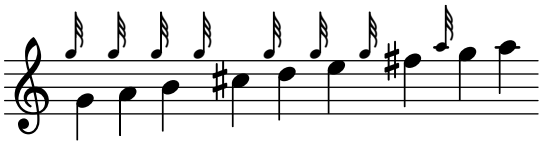
```
\include "bagpipe.ly"
```

to 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 `c-flat` or `f-flat`. Similarly, the *Piobaireachd* high g can be written `g-flat` when it occurs in light music.

See also

Snippets: Section "Wind instruments" in *Snippets*.

14.2.2 Bagpipe example

This is what the well known tune Amazing Grace looks like in bagpipe notation.
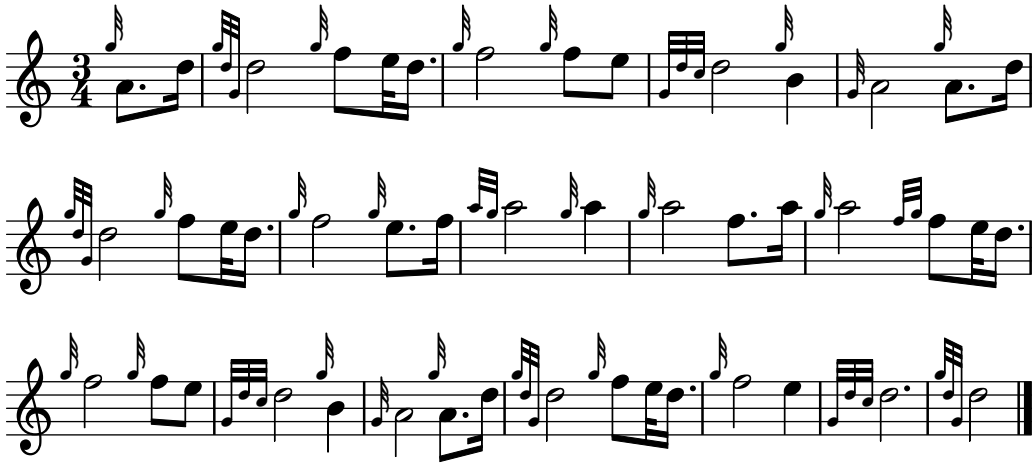
```
\include "bagpipe.ly"
\layout {
  indent = 0.0\cm
  \context { \Score \remove Bar_number_engraver }
}

\header {
  title = "Amazing Grace"
  meter = "Hymn"
  arranger = "Trad. arr."
}

{
  \hideKeySignature
  \time 3/4
  \grg \partial 4 a8. d16
  \slurd d2 \grg f8[ e32 d16.]
  \grg f2 \grg f8 e
  \thrwd d2 \grg b4
  \grG a2 \grg a8. d16
  \slurd d2 \grg f8[ e32 d16.]
  \grg f2 \grg e8. f16
  \dblA A2 \grg A4
  \grg A2 f8. A16
  \grg A2 \hdblf f8[ e32 d16.]
  \grg f2 \grg f8 e
  \thrwd d2 \grg b4
  \grG a2 \grg a8. d16
  \slurd d2 \grg f8[ e32 d16.]
  \grg f2 e4
  \thrwd d2.
  \slurd d2
  \bar "|."
}
```

Amazing Grace

Hymn Trad. arr.



See also

Snippets: Section “Wind instruments” in *Snippets*.

14.3 Woodwinds

This section discusses notation specifically for woodwind instruments.

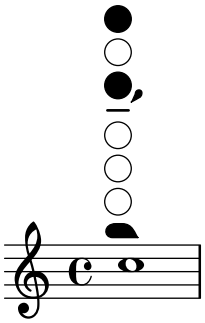
14.3.1 Woodwind diagrams

Woodwind diagrams can be used to indicate the fingering to be used for specific notes and are available for the following instruments:

- piccolo
- flute
- oboe
- clarinet
- bass clarinet
- low bass clarinet (bass clarinet to low C)
- saxophone
- bassoon
- contrabassoon

Woodwind diagrams are created as markups:

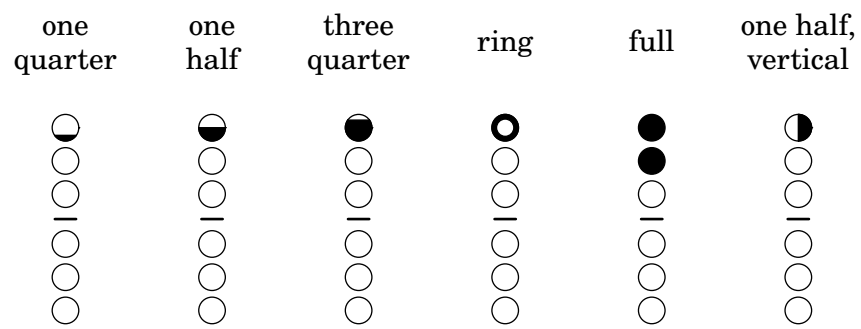
```
c''1^\markup {
  \woodwind-diagram #'piccolo #'((lh . (gis))
                                 (cc . (one three))
                                 (rh . (ees)))
}
```



Keys can be open, partially-covered, ring-depressed, or fully covered. The angle of partially-covered keys can be specified:

```
\markup \override #'(baseline-skip . 22) \column {
  \override #'(baseline-skip . 2.5) \fill-line {
    ""
    \raise #1.2 \center-column { "one" "quarter" }
    \raise #1.2 \center-column { "one" "half" }
    \raise #1.2 \center-column { "three" "quarter" }
    "ring"
    "full"
    \raise #1.2 \center-column { "one half," "vertical" }
    ""
  }
```

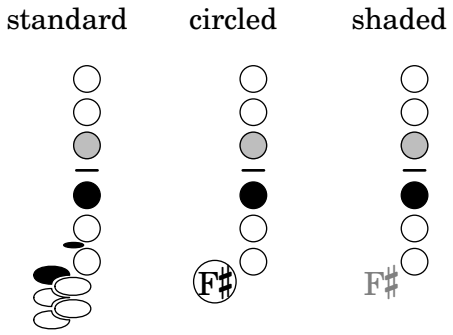
```
\fill-line {
  ""
  \woodwind-diagram #'flute #'((cc . (one1q))
                               (lh . ())
                               (rh . ()))
  \woodwind-diagram #'flute #'((cc . (one1h))
                               (lh . ())
                               (rh . ()))
  \woodwind-diagram #'flute #'((cc . (one3q))
                               (lh . ())
                               (rh . ()))
  \woodwind-diagram #'flute #'((cc . (oneR))
                               (lh . ())
                               (rh . ()))
  \woodwind-diagram #'flute #'((cc . (oneF two))
                               (lh . ())
                               (rh . ()))
  \override #'(woodwind-diagram-details . ((fill-angle . 90)))
  \woodwind-diagram #'flute #'((cc . (one1h))
                               (lh . ())
                               (rh . ()))
  ""
}
}
```



Trills are indicated as shaded keys, or in non-graphical mode, as either circled (the default) or shaded text:

```
\markup {
  \override #'(baseline-skip . 20)
  \override #'(line-width . 50)
  \column {
    \fill-line {
      ""
      "standard"
      "circled"
      "shaded"
      ""
    }
    \fill-line {
      ""
      \woodwind-diagram #'bass-clarinet
                        #'((cc . (threeT four))
```

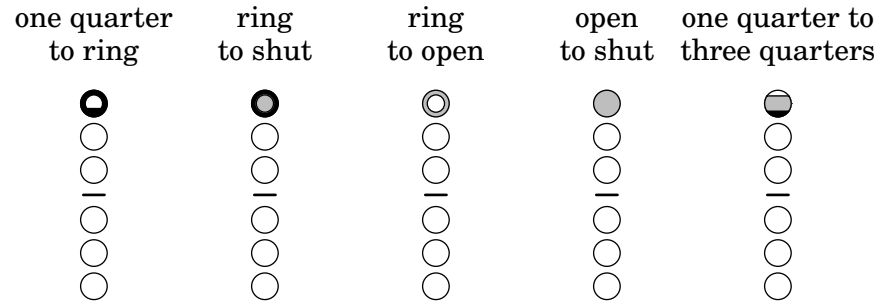
```
                        (lh . ())
                        (rh . (b fis)))
      \override #'(graphical . #f)
      \woodwind-diagram #'bass-clarinet
                        #'((cc . (threeT four))
                           (lh . ())
                           (rh . (b fisT)))
      \override #'(graphical . #f)
      \override #'(woodwind-diagram-details
                   . ((text-trill-circled . #f)))
      \woodwind-diagram #'bass-clarinet
                        #'((cc . (threeT four))
                           (lh . ())
                           (rh . (b fisT)))
      ""
    }
  }
}
```



A variety of trills can be displayed:

```
\markup {
  \override #'(baseline-skip . 22)
  \column {
    \override #'(baseline-skip . 2.5) \fill-line {
      ""
      \center-column { "one quarter" "to ring" }
      \center-column { "ring" "to shut" }
      \center-column { "ring" "to open" }
      \center-column { "open" "to shut" }
      \center-column { "one quarter to" "three quarters" }
      ""
    }
    \fill-line {
      ""
      \woodwind-diagram #'flute #'((cc . (one1qTR))
                                   (lh . ())
                                   (rh . ()))
      \woodwind-diagram #'flute #'((cc . (oneTR))
                                   (lh . ())
                                   (rh . ()))
      \woodwind-diagram #'flute #'((cc . (oneRT))
                                   (lh . ())
```

```
                                  (rh . ()))
      \woodwind-diagram #'flute #'((cc . (oneT))
                                  (lh . ())
                                  (rh . ()))
      \woodwind-diagram #'flute #'((cc . (one1qT3q))
                                  (lh . ())
                                  (rh . ()))
      ""
    }
  }
}
```



The list of all possible keys and settings for a given instrument can be displayed on the console using `#(print-keys-verbose 'flute)` or in the log file using `#(print-keys-verbose 'flute (current-error-port))`, although they will not show up in the music output.

Creating new diagrams is possible, although this will require Scheme ability and may not be accessible to all users. The patterns for the diagrams are in files `scm/define-woodwind-diagrams.scm` and `scm/display-woodwind-diagrams.scm`.

Selected snippets

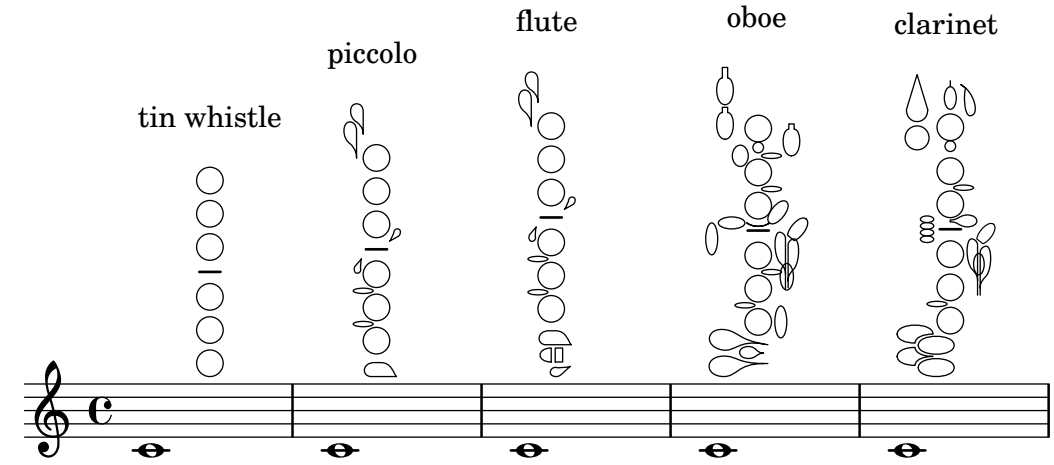
Woodwind diagrams listing

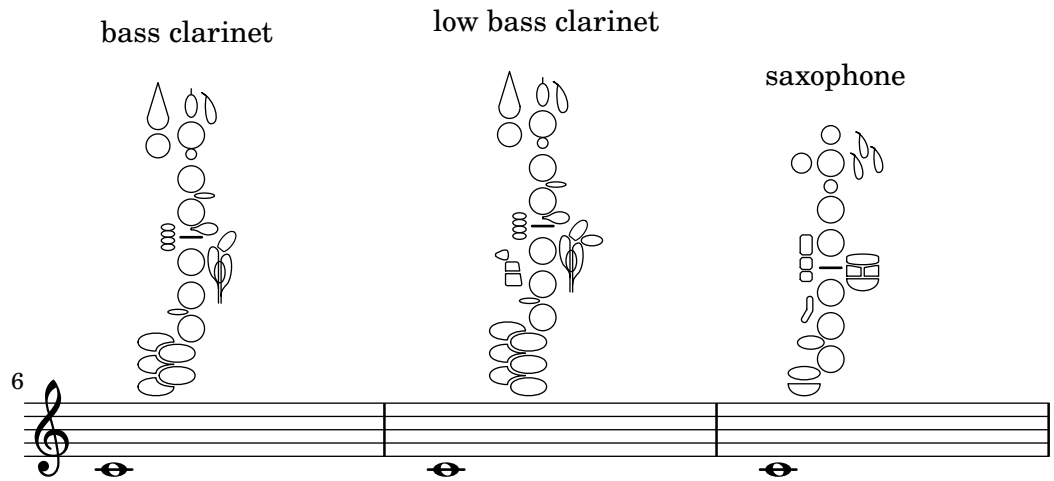
The following music shows all of the woodwind diagrams currently defined in LilyPond.

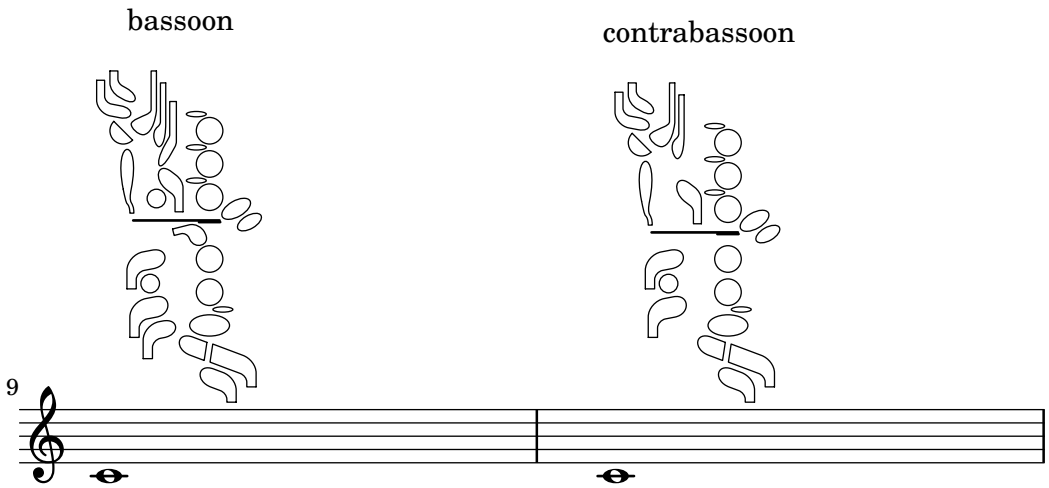
```
\relative c' {
  \textLengthOn
  c1^\markup \center-column { "tin whistle"
                              " "
                              \woodwind-diagram #'tin-whistle #'() }
  c1^\markup \center-column { "piccolo"
                              " "
                              \woodwind-diagram #'piccolo #'() }
  c1^\markup \center-column { "flute"
                              " "
                              \woodwind-diagram #'flute #'() }
  c1^\markup \center-column { "oboe"
                              " "
                              \woodwind-diagram #'oboe #'() }
  c1^\markup \center-column { "clarinet"
                              " "
                              \woodwind-diagram #'clarinet #'() }
  \break
```

```
  c1^\markup \center-column { "bass clarinet"
                              " "
                              \woodwind-diagram #'bass-clarinet #'() }
  c1^\markup \center-column { "low bass clarinet"
                              " "
                              \woodwind-diagram #'low-bass-clarinet #'() }
  c1^\markup \center-column { "saxophone"
                              " "
                              \woodwind-diagram #'saxophone #'() }
  c1^\markup \center-column { "bassoon"
                              " "
                              \woodwind-diagram #'bassoon #'() }
  c1^\markup \center-column { "contrabassoon"
                              " "
                              \woodwind-diagram #'contrabassoon #'() }
}

\paper {
  system-system-spacing.padding = 5
}
```





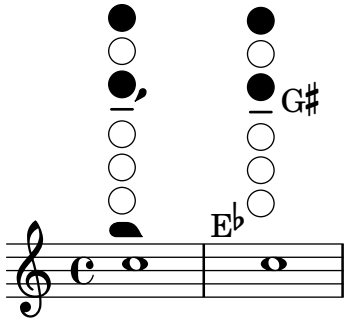


Graphical and text woodwind diagrams

In many cases, the keys other than the central column can be displayed by key name as well as by graphical means.

```
\relative c'' {
  \textLengthOn
  c1^\markup
    \woodwind-diagram #'piccolo
                      #'((cc . (one three))
                         (lh . (gis))
                         (rh . (ees)))

  c^\markup
    \override #'(graphical . #f)
      \woodwind-diagram #'piccolo
                        #'((cc . (one three))
                           (lh . (gis))
                           (rh . (ees)))
}
```



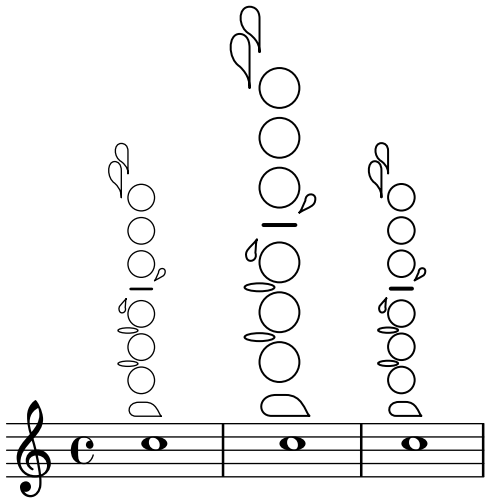
Changing the size of woodwind diagrams

The size and thickness of woodwind diagrams can be changed.

```
\relative c'' {
  \textLengthOn
  c1^\markup
       \woodwind-diagram #'piccolo #'()

  c^\markup \override #'(size . 1.5)
```

```
        \woodwind-diagram #'piccolo #'()
    c^\markup \override #'(thickness . 0.15)
        \woodwind-diagram #'piccolo #'()
}
```
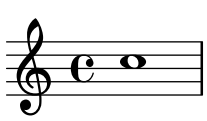


Woodwind diagrams key lists

The snippet below produces a list of all possible keys and key settings for woodwind diagrams as defined in `scm/define-woodwind-diagrams.scm`. The list gets written to stderr but is not shown in the music. If output to stdout is wanted instead, omit the code `(current-error-port)` from the commands.

```
#(print-keys-verbose 'piccolo (current-error-port))
#(print-keys-verbose 'flute (current-error-port))
#(print-keys-verbose 'flute-b-extension (current-error-port))
#(print-keys-verbose 'tin-whistle (current-error-port))
#(print-keys-verbose 'oboe (current-error-port))
#(print-keys-verbose 'clarinet (current-error-port))
#(print-keys-verbose 'bass-clarinet (current-error-port))
#(print-keys-verbose 'low-bass-clarinet (current-error-port))
#(print-keys-verbose 'saxophone (current-error-port))
#(print-keys-verbose 'soprano-saxophone (current-error-port))
#(print-keys-verbose 'alto-saxophone (current-error-port))
#(print-keys-verbose 'tenor-saxophone (current-error-port))
#(print-keys-verbose 'baritone-saxophone (current-error-port))
#(print-keys-verbose 'bassoon (current-error-port))
#(print-keys-verbose 'contrabassoon (current-error-port))

\score {c''1}
```



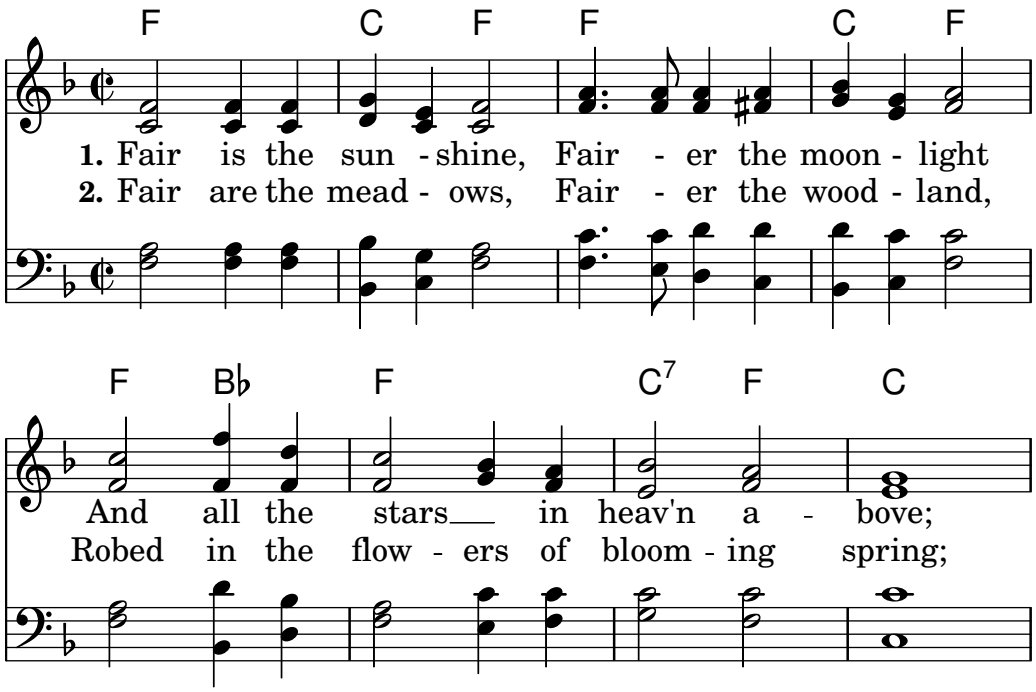
See also

Installed Files: `scm/define-woodwind-diagrams.scm`, `scm/display-woodwind-diagrams.scm`.

Snippets: Section “Wind instruments” in *Snippets*.

Internals Reference: Section “TextScript” in *Internals Reference*, Section “instrument-specific-markup-interface” in *Internals Reference*.

15 Chord notation



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.

15.1 Chord mode

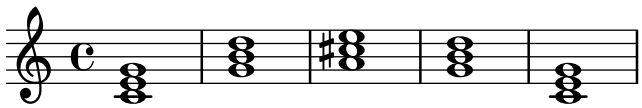
Chord mode is used to enter chords using an indicator of the chord structure, rather than the chord pitches.

15.1.1 Chord mode overview

Chords can be entered as simultaneous music, as discussed in Section 5.1.1 [Chorded notes], page 216.

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 Chapter 19 [Input modes], page 589.

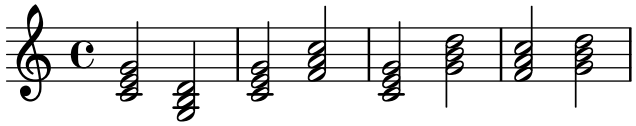
```
\chordmode { c1 g a g c }
```



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:

```
\relative {
  <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: Section 5.1.1 [Chorded notes], page 216, Chapter 19 [Input modes], page 589.

Snippets: Section “Chord notation” in *Snippets*.

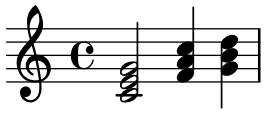
Known issues and warnings

Predefined shorthands for articulations and ornaments cannot be used on notes in chord mode, see Section 3.1.1 [Articulations and ornamentations], page 156.

15.1.2 Common chords
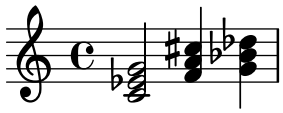
Major triads are entered by including the root and an optional duration:

```
\chordmode { c2 f4 g }
```



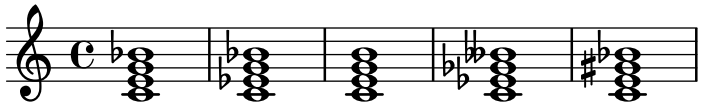
Minor, augmented, and diminished triads are entered by placing : and a quality modifier string after the duration:

```
\chordmode { c2:m f4:aug g:dim }
```



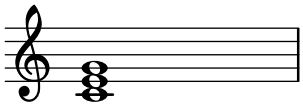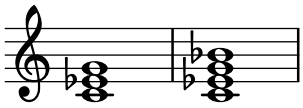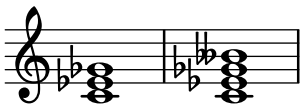
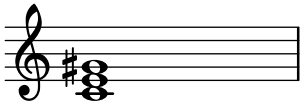
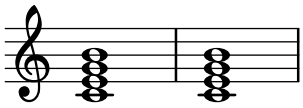
Seventh chords can be created:

```
\chordmode { c1:7 c:m7 c:maj7 c:dim7 c:aug7 }
```



The table below 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 B.2 [Common chord modifiers], page 890.

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 B.2 [Common chord modifiers], page 890, Section 15.1.3 [Extended and altered chords], page 510.

Snippets: Section “Chord notation” in *Snippets*.

Known issues and warnings

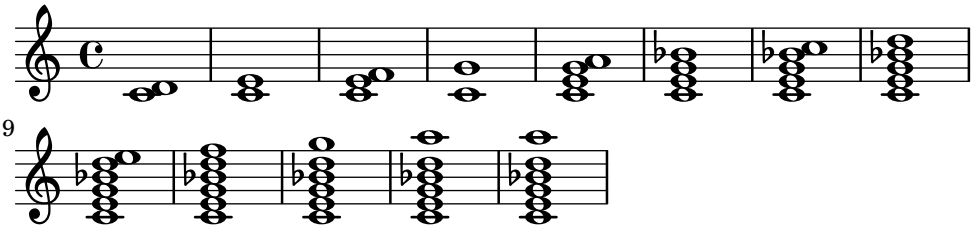
Only one quality modifier should be used per chord, typically on the highest step present in the chord. Chords with more than quality modifier will be parsed without an error or warning, but the results are unpredictable. Chords that cannot be achieved with a single quality modifier should be altered by individual pitches, as described in Section 15.1.3 [Extended and altered chords], page 510.

15.1.3 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.

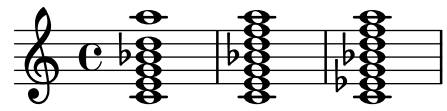
```
\chordmode {
  c1:2 c:3 c:4 c:5
  c1:6 c:7 c:8 c:9
  c1:10 c:11 c:12 c:13
  c1:14
}
```



As a special exception, `c:5` produces a 'power chord' only consisting of root and fifth.
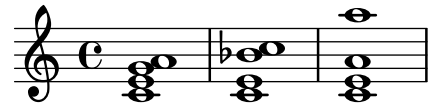
Since an unaltered 11 does not sound good when combined with an unaltered 13, the 11 is removed from a `:13` major chord (unless it is added explicitly).

```
\chordmode {
  c1:13 c:13.11 c:m13
}
```



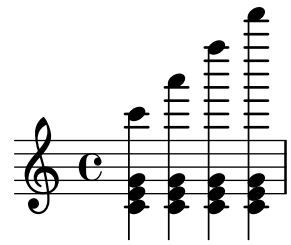
Individual steps can be added to a chord. Additions follow the extent and are prefixed by a dot (`.`). The basic seventh step added to a chord is the minor or flatted seventh, rather than the major seventh.

```
\chordmode {
  c1:3.5.6 c:3.7.8 c:3.6.13
}
```
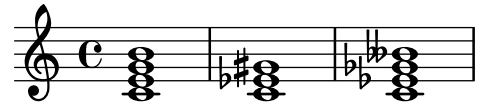


Added steps can be as high as desired.

```
\chordmode {
  c4:3.5.15 c:3.5.20 c:3.5.25 c:3.5.30
}
```



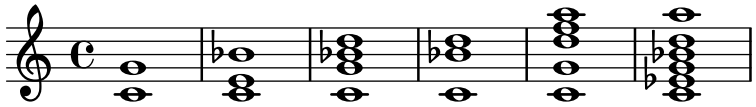
Added chord steps can be altered by suffixing a `-` or `+` sign to the number. To alter a step that is automatically included as part of the basic chord structure, add it as an altered step.

```
\chordmode {
  c1:7+ c:5+.3- c:3-.5-.7-
}
```



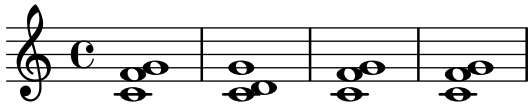
Following any steps to be added, a series of steps to be removed is introduced in a modifier string with a prefix of `^`. If more than one step is to be removed, the steps to be removed are separated by `.` following the initial `^`.

```
\chordmode {
  c1^3 c:7^5 c:9^3 c:9^3.5 c:13.11^3.7 c:m13^11
}
```



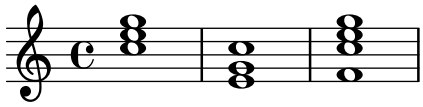
The modifier sus can be added to the modifier string to create suspended chords. This removes the 3rd step from the chord. Append either 2 or 4 to add the 2nd or 4th step to the chord. When sus is followed by either a 2nd or 4th step, it is equivalent to ^3, otherwise to sus4, namely 5.4.

```
\chordmode {
  c1:sus c:sus2 c:sus4 c:5.4
}
```



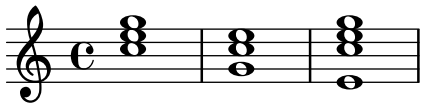
Added bass notes (putting a pitch other than the root on the bottom of the chord) can be specified by appending */pitch* to the chord.

```
\chordmode {
  c'1 c'/e c'/f
}
```



If the added pitch is already part of the chord, this may be used to print chord inversions, in which case the pitch is not added but merely moved to the bottom of the chord. It may however be treated as an added note (and thus printed twice), by using the syntax */+pitch*.

```
\chordmode {
  c'1 c'/g c'/+e
}
```



Automatic chord inversions and voicings are demonstrated in Section 15.1.4 [Chord inversions and specific voicings], page 513.

Chord modifiers that can be used to produce a variety of standard chords are shown in Section B.2 [Common chord modifiers], page 890.

See also
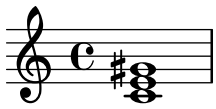
Notation Reference: Section 15.1.4 [Chord inversions and specific voicings], page 513, Section B.2 [Common chord modifiers], page 890.

Snippets: Section “Chord notation” in *Snippets*.

Known issues and warnings

Later entries for a given step override earlier entries. The following simply produces the augmented chord, since 5+ is interpreted last.

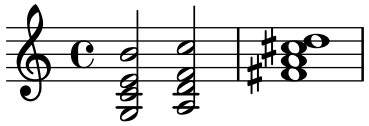
```
\chordmode { c1:3.5.5-.5+ }
```



15.1.4 Chord inversions and specific voicings

In addition to chord modifiers and added bass notes, various functions may be used to automatically print chords in a specific inversion or voicing – for example the so-called 'drop 2' voicing commonly used in jazz music.

```
\chordmode {
  \dropNote 2 {
    c2:maj7 d:m7
  }
  \invertChords 1 d1:maj7
}
```



Unlike added bass notes shown in Section 15.1.3 [Extended and altered chords], page 510, this only affects the way chords are printed on a staff, and not chord names written with letters. Furthermore, these functions may be used not only in chord mode but also with <...> chords constructs explained in Section 5.1.1 [Chorded notes], page 216.

See also

Notation Reference: Section 15.1.3 [Extended and altered chords], page 510, Section 5.1.1 [Chorded notes], page 216.

Snippets: Section "Chord notation" in *Snippets*.

15.2 Displaying chords

Chords can be displayed by name, in addition to the standard display as notes on a staff.

15.2.1 Printing chord names

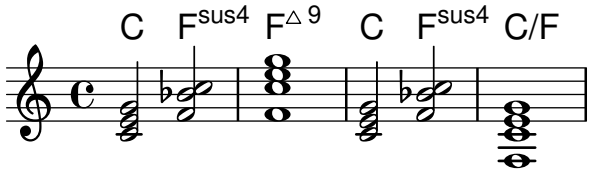
Chord names are printed in the ChordNames context:

```
\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:

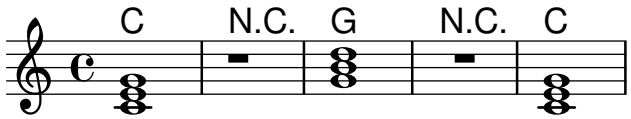
```
chordmusic = \relative {
  <c' e g>2 <f bes c>
  <f c' e g>1
  \chordmode {
    c2 f:sus4 c1:/f
  }
}
<<
  \new ChordNames {
    \chordmusic
  }
  {
    \chordmusic
  }
>>
```



When passed to a ChordNames context, rests (including multi-measure rests) cause the text "N.C." (*No Chord*) to be displayed.

```
myChords = \chordmode {
  c1
  r1
  g1
  R1
  c1
}

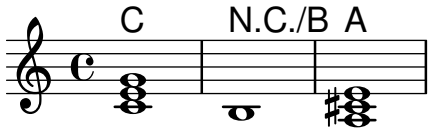
<<
  \new ChordNames \myChords
  \new Staff \myChords
>>
```



It is also possible to display "N.C." with a bass note by forcing the chord to be a single note identical to the bass.

```
myChords = \chordmode {
  c1
  b1:1/b
  a,1
}

<<
  \new ChordNames \myChords
  \new Staff \myChords
>>
```



`\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△

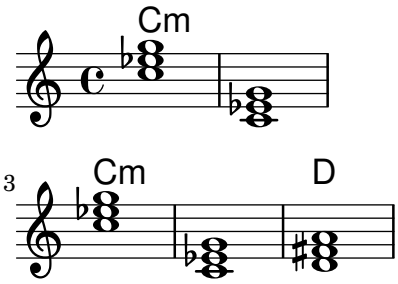
Selected snippets

Showing chords at changes

By default, every chord entered is printed. This behavior can be modified so that chord names are printed only at the start of lines or when the chord changes.

```
harmonies = \chordmode {
  c'1:m c:m \break
  c'1:m c:m d
}

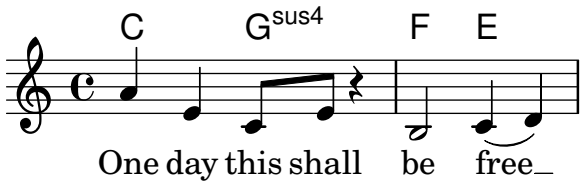
<<
  \new ChordNames {
    \set chordChanges = ##t
    \harmonies
  }
  \new Staff {
    \harmonies
  }
>>
```



Simple lead sheet

When put together, chord names, a melody, and lyrics form a lead sheet.

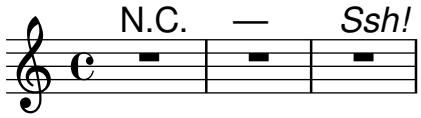
```
<<
  \chords { c2 g:sus4 f e }
  \new Staff \relative c'' {
    a4 e c8 e r4
    b2 c4( d)
  }
  \addlyrics { One day this shall be free __ }
>>
```



Customizing the no-chord symbol

By default, rests in a `ChordNames` context cause the text "N.C." to be printed. This markup can be customized by setting the `noChordSymbol` context property.

```
<<
  \chords {
    R1
    \set noChordSymbol = "---"
    R1
    \set noChordSymbol = \markup \italic "Ssh!"
    R1
  }
  {
    R1*3
  }
>>
```



See also

Music Glossary: Section "chord" in *Music Glossary*.

Notation Reference: Section 5.2.6 [Writing music in parallel], page 238.

Snippets: Section "Chord notation" 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.

15.2.2 Customizing chord names

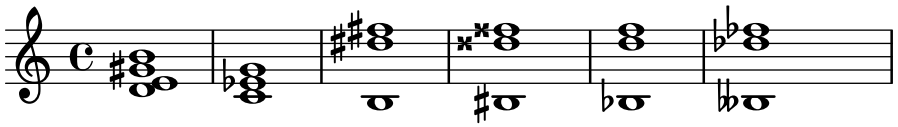
There is no unique system for naming chords. Different musical traditions use different names for the same set of chords. There are also different symbols displayed for a given chord name. The names and symbols displayed for chord names are customizable.

The basic chord name layout is a system for Jazz music proposed by Klaus Ignatzek (see Section "Literature list" in *Essay*). (Other chord naming systems may be implemented through Scheme functions, see Section "Chord names alternative" in *Snippets* for an example.) A list of common jazz chords notations can be found in Section B.1 [Chord name chart], page 890.

The default naming system may be tweaked easily in a number of ways. To begin with, predefined commands allow the display of different languages for the root pitch. These include `\germanChords`, `\italianChords`, `\frenchChords`, and `\englishChords`.

A special case is `\semiGermanChords`, which is intended for educational purposes in German-speaking countries while addressing an international audience. It displays letter B with a flat for German pitch B; everything else is identical to `\germanChords`.

default	E/D	Cm	B/B	B♯/B♯	B♭/B♭	B𝄫/B𝄫
German	E/d	Cm	H/h	H♯/his	B/b	H𝄫/heses
semi-German	E/d	Cm	H/h	H♯/his	B♭/b	H𝄫/heses
Italian	Mi/Re	Do m	Si/Si	Si♯/Si♯	Si♭/Si♭	Si𝄫/Si𝄫
French	Mi/Ré	Do m	Si/Si	Si♯/Si♯	Si♭/Si♭	Si𝄫/Si𝄫
English	E/D	Cm	B/B	B♯/B♯	B♭/B♭	B𝄫/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
}
```



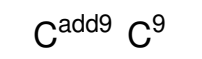
The chord name display can also be tuned by adjusting the following properties.

`chordRootNamer`
: The default setup is to print the chord name 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 different function in this property.

`majorSevenSymbol`
: This property holds the markup object that identifies a major 7 chord, to be printed right after the chord root. Predefined values are `whiteTriangleMarkup` (the default) and `blackTriangleMarkup`.

`additionalPitchPrefix`
: If the chord name contains additional pitches, they normally have the prefix 'add', which is LilyPond's default. This prefix can be changed or suppressed.

```
\new ChordNames {
  <c e g d'>    % add9
  \set additionalPitchPrefix = ""
  <c e g d'>    % add9
}
```



chordNoteNamer

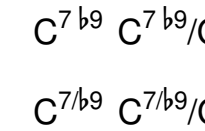
If a chord name contains additional pitches other than the root (e.g., an added bass note), the function assigned to this property is used to print them. If not set or set to `'()`, use the function assigned to `chordRootNamer` instead.

As an example, the function `chord-name:german-lowercase-name-markup` is used by `\germanChords` to print bass notes in lower case.

chordNameSeparator

Different parts of a chord name are normally separated by a small amount of horizontal space. By setting `chordNameSeparator`, you can use any desired markup as a separator. This does not affect the separator between a chord and its bass note; to customize that, use `slashChordSeparator`.

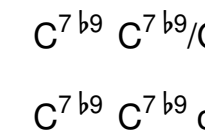
```
\chords {
  c4:7.9- c:7.9-/g
  \set chordNameSeparator = \markup { "/" }
  \break
  c4:7.9- c:7.9-/g
}
```



slashChordSeparator

Chords can be played over a bass note other than the conventional root of the chord. These are known as “inversions” or “slash chords”, because the default way of notating them is with a forward slash between the main chord and the bass note. Therefore the value of `slashChordSeparator` defaults to a forward slash, but you can change it to other markup.

```
\chords {
  c4:7.9- c:7.9-/g
  \set slashChordSeparator = \markup { " over " }
  \break
  c4:7.9- c:7.9-/g
}
```



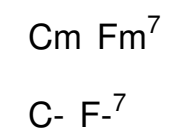
chordNameExceptions

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 follows the `chordRootNamer` output to create the chord name.

minorChordModifier

Minor chords are often denoted via an 'm' suffix to the right of the root of the chord (this is LilyPond's default). However, some idioms prefer other suffixes, such as a minus sign.

```
\chords {
  c4:min f:min7
  \set minorChordModifier = \markup { "-" }
  \break
  c4:min f:min7
}
```



chordPrefixSpacer

The modifier for minor chords as determined by `minorChordModifier` is by default printed immediately to the right of the chord root. Some whitespace can be placed between the root and the modifier by setting `chordPrefixSpacer` (used, for example, in `\italianChords`). No whitespace is inserted if the root is altered.

Predefined commands

`\whiteTriangleMarkup`, `\blackTriangleMarkup`, `\germanChords`, `\semiGermanChords`, `\italianChords`, `\frenchChords`, `\englishChords`.

Selected snippets

Chord name exceptions

The property `chordNameExceptions` stores a list of chord name exceptions to handle cases either not covered or handled incorrectly.

The default chord names used by LilyPond follow the rules as given in Klaus Ignatzek's book "Die Jazzmethode für Klavier 1"; the algorithm to convert chords to chord names can be found in file `scm/chord-ignatzek-names.scm`. Additional rules are given as chord exceptions and stored in the variable `ignatzekExceptions`, as set up in file `ly/chord-modifiers-init.ly`.

This snippet modifies these exceptions in three steps.

1. Set up some music with chords and associated markup. By convention, the root (i.e., the lowest note) of each chord should have pitch c.
2. Call Scheme function `sequential-music-to-chord-exceptions` to create a new list of exceptions, then concatenate it with the existing ones. Since `ignatzekExceptions` is set up with this function's second parameter set to `#t` (to ignore the root of the chords), we have to do the same.
3. Register the new exception list.

```
% Step 1: Define music with chords and markup for maj9 and 6(add9).
chExceptionMusic = {
  <c e g b d'>-\markup { \super "maj9" }
  <c e g a d'>-\markup { \super "6(add9)" }
}

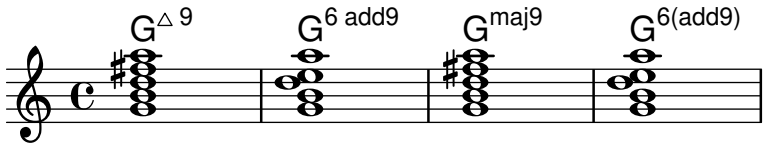
% Step 2: Create extended exception list.
chExceptions =
```

```
#(append (sequential-music-to-chord-exceptions chExceptionMusic #t)
         ignatzekExceptions)

theMusic = \chordmode {
  g1:maj9 g1:6.9
  % Step 3: Register extended exception list.
  \set chordNameExceptions = #chExceptions
  g1:maj9 g1:6.9
}

<<
  \new ChordNames \theMusic
  \new Voice \theMusic
>>

\layout {
  line-width = 10\cm
  ragged-right = ##f
}
```



Chord name major7

The layout of the major 7 can be tuned with the majorSevenSymbol context property.

```
\chords {
  c:7+
  \set majorSevenSymbol = \markup { j7 }
  c:7+
}
```

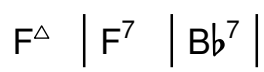


Adding bar lines to ChordNames context

To add bar line indications in the ChordNames context, add the Bar_engraver.

```
\new ChordNames \with {
  \override BarLine.bar-extent = #'(-1 . 3)
  \consists "Bar_engraver"
}

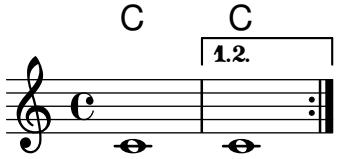
\chordmode {
  f1:maj7 f:7 bes:7
}
```



Volta below chords

By adding the Volta_engraver to the relevant staff, volte can be put below 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
}
```


See also

Notation Reference: Section B.1 [Chord name chart], page 890, Section B.2 [Common chord modifiers], page 890.

Essay on automated music engraving: Section “Literature list” in *Essay*.

Installed Files: `scm/chords-ignatzek-names.scm`, `scm/chord-entry.scm`, `ly/chord-modifiers-init.ly`.

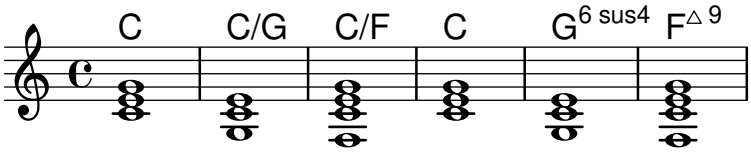
Snippets: Section “Chord notation” in *Snippets*.

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.

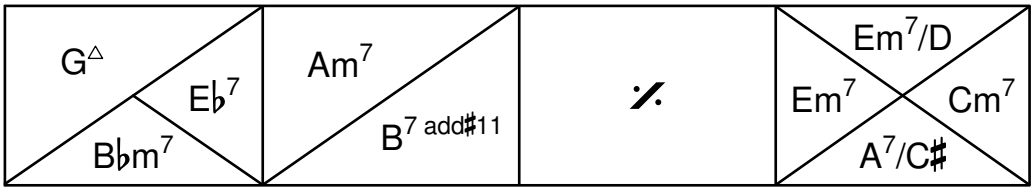
```
myChords = \relative c' {
  \chordmode { c1 c/g c/f }
  <c e g>1 <g c e> <f c' e g>
}
<<
  \new ChordNames { \myChords }
  \new Staff { \myChords }
```

```
>>
```



15.2.3 Chord grids

In some European countries, particularly France, jazz musicians use so-called 'chord grids', which notate chords visually by placing them in squares.



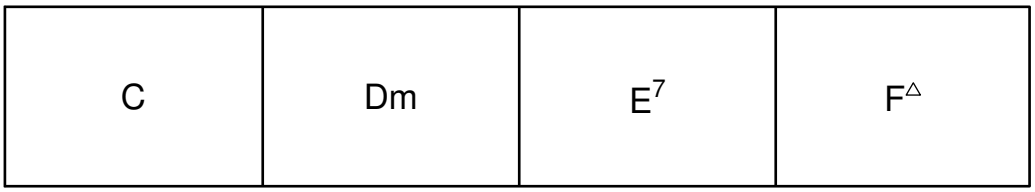
Although they are omitted in the rest of this section for brevity, it is recommended to use the following `\paper` settings for chord grids:

```
\paper {
  indent = 0
  ragged-right = ##f
}
```

`indent = 0` ensures that the first line is not indented as it would normally be (see Section 26.5.3 [`\paper` variables for shifts and indents], page 678). `ragged-right = ##f` is necessary for single-line grids to ensure they span the whole page; see Section 26.5.1 [`\paper` variables for widths and margins], page 676.

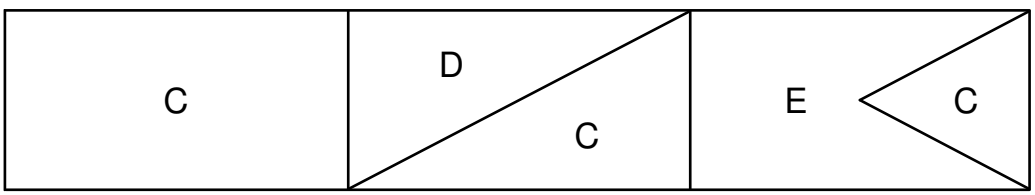
In order to create a chord grid, instantiate a `ChordGrid` context.

```
\new ChordGrid \chordmode { c1 d1:m e1:7 f1:7+ }
```

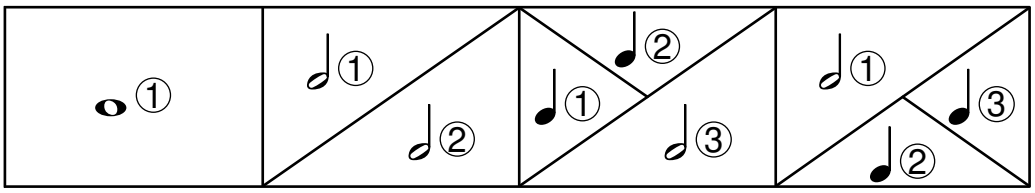


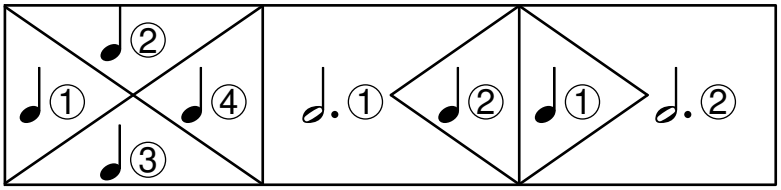
Each square is automatically subdivided.

```
\new ChordGrid \chordmode {
  c1
  d2 c2
  e2. c4
}
```



Chords spanning a complete measure are centered within their square. Chords lasting half a measure take half the square, and those lasting a quarter of a measure take a quarter of the square. This summary picture shows the default rules for subdividing the square:

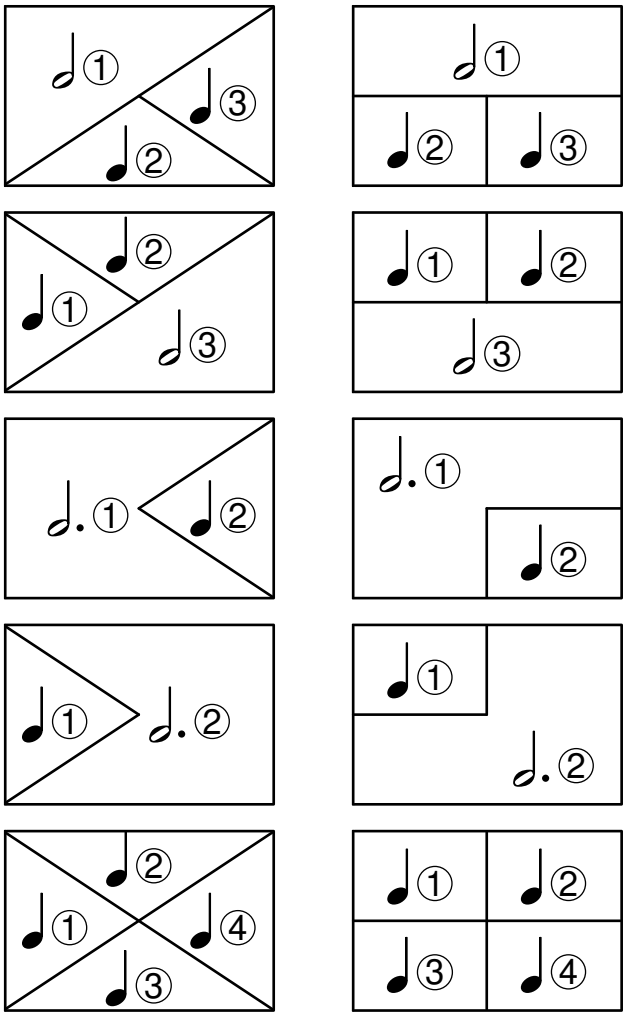




The `\medianChordGridStyle` changes the default display of squares with particular measure divisions to use the style recommended by Philippe Baudoin in his book *Jazz, mode d'emploi* ("Jazz, user instructions").

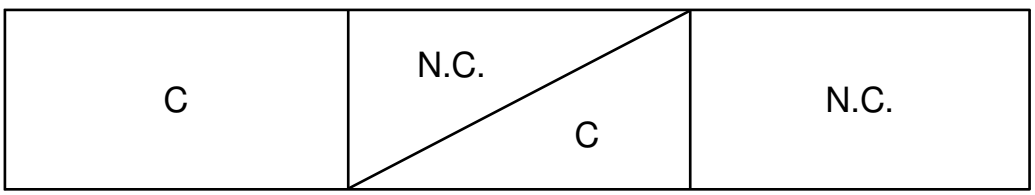
```
\layout {
  \context {
    \ChordGrid
    \medianChordGridStyle
  }
}
```

Default style `\medianChordGridStyle`



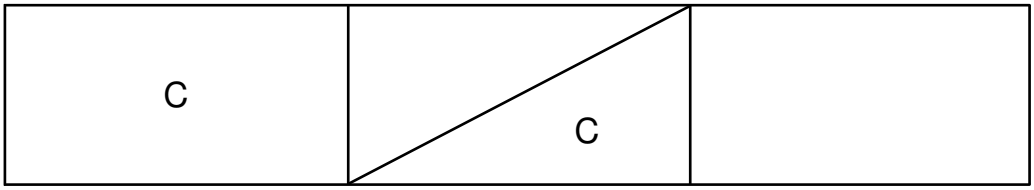
In chord grids, rests cause the `noChordSymbol` to be printed, just like in a regular `ChordNames` context (see Section 15.2.1 [Printing chord names], page 513).

```
\new ChordGrid \chordmode { c1 r2 c2 R1 }
```



Skips cause blank space. They can occupy part of a square.

```
\new ChordGrid \chordmode { c1 s2 c2 s1 }
```



Selected snippets

Customizing the chord grid style

Custom divisions of chord squares can be defined through the `measure-division-lines-alist` and `measure-division-chord-placement-alist` properties of `ChordSquare`. These are both alists. Their keys are measure divisions, namely lists which give the fraction of the measure that each chord (or rest, or skip) represents. More precisely, a measure division alist is made of positive, exact numbers adding up to 1, for example: `'(1/2 1/4 1/4)`. The exactness requirement means that, e.g., `1/2` is valid but not `0.5`.

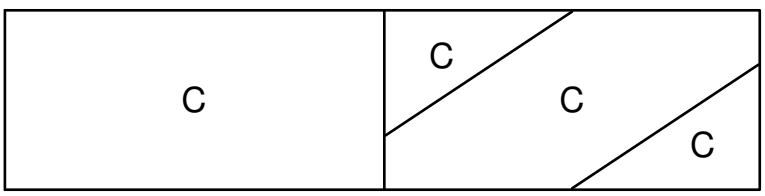
The values in `measure-division-lines-alist` are lists of lines, which are represented as `(x1 y1 x2 y2)`. The line starts at the point `(x1 . y1)` and ends at `(x2 . y2)`. Coordinates are expressed in the [-1, 1] scale relative to the extent of the square.

The values in `measure-division-chord-placement-alist` are lists of `(x . y)` pairs giving the placement of the respective chords.

This example defines a peculiar chord grid style that has a rule for measures divided in three equal parts.

```
\paper {
  line-width = 10\cm
  ragged-right = ##f
}

\new ChordGrid \with {
  \override ChordSquare.measure-division-lines-alist =
    #'(((1) . ())
       ((1/3 1/3 1/3) . ((-1 -0.4 0 1) (0 -1 1 0.4))))
  \override ChordSquare.measure-division-chord-placement-alist =
    #'(((1) . ((0 . 0)))
       ((1/3 1/3 1/3) . ((-0.7 . 0.5) (0 . 0) (0.7 . -0.5))))
}
\chordmode {
  \time 3/4
  c2.
  c4 c4 c4
}
```
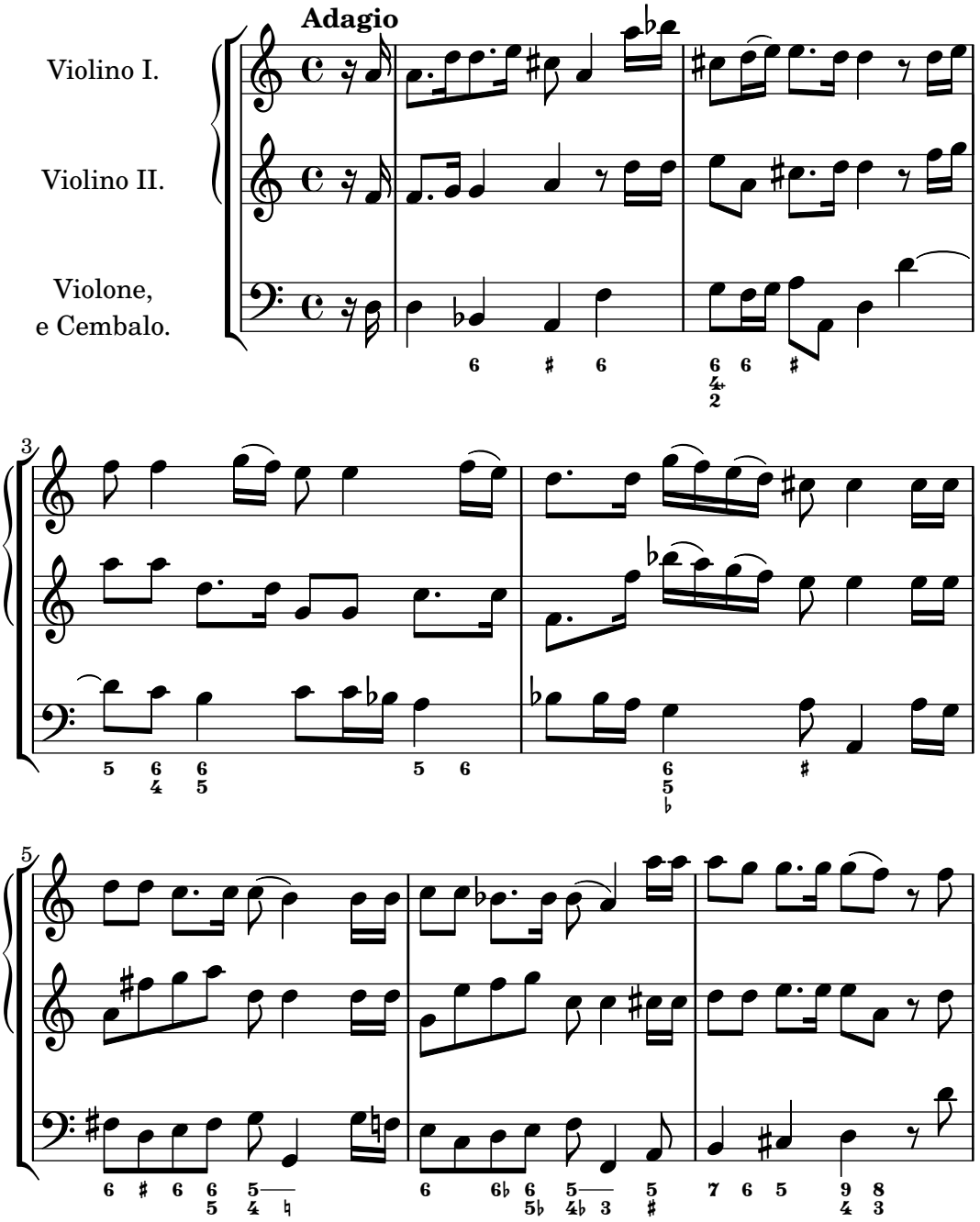


See also

Music Glossary: Section “chord grid” in *Music Glossary*.

Internals Reference: Section “ChordGrid” in *Internals Reference*, Section “ChordGridScore” in *Internals Reference*, Section “GridChordName” in *Internals Reference*, Section “ChordSquare” in *Internals Reference*, Section “Grid_chord_name_engraver” in *Internals Reference*, Section “Chord_square_engraver” in *Internals Reference*.

15.3 Figured bass

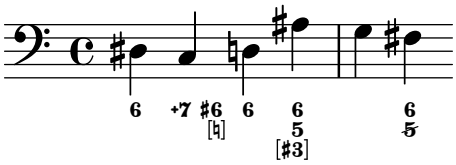


Figured bass notation can be displayed.

15.3.1 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 “Chord notation” in *Snippets*.

15.3.2 Entering figured bass

`\figuremode` is used to switch the input mode to figure mode. See Chapter 19 [Input modes], page 589, for more information on different input modes.

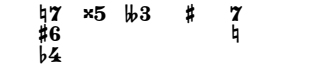
In figure mode, a group of bass figures is delimited by ‘<’ and ‘>’. The duration is entered after the ‘>’.

```
\new FiguredBass {
  \figuremode {
    <6 4>2
  }
}
```



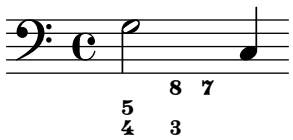
Accidentals (including naturals) may be used for modifying scale steps. These are entered by appending ‘+’ (for sharps), ‘-’ (for flats) or ‘!’ (for naturals) after the number. For double accidentals the modifier is applied twice. For the modification of the third step the number is often omitted, which can be achieved by using ‘_’ instead of a number.

```
\figures {
  <7! 6+ 4-> <5++> <3--> <_+> <7 _!>
}
```



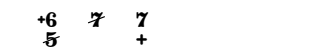
If used without accidental, '_' creates an empty figure which nevertheless takes up space. This can be used for controlling the stacking of bass figures.

```
<<
  {
    \clef bass
    g2 c4
  }
  \figures {
    <_ 5 4>4 <8 _ 3>8 <7>
  }
>>
```



Augmented and diminished steps can be indicated.

```
\figures {
  <6\+ 5/> <7/> <7 _\+>
}
```



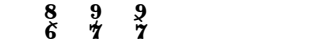
A backward slash through a figure is also available.

```
\figures {
  <5> <5\\>
}
```



For some figures, special backward slash glyphs are provided.

```
\figures {
  <8 6\\> <9 7\\> <9\\ 7>
}
```



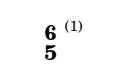
Brackets can be added around accidentals, figures, and consecutive groups of figures.

```
\figures {
  <9[-] 8 [7-] 5 [4[!] 2+]>
}
```



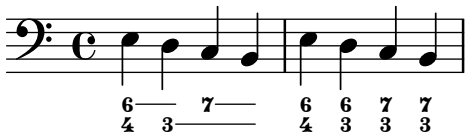
Any text markup can be inserted as a figure.

```
\figures {
  <\markup { \fontsize #-5 \number 6 \teeny \super (1) } 5>
}
```



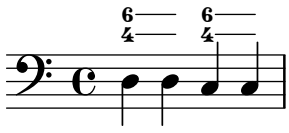
Continuation lines can be used to indicate repeated figures.

```
<<
  {
    \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>
  }
>>
```

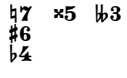


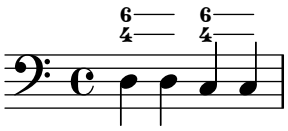
In this case, the extender lines replace existing figures, unless the continuation lines have been explicitly terminated with `\!`.

```
<<
  \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	
\+, /	augmented and diminished steps	

\\	raised by a semitone	
\!	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.

If plus signs appear after the number, specially designed glyphs are provided for some figures.

```
#(set-global-staff-size 26)

\figures {
  <5\+> <5+ 4\+> <6 4- 2\+> r
  \set figuredBassAlterationDirection = #RIGHT
  <5\+> <5+ 4\+> <6 4- 2\+> r
  \set figuredBassPlusDirection = #RIGHT
  <5\+> <5+ 4\+> <6 4- 2\+> r
  \set figuredBassAlterationDirection = #LEFT
  <5\+> <5+ 4\+> <6 4- 2\+> r
}
```



Adjusting figured bass alteration glyphs

In figured bass, specially designed glyphs for `6\\`, `7\\`, and `9\\` are used by default. Similarly, specially designed glyphs for symbols `2\+`, `4\+`, and `5\+` are used by default if plus signs appear after the number.

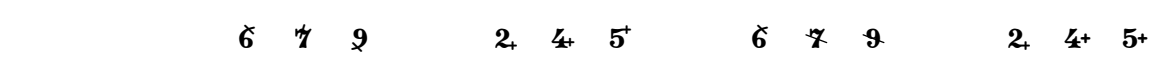
To change that, pass an alist to `figuredBassPlusStrokedAlist` and set the glyph in question to `#f` (or omit it).

```
#(set-global-staff-size 26)

\figures {
  \set figuredBassPlusDirection = #RIGHT
  <6\\> <7\\> <9\\> r
  <2\+> <4\+> <5\+> r

  \set figuredBassPlusStrokedAlist =
    #'((2 . "figbass.twoplus")
       ;; (4 . "figbass.fourplus")
       ;; (5 . "figbass.fiveplus")
       (6 . "figbass.sixstroked")
       ;; (7 . "figbass.sevenstroked")
```

```
        ;; (9 . "figbass.ninestroked")
      )
  <6\\> <7\\> <9\\> r
  <2\+> <4\+> <5\+> r
}
```



See also

Snippets: Section “Chord notation” in *Snippets*.

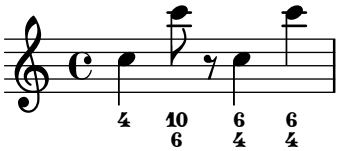
Internals Reference: Section “BassFigure” in *Internals Reference*, Section “BassFigure-Alignment” in *Internals Reference*, Section “BassFigureLine” in *Internals Reference*, Section “BassFigureBracket” in *Internals Reference*, Section “BassFigureContinuation” in *Internals Reference*, Section “FiguredBass” in *Internals Reference*.

15.3.3 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''4 c'8 r8 c,4 c'
  }
  \new FiguredBass {
    \figuremode {
      <4>4 <10 6>8 s8
      <6 4>4 <6 4>
    }
  }
>>
```
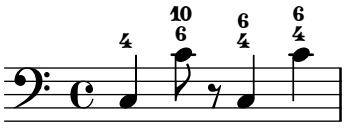


In the example above, the `FiguredBass` context must be explicitly instantiated to avoid creating a second (empty) staff.

Figured bass can also be added to `Staff` contexts directly. In this case, the vertical position of the figures is adjusted automatically.

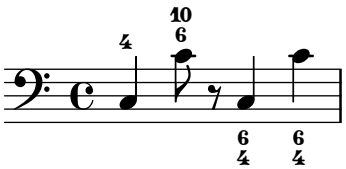
```
<<
  \new Staff = "myStaff"
  \figuremode {
    <4>4 <10 6>8 s8
    <6 4>4 <6 4>
  }
  %% Put notes on same Staff as figures
  \context Staff = "myStaff" {
    \clef bass
    c4 c'8 r8 c4 c'
  }
```

```
>>
```



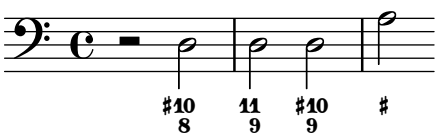
When added in a `Staff` context, figured bass can be displayed above or below the staff.

```
<<
  \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'
  }
>>
```



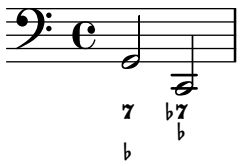
The horizontal alignment of numbers in a figured bass stack that have more than a single digit can be controlled with the context property `figuredBassLargeNumberAlignment`.

```
<<
  \new Voice {
    \clef bass
    r2 d | d d | a2
  }
  \new FiguredBass \figuremode {
    s2 <10+ 8> |
    \set figuredBassLargeNumberAlignment = #RIGHT
    <11 9>2
    \set figuredBassLargeNumberAlignment = #LEFT
    <10+ 9>2 |
    <_+>2
  }
>>
```



The vertical distance of figured bass elements can be controlled with subproperties `minimum-distance` and `padding` of `staff-staff-spacing`.

```
<<
  { \clef bass g,2 c, }
  \figures {
    \once \override BassFigureLine
                    .staff-staff-spacing.minimum-distance = 3
    <7 _-> <7- _->
  }
>>
```



Predefined commands

\bassFigureStaffAlignmentDown, \bassFigureStaffAlignmentUp, \bassFigureStaffAlignmentNeutral.

See also

Snippets: Section "Chord notation" in *Snippets*.

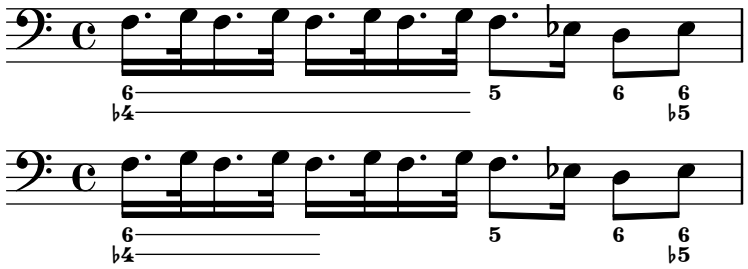
Internals Reference: Section "BassFigure" in *Internals Reference*, Section "BassFigureAlignment" in *Internals Reference*, Section "BassFigureLine" in *Internals Reference*, Section "BassFigureBracket" in *Internals Reference*, Section "BassFigureContinuation" in *Internals Reference*, Section "FiguredBass" 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
    \*4 { f16. g32 } f8. es16 d8 es
  }
  \figures {
    \bassFigureExtendersOn
    % The extenders are correct here,
    % with the same rhythm as the bass.
    \*4 { <6 4->16. <6 4->32 }
    <5>8. r16 <6>8 <6\! 5->
  }
>>
<<
  {
    \clef bass
    \*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->
  }
>>
```



16 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.

16.1 Pitch and harmony in contemporary music

This section highlights issues that are relevant to notating pitch and harmony in contemporary music.

16.1.1 References for pitch and harmony in contemporary music

- Standard quarter tone notation is addressed in Section 1.1.5 [Note names in other languages], page 10.
- Non-standard key signatures are addressed in Section 1.3.2 [Key signature], page 24.
- Contemporary practices in displaying accidentals are addressed in Section 1.3.5 [Automatic accidentals], page 31.

16.1.2 Microtonal notation

16.1.3 Contemporary key signatures and harmony

16.2 Contemporary approaches to rhythm

This section highlights issues that are relevant to the notation of rhythm in contemporary music.

16.2.1 References for contemporary approaches to rhythm

- Complex time signatures are addressed in Section 2.3.2 [Time signature], page 80.
- Basic polymetric notation is addressed in Section 2.3.6 [Polymetric notation], page 95.
- Feathered beams are addressed in Section 2.4.4 [Feathered beams], page 119.
- Mensurstriche bar lines (bar lines between staves only) are addressed in Section 6.1.2 [Grouping staves], page 244.

16.2.2 Tuplets in contemporary music

16.2.3 Contemporary time signatures

16.2.4 Extended polymetric notation

16.2.5 Beams in contemporary music

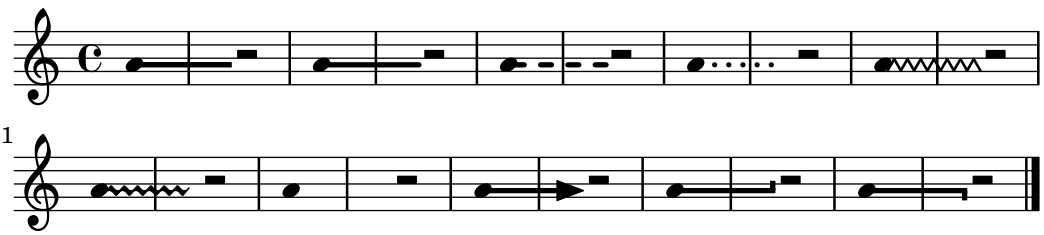
16.2.6 Bar lines in contemporary music

16.3 Graphical notation

Rhythmic items may be continued by a duration line, which gets represented by a `DurationLine` grob. Possible styles are `'beam`, `'line`, `'dashed-line`, `'dotted-line`, `'zigzag`, `'trill` and `'none`. The duration line may end with a hook (beam-style only) or an arrow.

```
\layout {
  \context {
    \Voice
    \consists Duration_line_engraver
    \omit Stem
    \omit Flag
    \omit Beam
    \override NoteHead.duration-log = 2
  }
}

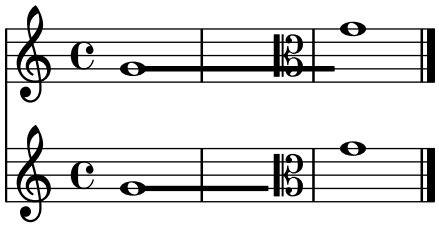
{
  a'1\- s2 r
  \once \override DurationLine.style = #'line
  a'1\- s2 r
  \once \override DurationLine.style = #'dashed-line
  \once \override DurationLine.dash-period = 2
  a'1\- s2 r
  \once \override DurationLine.style = #'dotted-line
  \once \override DurationLine.dash-period = 1
  \once \override DurationLine.bound-details.right.padding = 1
  a'1\- s2 r
  \once \override DurationLine.thickness = 2
  \once \override DurationLine.style = #'zigzag
  a'1\- s2 r
  \once \override DurationLine.style = #'trill
  a'1\- s2 r
  \once \override DurationLine.style = #'none
  a'1\- s2 r
  \once \override DurationLine.bound-details.right.end-style = #'arrow
  a'1\- s2 r
  \override DurationLine.bound-details.right.end-style = #'hook
  a'1\- s2 r
  \override DurationLine.details.hook-direction = #DOWN
  a'1\- s2 r
  \bar "|."
}
```



DurationLine may avoid mid-line items from BreakAlignGroup.

```
\layout {
  \context {
    \Voice
    \consists "Duration_line_engraver"
  }
}
```

```
<<
  \new Staff {
    g'1\- s \clef "alto" g'
  }
  \new Staff {
    \override DurationLine.bound-details
                          .right.end-on-break-align-group = ##t
    g'1\- s \clef "alto" g' \bar "|."
  }
>>
```



16.4 Contemporary scoring techniques

16.5 New instrumental techniques

16.6 Further reading and scores of interest

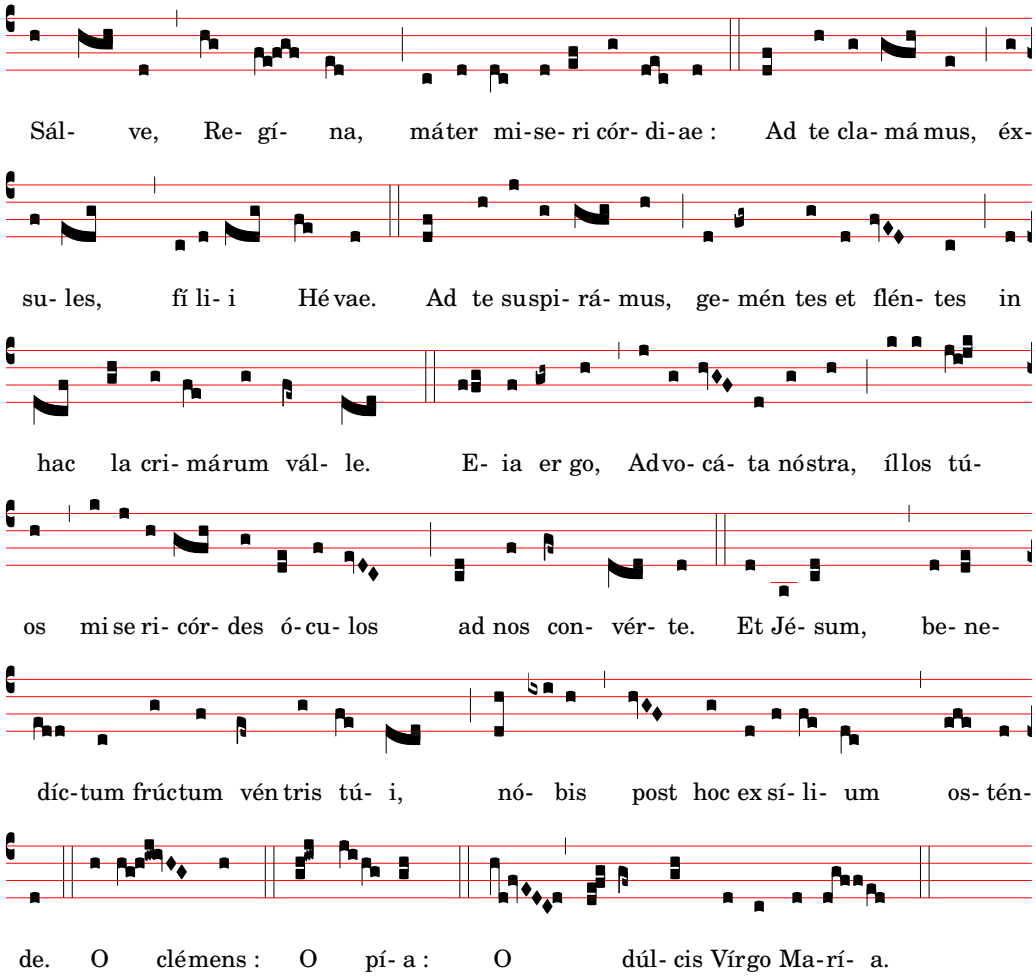
This section suggests books, musical examples and other resources useful in studying contemporary musical notation.

16.6.1 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]

16.6.2 Scores and musical examples

17 Ancient notation



Support for ancient notation includes features for mensural notation, Gregorian chant notation, and Kievan square 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 predefined contexts for these styles.

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

- Section 17.3.4 [Mensural note heads], page 544,
- Section 17.3.7 [Mensural accidentals and key signatures], page 546,
- Section 17.3.6 [Mensural rests], page 546,
- Section 17.3.2 [Mensural clefs], page 543,
- Section 17.4.2 [Gregorian clefs], page 555,
- Section 17.3.5 [Mensural flags], page 545,
- Section 17.3.3 [Mensural time signatures], page 544.

Some notational concepts are introduced specifically for ancient notation,

- Section 17.2.3 [Custodes], page 541,
- Section 17.4.4 [Divisiones], page 556,
- Section 17.2.2 [Ligatures], page 540.

See also

Music Glossary: Section “custos” in *Music Glossary*, Section “ligature” in *Music Glossary*, Section “mensural notation” in *Music Glossary*.

Notation Reference: Section 17.3.4 [Mensural note heads], page 544, Section 17.3.7 [Mensural accidentals and key signatures], page 546, Section 17.3.6 [Mensural rests], page 546, Section 17.4.2 [Gregorian clefs], page 555, Section 17.3.5 [Mensural flags], page 545, Section 17.3.3 [Mensural time signatures], page 544, Section 17.2.3 [Custodes], page 541, Section 17.4.4 [Divisiones], page 556, Section 17.2.2 [Ligatures], page 540.

17.1 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 note heads, 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 note heads and its rests which approach a hand-drawn style.
- The *Neomensural* style is a modernized and stylized version of the former: the note heads 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 *Petrucci* 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 note heads (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 note heads, classical rests and Vaticana clefs in the same piece, if one wishes.

See also

Music Glossary: Section “mensural notation” in *Music Glossary*, Section “flag” in *Music Glossary*.

17.2 Ancient notation – common features

17.2.1 Predefined contexts

For Gregorian chant and mensural notation, there are predefined voice, staff, and score 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 predefined contexts `VaticanaScore`, `VaticanaVoice`, `VaticanaStaff`, `MensuralVoice`, `MensuralStaff`, `PetrucciStaff`, `PetrucciVoice`, `KievanVoice`, and `KievanStaff`.

See also

Music Glossary: Section "mensural notation" in *Music Glossary*.
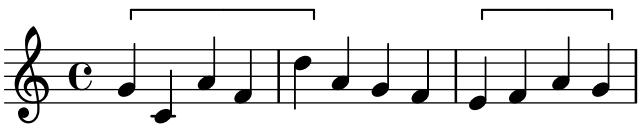
Notation Reference: Section 17.4.1 [Gregorian chant contexts], page 555, Section 17.3.1 [Mensural contexts], page 542, Section 17.5.1 [Kievan contexts], page 564.

17.2.2 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.

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



Three other ligature styles are available: 'Vaticana' for Gregorian chant, 'Mensural' for mensural music (only white mensural ligatures are supported for mensural music, and with certain limitations), and 'Kievan' for Kievan melismata. 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. For more information, see Section 17.3.10 [White mensural ligatures], page 549, Section 17.4.7 [Gregorian square neume ligatures], page 558, and Section 17.5.6 [Kievan melismata], page 566.

See also

Music Glossary: Section "ligature" in *Music Glossary*.

Notation Reference: Section 17.3.10 [White mensural ligatures], page 549, Section 17.4.7 [Gregorian square neume ligatures], page 558.

Known issues and warnings

Spacing required for ligatures is not currently implemented and, as a result, there may end up being too much space between them. Line breaking may also be unsatisfactory.

Lyrics might not align as expected when using ligatures.

Accidentals must not be printed within a ligature, but instead 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\[ ... note\]`.

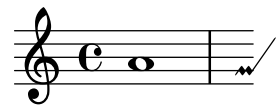
17.2.3 Custodes
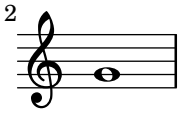
A *custos* (plural: *custodes*; Latin word for "guard") is a symbol that appears at the end of a staff. It anticipates the pitch of the first note of the following line, thus helping the performer to manage line breaks during performance.

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:

```
\score {
  \relative {
    a'1
    \break
    g
  }
  \layout {
    \context {
      \Staff
      \consists Custos_engraver
      \override Custos.style = #'mensural
    }
  }
}
```

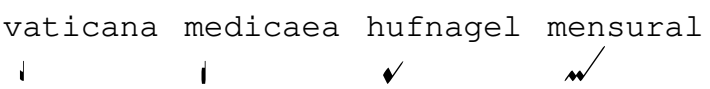




The custos glyph is selected by the `style` property. The styles supported are `vaticana`, `medicaea`, `hufnagel`, and `mensural`.

```
\new Lyrics \lyricmode {
  \markup { \column {
    \typewriter "vaticana "
    \line { " " \musicglyph "custodes.vaticana.u0" }
  } }
  \markup { \column {
    \typewriter "medicaea "
    \line { " " \musicglyph "custodes.medicaea.u0" }
  }}
  \markup { \column {
```

```
    \typewriter "hufnagel "
    \line { " " \musicglyph "custodes.hufnagel.u0" }
  }}
  \markup { \column {
    \typewriter "mensural "
    \line { " " \musicglyph "custodes.mensural.u0" }
  }}
}
```



See also

Music Glossary: Section “custos” in *Music Glossary*.

Snippets: Section “Ancient notation” in *Snippets*.

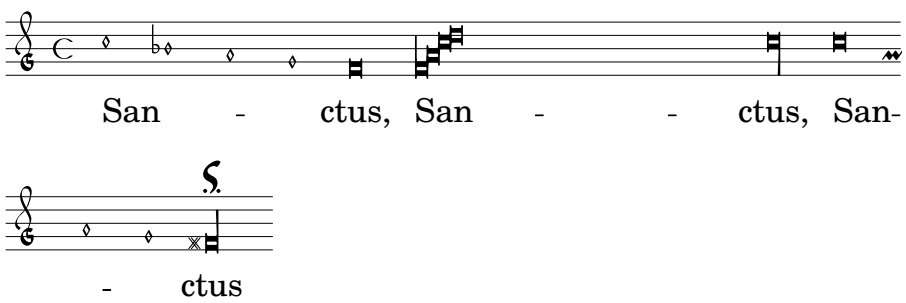
Internals Reference: Section “Custos” in *Internals Reference*.

17.3 Typesetting mensural music

17.3.1 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" \relative {
      \hide Score.BarNumber {
        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
    }
  >>
}
```



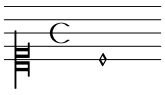
See also

Music Glossary: Section “mensural notation” in *Music Glossary*.
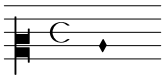
17.3.2 Mensural clefs

Mensural clefs are supported using 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.

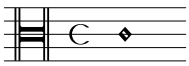
```
\new MensuralStaff {
  \clef "mensural-c1" c'1
}
```
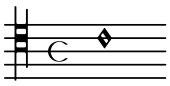


```
\new MensuralStaff {
  \override NoteHead.style = #'blackmensural
  \clef "blackmensural-c2" c'1
}
```



```
\new MensuralStaff {
  \override NoteHead.style = #'neomensural
  \clef "neomensural-c3" c'1
}
```



```
\new PetrucciStaff {
  \clef "petrucci-c4" c'1
}
```



It is possible to manually force a clef glyph to be typeset on an arbitrary line, see Section 1.3.1 [Clef], page 19. For the complete range of possible clefs, see Section B.11 [Clef styles], page 923.

See also

Music Glossary: Section “mensural notation” in *Music Glossary*, Section “clef” in *Music Glossary*.

Notation Reference: Section 17.4.2 [Gregorian clefs], page 555, Section 1.3.1 [Clef], page 19.

Installed Files: `scm/parser-clef.scm`.

Snippets: Section “Pitches” in *Snippets*.

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

Known issues and warnings

The Mensural g clef is mapped to the Petrucci g clef.

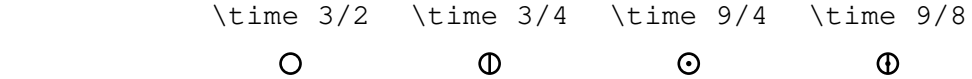
17.3.3 Mensural time signatures

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.

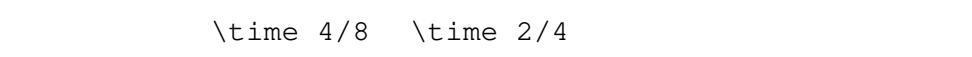
`\time 4/4` `\time 2/2` `\time 6/4` `\time 6/8`



`\time 3/2` `\time 3/4` `\time 9/4` `\time 9/8`

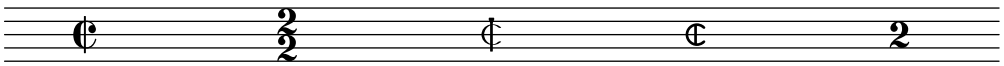


`\time 4/8` `\time 2/4`



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:

`default` `numbered` `mensural` `neomensural` `single-number`



See Section 2.3.2 [Time signature], page 80, for a general introduction to the use of time signatures.

See also

Music Glossary: Section “mensural notation” in *Music Glossary*.

Notation Reference: Section 2.3.2 [Time signature], page 80.

Known issues and warnings

Ratios of note durations cannot change with the time signature, as those are not constant. For example, the ratio of 1 breve = 3 semibreves (*tempus perfectum*) can 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.

17.3.4 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`, `petrucci`, `blackpetrucci` or `semipetrucci`.

The `baroque` style differs from the `default` style by:

- Providing a `maxima` note head, 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
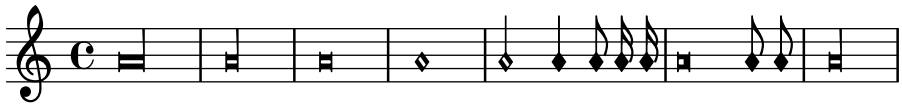
- Centering the stems on the note heads.

The `blackpetrucci` style produces note heads usable in black mensural notation or coloratio sections in white mensural notation. Because note head style does not influence flag count, in this style a semiminima should be notated as `a8*2`, not `a4`, otherwise it will look like a minima. The multiplier can be different if coloratio is used, e.g., to notate triplets.

Use `semipetrucci` style to draw half-colored note heads (breves, longas and maximas).

The following example demonstrates the `petrucci` style:

```
\compressEmptyMeasures
\autoBeamOff
\override NoteHead.style = #'petrucci
a'\maxima a'\longa a'\breve a'1 a'2 a'4 a'8 a'16 a'
\override NoteHead.style = #'semipetrucci
a'\breve*5/6
\override NoteHead.style = #'blackpetrucci
a'8*4/3 a'
\override NoteHead.style = #'petrucci
a'\longa
```



Section B.9 [Note head styles], page 922, gives an overview of all available note head styles.

See also

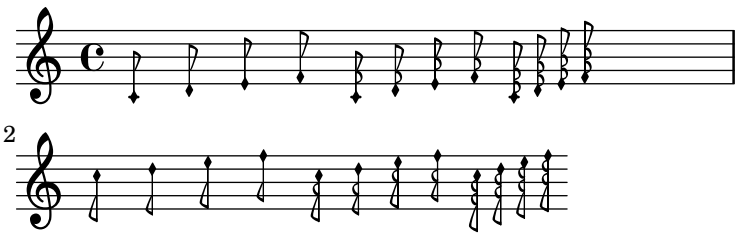
Music Glossary: Section “mensural notation” in *Music Glossary*, Section “note head” in *Music Glossary*.

Notation Reference: Section B.9 [Note head styles], page 922.

17.3.5 Mensural flags

Use the `style` property of grob `Flag` to select ancient flags. Besides the `default` flag style, only the `mensural` style is supported.

```
\relative c' {
  \override Flag.style = #'mensural
  \override Stem.thickness = 1.0
  \override NoteHead.style = #'mensural
  \autoBeamOff
  c8 d e f c16 d e f c32 d e f s8
  c'8 d e f c16 d e f c32 d e f
}
```



Note that the innermost flare of each mensural flag 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.

See also

Music Glossary: Section “mensural notation” in *Music Glossary*, Section “flag” in *Music Glossary*.

Known issues and warnings

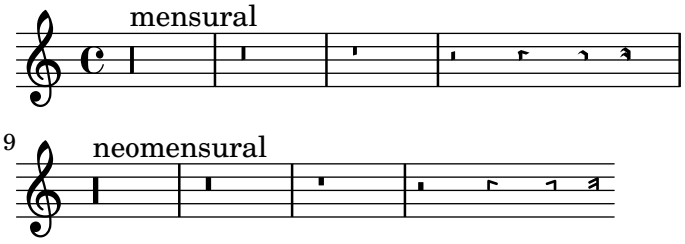
Vertically aligning each flag with a staff line assumes that stems always end either exactly on or exactly in the middle of two staff lines. This may not always be true when using advanced layout features of classical notation (which however are typically out of scope for mensural notation).

17.3.6 Mensural rests

Use the `style` property of grob `Rest` to select ancient rests. Supported ancient styles are `neomensural`, and `mensural`.

The following example demonstrates these styles:

```
\compressEmptyMeasures
\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 32nd and 64th rests specifically for the mensural or neo-mensural styles. Rests from the default style are used.

See also

Music Glossary: Section “mensural notation” in *Music Glossary*.

Notation Reference: Section 2.2.1 [Rests], page 67.

Snippets: Section “Ancient notation” in *Snippets*.

Known issues and warnings

The glyph for the maxima rest in mensural style is actually a perfect longa rest; use two (or three) longa rests to print a maxima rest. Longa rests are not grouped automatically, so have to be done manually by using pitched rests.

17.3.7 Mensural accidentals and key signatures

The `mensural` style provides a sharp and a flat sign different from the default style. Mensural notation rarely used a natural sign: instead the appropriate sharp or flat is used. For example, a B natural in the key of F major would be indicated with a sharp. However, if specifically called for, the natural sign is taken from the `vaticana` style.

mensural



The way to use this style is covered in Section 1.3.6 [Alternate accidental glyphs], page 39. It is the default in the `MensuralStaff` context.

See also

Music Glossary: Section “mensural notation” in *Music Glossary*, Section “Pitch names” in *Music Glossary*, Section “accidental” in *Music Glossary*, Section “key signature” in *Music Glossary*.

Notation Reference: Chapter 1 [Pitches], page 3, Section 1.1.3 [Accidentals], page 8, Section 1.3.5 [Automatic accidentals], page 31, Section 1.3.6 [Alternate accidental glyphs], page 39, Section B.10 [Accidental glyph sets], page 922, Section 1.3.2 [Key signature], page 24.
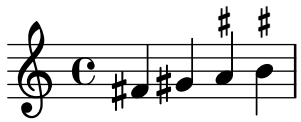
Internals Reference: Section “KeySignature” in *Internals Reference*.

17.3.8 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.
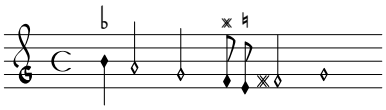
Support for such suggested accidentals is included, and can be switched on by setting `suggestAccidentals` to `#t`.

```
\relative {
  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
  \new MensuralVoice  {
    \once \set suggestAccidentals = ##t
    bes'4 a2 g2 \ficta fis8 \ficta e! fis2 g1
  }
}
```



See also

Internals Reference: Section “Accidental_engraver” in *Internals Reference*, Section “AccidentalSuggestion” in *Internals Reference*.

17.3.9 Historical overview of ligatures

Early polyphony used modal rhythm and encoded the meter by ligatures (as they existed in Gregorian neume notation, see Section 17.4.7 [Gregorian square neume ligatures], page 558), breaking the flow of notes *not by feet*, but after each metric stress, i.e., the last note of a ligature was stressed, the preceding ones were not. The most common ligatures had two or three notes (*ligatura binaria/ternaria*: binary/ternary ligature), corresponding to the stress distribution

of the common meters iamb (short-long), trochee (long-short), dactyl (long-short-short), and anapaest (short-short-long). (Note that long-short is the quantitative concept; stress is the qualitative counterpart of long.)

When modal rhythm was abandoned and note shapes started to encode length, ligatures were not abandoned but (interpreting the meter strictly quantitatively) read as breves followed by a final longa. The pes () indicates an ascending ligature, and the climacus () a descending one.

New shapes were introduced to encode other length sequences, preferably independently of context, so that ligatures of arbitrary length became possible. Shapes for binary brevis-brevis ligatures might have been taken from the first two notes of the existing ternary ligatures (meaning brevis-brevis-longa): the torculus () and the modified scandici () start the same way; the shape  is thus usable as an ascending binary brevis-brevis ligature. Climacus must have been abandoned, as it is indistinguishable from the sequence  of non-ligated notes, so only the porrectus () remained to use its start as the descending form: the oblique parallelogram  is referred to as a *flexa*; it always denotes *two* notes (at the starting and ending pitches), not a scale of all pitches between the two.

Binary ligatures starting with a longa were crafted by tweaking the existing brevis-brevis or brevis-longa shapes. For the descending shapes longa-breve () and longa-longa () the initial stem was removed; for the ascending shapes longa-breve () and longa-longa () a right downward stem was added to the first note.

The pes shape was difficult to draw with void heads if the penultimate pitch was close (the same holds for the porrectus), so such final notes were replaced by virgae (for example,  and , perhaps inspired by the scandicus). Nevertheless, the original shape remained in use for quite a long time, even though it looks like a divisi, and at end of sections it may be impossible to tell which one is meant.

Binary ligatures for semibreve pairs like , , and were also introduced, denoted by an upward left stem.

The following table summarizes the customary binary ligatures.

Lengths	**S S**	**B B**	**L B**	**B L**	**L L**
Ascending					
Descending	,				

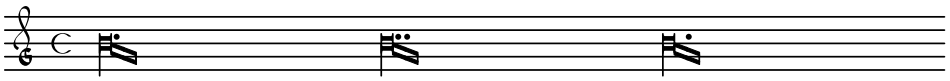
Maximae were easy to introduce into ligatures, because they post-date the era of modal rhythm, so they do not have to comply with old customs: they can appear anywhere within a ligature in their customary form, with or without a downward right stem. In the middle of a ligature, breves and longae were also written in their customary shape (stems of longae point always downwards), except when a pair of two brevis shaped noteheads is merged to form a flexa (which can be compulsory only at the end, but frequently used in the middle too). Rarely initial breves got a left stem even in an ascending configuration, or final longae got a right stem even in descending configurations.

Gregorian neume ligatures could not have been abandoned, since plainchant notation remained in use (some manuscripts contain plainchant and mensural notation side by side). Evolution of mensural ligatures must have been influenced by that – that may be the cause of using almost exclusively breves in the middle of a ligature; that the flexa shape is mostly used when the second note is lower than its neighbors, is probably also a heritage of neumes.

The table above is a quite practical cheat sheet for reading *any* mensural ligature: the first and last notes should be decoded from this table, the rest can be read roughly as is (for "exact" rules see Section 17.3.11 [Reading white mensural ligatures], page 551). The flexa shape poses some practical difficulties: pitches can be hard to decide whether on a line or in a neighboring

space; also notes could be dotted (such a dot appears above or below the next note), and it may be hard to decide whether a dot above a flexa belongs to the first note of the flexa or the preceding one:

```
\new PetrucciVoice {
  \relative {
    \[ b'\breve. a\breve g\breve \]
    \[ b\breve. a\breve. g\breve \]
    \[ b\breve a\breve. g\breve \]
  }
}
```



17.3.10 White mensural ligatures

There is limited support for white mensural ligatures.

To engrave white mensural ligatures, replace the `Ligature_bracket_engraver` with the `Mensural_ligature_engraver` in the `Voice` context's layout block:

```
\layout {
  \context {
    \Voice
    \remove Ligature_bracket_engraver
    \consists Mensural_ligature_engraver
  }
}
```

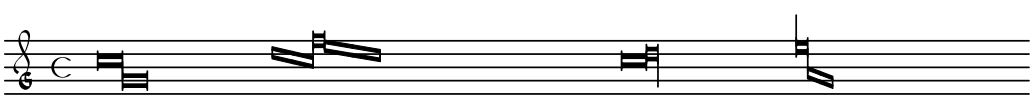
In the following, we use a `PetrucciStaff` context, which does this replacement, among other settings, approximating the mensural typesetting of Ottaviano Petrucci's *Harmonices Musices Odhecaton* (Venice, 1501). The accompanying voice context is called `PetrucciVoice`.

There is no additional input language to describe the shape of a white mensural ligature; instead, the shape is determined solely from the pitches and durations 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.
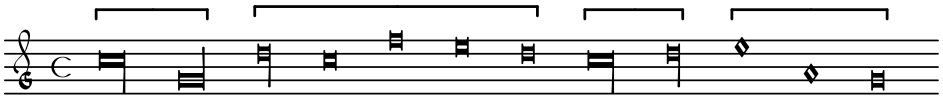
By default optional stems and flexae are not drawn, and final ascending longae are drawn as virgae. To draw alternatives, some tweaks are provided: An optional flexa can be requested by setting the `ligature-flexa` property of the *second* note head. The width of a flexa can be set by the note head property `flexa-width`.

For example,

```
\new PetrucciStaff \relative {
  \[ c''\maxima g \]
  \[ d'\longa
     \tweak ligature-flexa ##t
     \tweak flexa-width #3.2 c\breve f e
     \tweak flexa-width #4.3 d \]
  \[ c\maxima d\longa \]
  \[ e1 a, g\breve \]
}
```



After replacing `Mensural_ligature_engraver` with `Ligature_bracket_engraver`, the same music looks as follows:

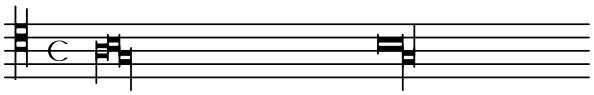


Optional stems can be requested by the following properties:

- `left-down-stem` for downward left stem for an initial breve;
- `right-down-stem` for downward right stem for a maxima or a longa (which is by default stemless in descending start/end circumstances);
- `right-up-stem` for upward right stem for a final maxima/longa.

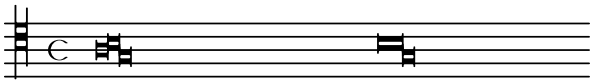
Here is an example that demonstrates these tweaks.

```
\new PetrucciStaff \relative {
  \clef "petrucci-c4"
  \[ \tweak left-down-stem ##t a\breve b
     \tweak right-down-stem ##t g\longa \]
  \[ \tweak right-down-stem ##t b\maxima
     \tweak right-up-stem ##t g\longa \]
}
```
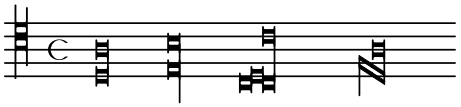


Without tweaking, the same ligatures look as follows:

```
\new PetrucciStaff \relative {
  \clef "petrucci-c4"
  \[ a\breve b g\longa \]
  \[ b\maxima g\longa \]
}
```
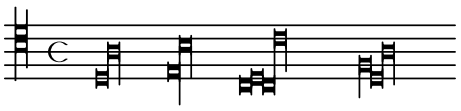


Pes-style final ascending longa can be drawn by `ligature-pes`:

```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
}
\new PetrucciStaff {
  \clef "petrucci-c4"
  \[ d\breve \tweak ligature-pes ##t a\longa \]
  \[ e \tweak ligature-pes ##t b \]
  \[ c\breve d c \tweak ligature-pes ##t c'\longa \]
  \[ f\breve d \tweak ligature-pes ##t a\longa \]
}
```



In the last case LilyPond draws the breves as flexa, otherwise the initial note would collide with the final one. Without tweaking these ligatures look like

```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
}
\new PetrucciStaff {
  \clef "petrucci-c4"
  \[ d\breve a\longa \]
  \[ e b \]
  \[ c\breve d c c'\longa \]
  \[ f\breve d a\longa \]
}
```



See also

Music Glossary: Section “ligature” in *Music Glossary*.

Notation Reference: Section 17.4.7 [Gregorian square neume ligatures], page 558, Section 17.2.2 [Ligatures], page 540.

Known issues and warnings

Horizontal spacing of ligatures may be poor.

Accidentals may collide with previous notes.

17.3.11 Reading white mensural ligatures

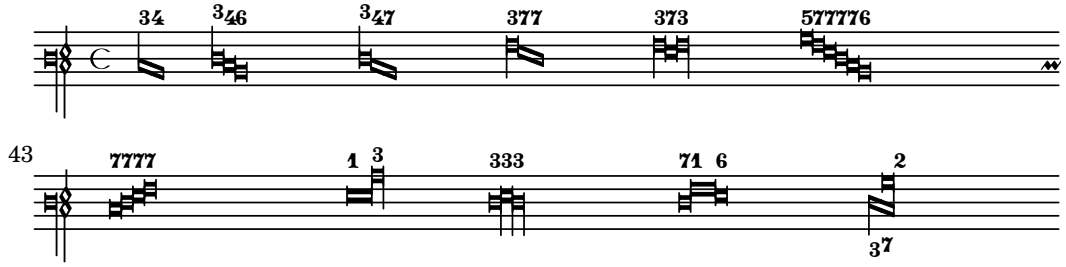
Ligatures are read from left to right and (if, very rarely, two notes appear vertically aligned) bottom up. First the ligature is to be parsed into notes (a flexa shape denotes two notes, any note may be dotted (see above) or colored), and each stem is to be assigned to a note: a final or downward middle stem belongs to the preceding note, or, if two vertically aligned notes precede the stem, to the lower of those (so is a right stem of that note), while an initial or upward middle stem belongs to the next note (so is a left stem – upward middle stems are extremely rare).

Then each note is assigned a length as follows:

1. (rule of maxima) if the shape of the note is maxima, then the length is `\maxima`, otherwise
2. (rule of pes) if the note is aligned above the previous note, then the length is `\longa`, otherwise
3. (rule of stems) if the note has a stem, then the length is semibreve (`1`) for an upward left stem, `\breve` for a downward left stem, and `\longa` for a right stem; otherwise
4. (appendix to the stem rule) if the previous note has an upward left stem, then the length is semibreve (`1`), otherwise
5. (rule of descending start) if the note is the first one, and is higher then the next note, then the length is `\longa`, otherwise
6. (rule of descending end) if the note is the final one, and lower then the penultimate note, and of breve shape, then the length is `\longa`, otherwise

7. the length is \breve.

(Note that lengths shorter than the semibreve cannot be ligated.) Some examples showing which rule applies on which note:



Besides these rules, there are some restrictions/conventions:

A. successive notes must not have the same pitch.

B. vertically aligned notes can occur only at the end of a ligature.

C. a ligature contains at least two notes (occasionally one may encounter  looking like a unary ligature, but it probably fixes scribal mistakes, correcting a breve to semibreve by adding the stem).

D. semibreves can appear only at the beginning and just one pair of them (some English manuscripts have ligatures with four or six semibreves).

E. flexae are descending (however, ascending flexae are ubiquitous in manuscripts not respecting this convention).

F. non-final flexae are turning, i.e., descending flexae are followed by a higher, ascending ones by a lower note.

G. second note of a flexa must not be a longa; in other words, a flexa must not have a right stem.

H. middle notes are breves; this is actually an explicit rule of Tinctoris, does not exactly match praxis, but can be refined in several steps:

I. middle longae and maximae occur in ligatures consisting only of longae and maximae;

J. middle longae are higher than the previous note;

K. middle longae are lower than the next note.
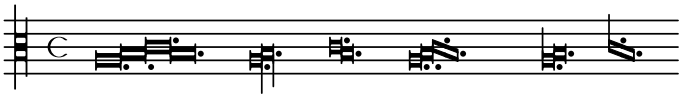
There is also a tendence that maximae in a mixed (not all maxima) ligature have a right stem, except for descending starts and ends.

Some examples showing these restrictions in action:

- In Obrecht's Missa Salve diva parens, the same four-note motif is written in decreasing lengths; while (dotted) maximae or breves can form a four-note ligature, longae and semibreves cannot (restrictions K and D).
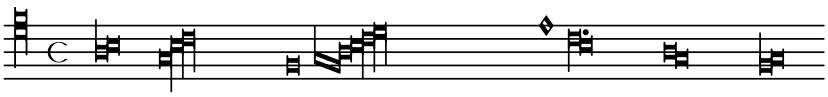
```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
}
\new PetrucciStaff {
  \clef "petrucci-c3"
  \[ a\maxima. b c' b \]
  \[ a\longa. b \] \[ c' b \]
```

```
  \[ a\breve. b c' b \]
  \[ a1. b \] \[ c' \tweak ligature-flexa ##t b \]
}
```



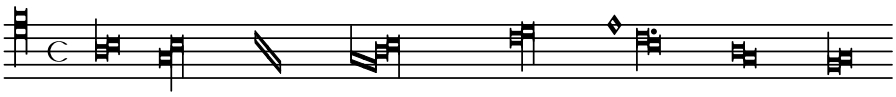
- The Amen in a Credo by Vaqueras is a long melisma; the one in the tenor part is written mostly in ligatures, broken by restrictions D, J, and K (while ignoring H and I):

```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
}
\new PetrucciStaff {
  \clef "petrucci-c5"
  \[ f1 g \] \[ e\longa g a \] d\breve
  \[ e1 \tweak ligature-flexa ##t d f\breve g\longa a b \]
  c'1 \[ a\breve. g\longa \] \[ f e \] \[ d1 e \]
}
```



Observing restriction H would result in

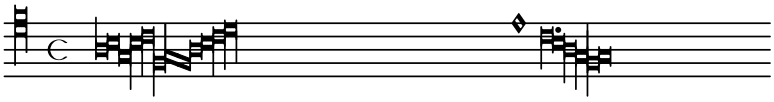
```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
}
\new PetrucciStaff {
  \clef "petrucci-c5"
  \[ f1 g \] \[ e\longa g \] \[ a d\breve \]
  \[ e1 \tweak ligature-flexa ##t d f\breve g\longa \] \[ a b \]
  c'1 \[ a\breve. g\longa \] \[ f e \] \[ d1 e \]
}
```



while ignoring all restrictions results in

```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
```

```
}
\new PetrucciStaff {
  \clef "petrucci-c5"
  \[ f1 g e\longa g a d\breve
     e1 \tweak ligature-flexa ##t d f\breve g\longa a b \]
  c'1 \[ a\breve. g\longa f e d1 e \]
}
```
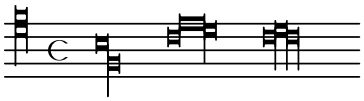


Rule 6 presents a conundrum: how to draw a final brevis if the penultimate note has a right stem (which is extremely rare because of restriction K)? The easy answer is "stop the ligature after the stem", but there are examples when  is to be read as longa-brevis (note that a final longa could always be disambiguated by adding a right stem).

An alternative interpretation is to read this stem as a right stem of the last note, and indeed there is a piece in Trent 89, fol 97, where  is to be read as three breves.

LilyPond draws both a `\breve` and a `\longa` as a brevis head when descending final notes, and allows tweaking `right-down-stem` to disambiguate the latter:

```
\layout {
  \context {
    \Score
    \override SpacingSpanner.base-shortest-duration =
      #(ly:make-moment 16 1)
  }
}
\new PetrucciStaff {
  \clef "petrucci-c5"
  \[ \tweak right-down-stem ##t g\longa d\breve \]
  \[ a\breve \tweak right-down-stem ##t c'\maxima b\longa \]
  \[ a\longa b \tweak right-down-stem ##t a \]
}
```



(All examples are real; in fact the middle one was originally written as , and the stem is a later addition.)

17.4 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 also

Music Glossary: Section "ligature" in *Music Glossary*.

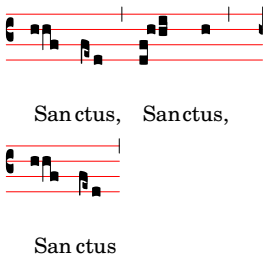
Notation Reference: Section 17.3.10 [White mensural ligatures], page 549, Section 17.2.2 [Ligatures], page 540.

17.4.1 Gregorian chant contexts

The predefined contexts `VaticanaScore`, `VaticanaVoice`, `VaticanaStaff`, and `VaticanaLyrics` can be used to engrave a piece of Gregorian chant in the style of the Editio Vaticana. These contexts initialize all relevant context and grob properties to proper values; you can immediately go ahead entering the chant, as the following excerpt demonstrates.

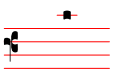
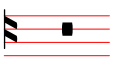
```
\new VaticanaScore {
  <<
    \new VaticanaVoice = "cantus" {
      \[ c'\melisma c' \flexa a \]
      \[ a \flexa \deminutum g\melismaEnd \]
      f \divisioMinima
      \[ f\melisma \pes a c' c' \pes d'\melismaEnd \]
      c' \divisioMinima \break
      \[ c'\melisma c' \flexa a \]
      \[ a \flexa \deminutum g\melismaEnd \] f \divisioMinima
    }
    \new VaticanaLyrics \lyricsto "cantus" {
      San -- ctus, San -- ctus, San -- ctus
    }
  >>
}

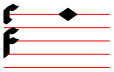
\layout {
  indent = 0
  ragged-last = ##t
}
```



17.4.2 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 Section 1.3.1 [Clef], page 19. 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	`vaticana-do1`, `vaticana-do2`, `vaticana-do3`	
Editio Vaticana style fa clef	`vaticana-fa1`, `vaticana-fa2`	
Editio Medicaea style do clef	`medicaea-do1`, `medicaea-do2`, `medicaea-do3`	

Editio Medicaea style fa clef	`medicaea-fa1`, `medicaea-fa2`	
Hufnagel style do clef	`hufnagel-do1`, `hufnagel-do2`, `hufnagel-do3`	
Hufnagel style fa clef	`hufnagel-fa1`, `hufnagel-fa2`	
Hufnagel style combined do/fa clef	`hufnagel-do-fa`	

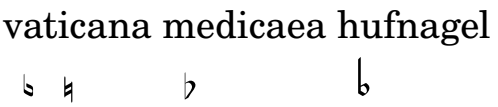
See also

Music Glossary: Section “clef” in *Music Glossary*.

Notation Reference: Section 1.3.1 [Clef], page 19.

17.4.3 Gregorian accidentals and key signatures

Accidentals for the three different Gregorian styles are available:



As shown, not all accidentals are supported by each style. When trying to access an unsupported accidental, LilyPond will switch to a different style.

How to switch between styles is covered in Section 1.3.6 [Alternate accidental glyphs], page 39.

See also

Music Glossary: Section “accidental” in *Music Glossary*, Section “key signature” in *Music Glossary*.

Notation Reference: Chapter 1 [Pitches], page 3, Section 1.1.3 [Accidentals], page 8, Section 1.3.5 [Automatic accidentals], page 31, Section 1.3.6 [Alternate accidental glyphs], page 39, Section 1.3.2 [Key signature], page 24.
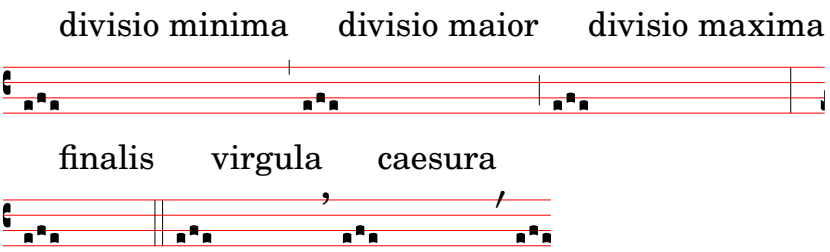
Internals Reference: Section “KeySignature” in *Internals Reference*.

17.4.4 Divisiones

There are no rests in Gregorian chant notation; instead, it uses divisiones.

A *divisio* (plural: *divisiones*; Latin word for ‘division’) is a staff-context symbol indicating 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. 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.

Some editions use *virgula* or *caesura* instead of divisio minima; the predefined staff contexts for Gregorian chant configure `\caesura` to produce an ancient caesura mark.



Predefined commands

`\virgula`, `\caesura`, `\divisioMinima`, `\divisioMaior`, `\divisioMaxima`, `\finalis`.

See also

Music Glossary: Section “caesura” in *Music Glossary*, Section “divisio” in *Music Glossary*.

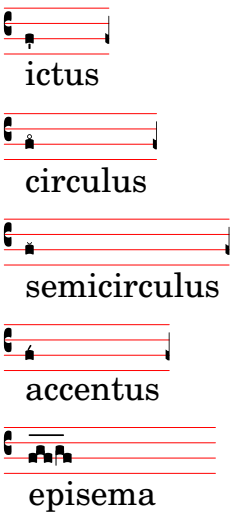
Notation Reference: Section 3.2.3 [Breath marks], page 176.

17.4.5 Gregorian articulation signs

In addition to the standard articulation signs described in section Section 3.1.1 [Articulations and ornamentations], page 156, articulation signs specifically designed for use with notation in *Editio Vaticana* style are provided.

```
\new VaticanaScore {
  \new VaticanaVoice {
    \override Script.padding = -0.1
    a\ictus_"ictus " \break
    a\circulus_"circulus " \break
    a\semicirculus_"semicirculus " \break
    a\accentus_"accentus " \break
    \[ a_"episema" \episemInitium \pes b
       \flexa a b \episemFinis \flexa a \]
  }
}

\layout {
  indent = 0
  ragged-last = ##t
}
```



See also

Notation Reference: Section 3.1.1 [Articulations and ornamentations], page 156.

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.

17.4.6 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`.

```
\new VaticanaScore {
  \new VaticanaVoice {
    \[ \augmentum a \flexa \augmentum g \]
    \augmentum g
  }
}
```



See also

Notation Reference: Section 3.2.3 [Breath marks], page 176.

Internals Reference: Section “Divisio” in *Internals Reference*.

Snippets: Section “Ancient notation” in *Snippets*.

17.4.7 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.

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`, `\stropha`, `\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.

  An alternative to `\pes` and `\flexa` (which internally do exactly the same) is the command `\~`.

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 curvation 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 `\[ a \pes b \flexa g \]` (or `\[ a \~ b \~ g\]`), produces a Torculus consisting of three Punctum heads, while the shape high-low-high, such as `\[ a \flexa g \pes b \]` (or `\[ a \~ g \~ b\]`), 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 `\stropha`.

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, LilyPond 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 neumes

Basic and *Liquescent* forms	**Output**	**LilyPond code**
Punctum		`\[ b \]`
		`\[ \cavum b \]`
		`\[ \linea b \]`
Punctum Auctum Ascendens		`\[ \auctum \ascendens b \]`
Punctum Auctum Descendens		`\[ \auctum \descendens b \]`
Punctum inclinatum		`\[ \inclinatum b \]`
Punctum Inclinatum Auctum		`\[ \inclinatum \auctum b \]`
Punctum Inclinatum Parvum		`\[ \inclinatum \deminutum b \]`
Virga		`\[ \virga b' \]`

Two-note ligatures

Clivis vel Flexa		`\[ b \flexa g \]` (`\[ b \~ g \]`)
Clivis Aucta Descendens		`\[ b \flexa \auctum \descendens g \]`
Clivis Aucta Ascendens		`\[ b \flexa \auctum \ascendens g \]`
Cephalicus		`\[ b \flexa \deminutum g \]`
Podatus/Pes		`\[ g \pes b \]` (`\[ g \~ b \]`)

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		`\[ a \pes b \flexa g \]` (`\[ a \~ b \~ g \]`)
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		`\[ a \flexa g \pes b \]` (`\[ a \~ g \~ b \]`)
Porrectus Auctus Descendens		`\[ a \flexa g \pes \auctum \descendens b \]`
Porrectus Deminutus		`\[ a \flexa g \pes \deminutum b \]`

Climacus 

```
\[ \virga b \inclinatum a
\inclinatum g \]
```

Climacus Auctus 

```
\[ \virga b \inclinatum a
\inclinatum \auctum g \]
```

Climacus Deminutus 

```
\[ \virga b \inclinatum a
\inclinatum \deminutum g \]
```

Scandicus 

```
\[ g \pes a \virga b \]
```

Scandicus Auctus Descendens 

```
\[ g \pes a \pes
\auctum \descendens b \]
```

Scandicus Deminutus 

```
\[ g \pes a \pes \deminutum b \]
```

Special signs

Quilisma		`\[ g \pes \quilisma a \pes b \]`
Quilisma Pes Auctus Descendens		`\[ g \quilisma a \pes \auctum \descendens b \]`
Oriscus		`\[ \oriscus b \]`
Pes Quassus		`\[ \oriscus g \pes \virga b \]`
Pes Quassus Auctus Descendens		`\[ \oriscus g \pes \auctum \descendens b \]`
Salicus		`\[ g \oriscus a \pes \virga b \]`
Salicus Auctus Descendens		`\[ g \oriscus a \pes \auctum \descendens b \]`
(Apo)stropha		`\[ \stropha b \]`
Stropha Aucta		`\[ \stropha \auctum b \]`
Bistropha		`\[ \stropha b \stropha b \]`
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. An alternative to these commands is using `\~`.

Use the unary music function \augmentum to add augmentum dots.

See also

Music Glossary: Section “ligature” in *Music Glossary*.

Notation Reference: Section 17.4.7 [Gregorian square neume ligatures], page 558, Section 17.3.10 [White mensural ligatures], page 549, Section 17.2.2 [Ligatures], page 540.

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.
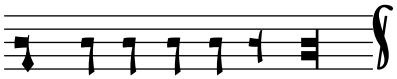
17.5 Typesetting Kievan square notation

17.5.1 Kievan contexts

As with Mensural and Gregorian notation, the predefined KievanVoice and KievanStaff contexts can be used to engrave a piece in square notation. These contexts initialize all relevant context properties and grob properties to proper values, so you can immediately go ahead entering the chant:

```
% Font settings for Cyrillic
\paper {
  property-defaults.fonts.serif = "Linux Libertine O,serif"
}

\score {
  <<
    \new KievanVoice = "melody" \relative c' {
      c4 c c c c2 b\longa \fine
    }
    \new Lyrics \lyricsto "melody" {
      Го -- спо -- ди по -- ми -- луй.
    }
  >>
}
```



Господи помилуй.

See also

Music Glossary: Section “Kievan notation” in *Music Glossary*.
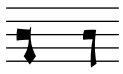
Known issues and warnings

LilyPond supports Kievan notation of the Synodal style, as used in the corpus of chant books printed by the Russian Holy Synod in the 1910’s and recently reprinted by the Moscow Patriarchate Publishing House. LilyPond does not support the older (less common) forms of Kievan notation that were used in Galicia to notate Rusyn plainchant.

17.5.2 Kievan clefs

There is only one clef used in Kievan notation (the Tse-fa-ut Clef). It is used to indicate the position of c:

```
\clef "kievan-do"
\kievanOn
c'
```



See also

Music Glossary: Section "Kievan notation" in *Music Glossary*, Section "clef" in *Music Glossary*.
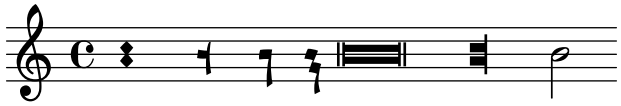
Notation Reference: Section 1.3.1 [Clef], page 19.

17.5.3 Kievan notes

For Kievan square notation, the appropriate note head style needs to be chosen and the flags and stems need to be turned off. This is accomplished by calling the `\kievanOn` function, which sets the appropriate properties of the note head, stems, and flags. Once Kievan note heads are not needed, these properties can be reverted by calling the `\kievanOff` function.

The Kievan final note, which usually comes at the end of a piece of music, may be selected by setting the duration to `\longa`. The Kievan recitative mark, used to indicate the chanting of several syllables on one note, may be selected by setting the duration to `\breve`. The following example demonstrates the various Kievan note heads:

```
\cadenzaOn
\kievanOn
b'1 b'2 b'4 b'8 b'\breve b'\longa
\kievanOff
b'2
```



See also

Music Glossary: Section "Kievan notation" in *Music Glossary*, Section "note head" in *Music Glossary*.

Notation Reference: Section B.9 [Note head styles], page 922.
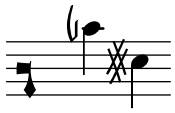
Known issues and warnings

LilyPond automatically determines if the stem up or stem down form of a note is drawn. When setting chant in square notation, however, it is customary to have the stems point in the same direction within a single melisma. This can be done manually by setting the `direction` property of the `Stem` object.

17.5.4 Kievan accidentals

The `kievan` style provides a sharp and a flat sign different from the default style. There is no natural sign in Kievan notation. The sharp sign is not used in Synodal music but may occur in earlier manuscripts. It has been included primarily for the sake of compatibility.

```
\clef "kievan-do"
\set Staff.alterationGlyphs =
  #alteration-kievan-glyph-name-alist
bes' dis'
```



See also
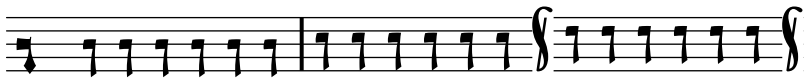
Music Glossary: Section “Kievan notation” in *Music Glossary*, Section “accidental” in *Music Glossary*.

Notation Reference: Section 1.1.3 [Accidentals], page 8, Section 1.3.5 [Automatic accidentals], page 31, Section 1.3.6 [Alternate accidental glyphs], page 39, Section B.8 [The Emmentaler font], page 907.

17.5.5 Kievan bar lines

In a `KievanStaff`, there are no measures, but the `\caesura` command creates a phrase bar line, and the `\section` and `\fine` commands create a special section bar line.

```
\new KievanStaff {
  c'4 4 4 4 4 4 \caesura  % \bar "."
  d'4 4 4 4 4 4 \section  % \bar "k"
  e'4 4 4 4 4 4 \fine
}
```



See also

Notation Reference: Section 2.5 [Bars], page 121, Section B.8 [The Emmentaler font], page 907.

17.5.6 Kievan melismata

Notes within a Kievan melisma are usually placed close to each other and the melismata separated by whitespace. This is done to allow the chanter to quickly identify the melodic structures of *Znamenny chant*. In LilyPond, melismata are treated as ligatures and the spacing is implemented by the `Kievan_ligature_engraver`.

When the `KievanVoice` and `KievanStaff` contexts are used, the `Kievan_ligature_engraver` is enabled by default. In other contexts, it can be invoked by replacing the `Ligature_bracket_engraver` with the `Kievan_ligature_engraver` in the layout block:
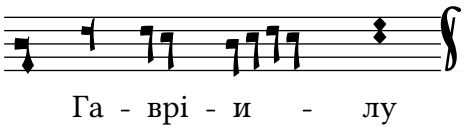
```
\layout {
  \context {
    \Voice
    \remove Ligature_bracket_engraver
    \consists Kievan_ligature_engraver
  }
}
```

The spacing between the notes within a Kievan ligature can be controlled by setting the `padding` property of the `KievanLigature`.

The following example demonstrates the use of Kievan ligatures:

```
% Font settings for Cyrillic
\paper {
  property-defaults.fonts.serif = "Linux Libertine O,serif"
}

\score {
  <<
    \new KievanVoice = "melody" \relative c' {
      e2 \[ e4( d4 ) \] \[ c4( d e d) \] e1 \fine
    }
    \new Lyrics \lyricsto "melody" {
      Га -- врі -- и -- лу
    }
  >>
}
```



See also

Music Glossary: Section “ligature” in *Music Glossary*.

Notation Reference: Section 17.3.10 [White mensural ligatures], page 549, Section 17.4.7 [Gregorian square neume ligatures], page 558, Section 17.2.2 [Ligatures], page 540.

Known issues and warnings

Horizontal spacing of ligatures is poor.

17.6 Working with ancient music – scenarios and solutions
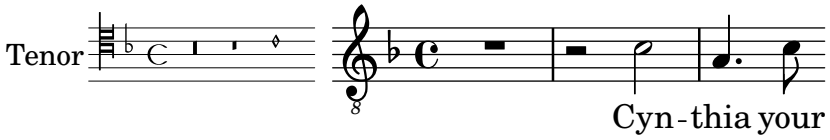
Working with ancient music frequently involves particular tasks which differ considerably from the modern notation for which LilyPond is designed. In the rest of this section, a number of typical scenarios are outlined, with suggestions of solutions. These involve:

- 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.

17.6.1 Incipits

It is customary when transcribing mensural music into modern notation to place an indication of how the initial rests and note or notes of the original version appeared – including the original clefs. This is called an *incipit*. The `\incipit` command uses the `indent` of the main staff to set the width occupied by the incipit together with the instrument name, and `incipit-width` to set the width of the incipit staff. An optional argument indicates how much the incipit should be shifted to the left.

```
\score {
  \new Staff <<
    \new Voice = Tenor {
      \set Staff.instrumentName = "Tenor"
      \override Staff.InstrumentName.self-alignment-X = #RIGHT
      \incipit #1 { \clef "mensural-c4" \key f \major r\breve r1 c'1 }
      \clef "treble_8"
      \key f \major
      R1 r2 c'2 |
      a4. c'8
    }
    \new Lyrics \lyricsto Tenor { Cyn -- thia your }
  >>
  \layout
  {
    indent = 5\cm
    incipit-width = 3\cm
  }
}
```



By default, LilyPond uses a 'MensuralStaff' context for typesetting an incipit. Other contexts can be used by directly writing `\incipit \new contexttype ...`; in this case a 'MensuralStaff' wrapper is only used when it can contain the specified context.

Known issues and warnings

Note that instrumentName must be set in the music for the incipit to be produced. If no instrument name is required then use `\set Staff.instrumentName = ""`.

17.6.2 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.

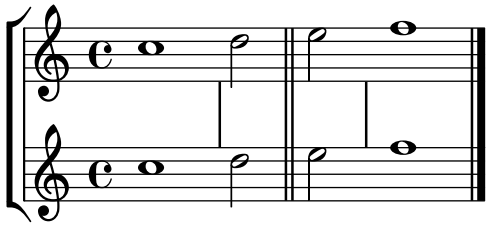
Mensurstriche, bar lines between but not through staves, can be printed by setting `measureBarType` to `"-span|"` and using a grouping context that allows span bars, such as `StaffGroup`.

```
\layout {
  \context {
    \Staff
    measureBarType = "-span|"
  }
}

music = \fixed c'' {
  c1
  d2 \section e2
  f1 \fine
```

```
}

\new StaffGroup <<
  \new Staff \music
  \new Staff \music
>>
```



17.6.3 Transcribing Gregorian chant

Gregorian chant can be transcribed into modern notation with a number of simple tweaks.

Stems. The `GregorianTranscriptionVoice` context does not create stems. You can extend this behavior to other contexts by removing `Stem_engraver`:

```
\layout {
  ...
  \context {
    \Voice
    \remove Stem_engraver
  }
}
```

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. An alternative is to use `\cadenzaOn` and `\cadenzaOff`.

The predefined staff contexts for ancient music do not create measure bar lines. You can extend this behavior to all other contexts with `\set Score.measureBarType = #'()` or to particular staves with `\set Staff.measureBarType = #'()`.

The predefined staff contexts for ancient music allow line breaks without bar lines. You can extend this behavior to all other contexts with `\set Score.forbidBreakBetweenBarLines = ##f` or to particular staves with `\set Staff.forbidBreakBetweenBarLines = ##f`.

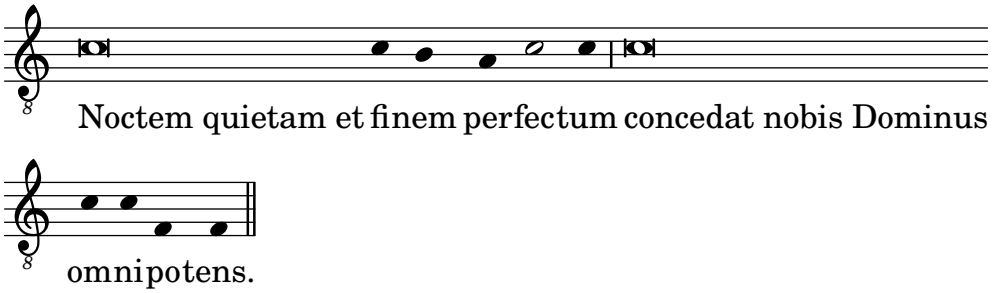
A common type of transcription is recitativic chant where the repeated notes are indicated with a single breve. The text to the recitation tone can be dealt with in two different ways: either set as a single, left-aligned syllable:

```
chant = \relative {
  \clef "G_8"
  c'\breve c4 b4 a c2 c4 \divisioMaior
  c\breve c4 c f, f \finalis
}

words = \lyricmode {
  \tweak self-alignment-X #LEFT "Noctem quietam et"
    fi -- nem per -- fec -- tum
  \tweak self-alignment-X #LEFT "concedat nobis Dominus"
```

```
    om -- ni -- po -- tens.
}

\score {
  \new GregorianTranscriptionStaff <<
    \new GregorianTranscriptionVoice = "melody" {
      \chant
    }
    \new GregorianTranscriptionLyrics = "one" {
      \lyricsto "melody" \words
    }
  >>
}
```



This works fine, as long as the text doesn't span a line break. If that is the case, an alternative is to add hidden notes to the score, as below.
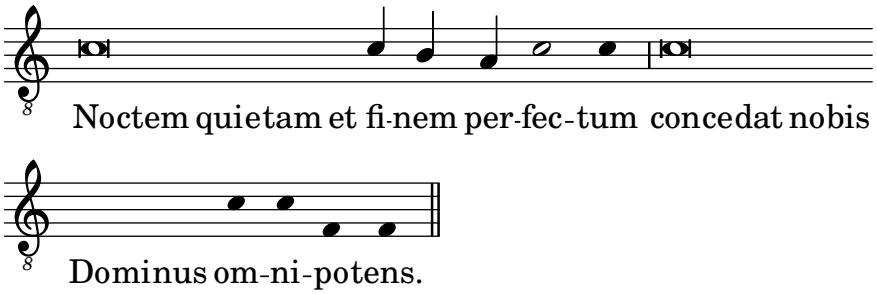
In some transcription styles, stems are used occasionally, for example to indicate the transition from a single-tone recitative to a fixed melodic gesture. In these cases, one can use the `Stem_engraver` and manually `\omit Stem` and `\undo \omit Stem`.

```
chant = \relative {
  \clef "G_8"
  \omit Stem
  \omit Flag
  c'\breve*1/16 \hide NoteHead c8 c c c c
  \undo \hide NoteHead
  \undo \omit Stem \stemUp c4 b4 a
  \omit Stem c2 c4 \divisioMaior
  c\breve*1/16 \hide NoteHead c8 c c c c c c
  \undo \hide NoteHead 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 GregorianTranscriptionStaff <<
    \new GregorianTranscriptionVoice = "melody" {
      \chant
    }
    \new GregorianTranscriptionLyrics = "one" {
      \lyricsto "melody" \verba
```

```
    }
  >>
  \layout {
    \context {
      \GregorianTranscriptionVoice
      \consists Stem_engraver
    }
  }
}
```
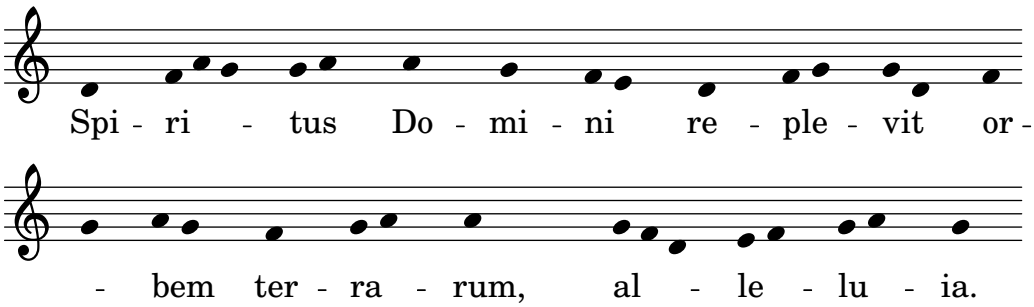


Another common situation is transcription of neumatic or melismatic chants, i.e., chants with a varying number of notes to each syllable. In this case, one would want to set the syllable groups clearly apart, usually also the subdivisions of a longer melisma. One way to achieve this is to use a fixed `\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 bar lines and all other rhythmical indications are made transparent, and the space around the bar lines 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 {
  \time 1/4
  d'4 \tuplet 3/2 { f8 a g } g a a4 g f8 e
  d4 f8 g g8 d f g a g f4 g8 a a4 s
  \tuplet 3/2 { g8 f d } e f g a g4
}

spirLyr = \lyricmode {
  \override Lyrics.LyricText.X-extent  = #'(0 . 3)
  Spi -- ri -- _ _ tus _ Do -- mi -- ni _
  re -- ple -- _ vit _ or -- _ bem _ ter -- ra -- _ rum,
  al -- _ _ le -- _ lu -- _ ia.
}
\score {
  \new GregorianTranscriptionStaff <<
    \new GregorianTranscriptionVoice = "chant" {
      \spiritus
    }
    \new GregorianTranscriptionLyrics = "one" {
      \lyricsto "chant" \spirLyr
    }
  >>
```

```
  \layout {
    \context {
      \GregorianTranscriptionStaff
      measureBarType = ""
      \override BarLine.X-extent = #'(-1 . 1)
      \hide TupletNumber
      \hide TupletBracket
    }
  }
}
```



17.6.4 Ancient and modern from one source

Using tags to produce mensural and modern music from the same source

Using tags it is possible to produce both mensural and modern notation from the same music. In this snippet, a function `\menrest` is introduced, allowing mensural rests to be pitched as in the original, but with modern rests in the standard staff position.

Tags can also be used where other differences are needed: for example using "whole measure rests" (`R1`, `R\breve`, etc.) in modern music, but normal rests (`r1`, `r\breve`, etc.) in the mensural version. Converting mensural music to its modern equivalent is usually referred to as *transcription*.

The call `c4.\Be c8 c\Am` is the same as `c4.[ c8 c]`. However, it suppresses warnings if it starts on a note that can't hold a beam but needs it anyway due to the use of `Completion_heads_engraver`.

[Note that the custos sticks out into the right margin and might be cut off if the LilyPond output gets cropped tightly. The use of `\with-true-dimensions` below avoids this.]

```
\layout {
  line-width = 150\mm
}

menrest = #(define-music-function (note) (ly:music?)
              #{
                \tag #'mens $(make-music 'RestEvent note)
                \tag #'mod $(make-music 'RestEvent note 'pitch '())
              #})
Be = \tag #'mod
       #(begin
          (ly:expect-warning (G_ "stem does not fit in beam"))
          (ly:expect-warning (G_ "beam was started here"))
          (make-span-event 'BeamEvent START))
Am = \tag #'mod ]
```

```
MenStyle = {
  \override Score.BarNumber.transparent = ##t
  \override Stem.neutral-direction = #up
  \omit Slur
  \omit Beam
}

finalis = \section

Music = \relative c'' {
  \key f \major
  g1 d'2 \menrest bes4 bes a2 \menrest r4 g4 fis4. fis8 fis4 fis \break
  g e f4.([ g8] a4[ g8 f] g2.\Be fis8 e\Am fis2) g\breve \finalis
}

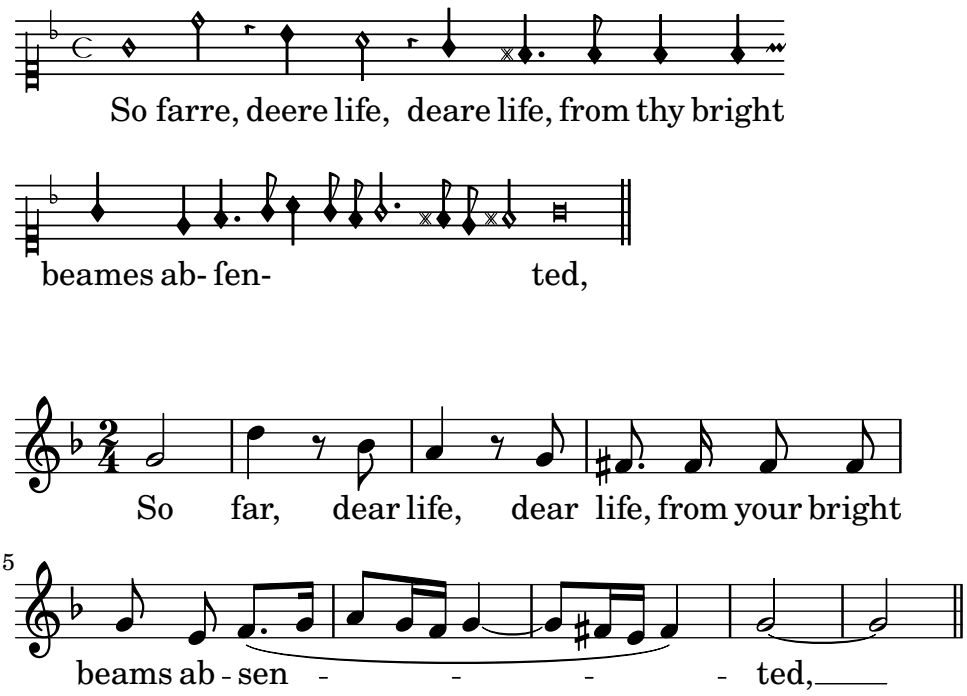
MenLyr = \lyricmode {
  So farre, deere life, deare life,
  from thy bright beames ab- en- ted,
}
ModLyr = \lyricmode {
  So far, dear life, dear life,
  from your bright beams ab -- sen -- ted, __
}

\score {
  \keepWithTag #'mens {
    <<
      \new PetrucciStaff {
        \new PetrucciVoice = "Cantus" {
          \clef "petrucci-c1" \time 4/4 \MenStyle \Music
        }
      }
      \new Lyrics \lyricsto "Cantus" \MenLyr
    >>
  }
  \layout {
    \context {
      \PetrucciVoice
      % No longer necessary starting with version 2.25.23.
      \override Flag.style = #'mensural
    }
  }
}

\markup\vspace #1

\score {
  \keepWithTag #'mod {
    \new ChoirStaff <<
      \new Staff {
        \new Voice = "Sop" \with {
```

```
            \remove "Note_heads_engraver"
            \consists "Completion_heads_engraver"
            \remove "Rest_engraver"
            \consists "Completion_rest_engraver"
          } \shiftDurations 1 0 { \time 2/4 \autoBeamOff \Music }
        }
        \new Lyrics \lyricsto "Sop" \ModLyr
      >>
    }
}
```



18 World music

The purpose of this section is to highlight musical notation issues that are relevant to traditions outside the Western tradition.

18.1 Common notation for non-Western music

This section discusses how to enter and print music scores that do not belong to the Western classical tradition, also referred to as *Common Practice Period*.

18.1.1 Extending notation and tuning systems

Standard classical notation (also known as *Common Practice Period* notation) is commonly used in all sorts of music, not limited to 'classical' Western music. This notation is discussed in Section 1.1 [Writing pitches], page 3, and the various note names that may be used are explained in Section 1.1.5 [Note names in other languages], page 10.

Some types of non-Western music and folk/traditional music often employ alternative or extended tuning systems that do not fit easily into standard, classical notation.

Standard notation is still used but with pitch differences being implicit. For example, *Arabic music* is notated with semi and quarter tone accidentals but with precise pitch alterations being determined by context. In the case of *Arabic music*, the init file `arabic.ly` provides a suitable set of macros and definitions that extend the standard notation using Italian note names. For more details see Section 18.2 [Arabic music], page 575.

Other types of music require extended or unique notations, for example, *Turkish classical music* (also known as Ottoman classical music) employs melodic forms known as *makamlar* where intervals are based on 1/9 divisions of the whole tone. Standard, Western staff notes are still used, but with special accidentals uniquely defined in the files `turkish-makam.ly`. For more information on Turkish classical music and makamlar see Section 18.3 [Turkish classical music], page 581.

Other, related init files are also available; `hel-arabic.ly` and `makam.ly`.

To locate these init files on your system, see Section "Other sources of information" in *Learning Manual*.

See also

Music Glossary: Section "Common Practice Period" in *Music Glossary*, Section "makamlar" in *Music Glossary*.

Learning Manual: Section "Other sources of information" in *Learning Manual*.

Notation Reference: Section 1.1 [Writing pitches], page 3, Section 1.1.5 [Note names in other languages], page 10, Section 18.2 [Arabic music], page 575, Section 18.3 [Turkish classical music], page 581, Section 18.4 [Persian classical music], page 583.

18.2 Arabic music

This section highlights issues that are relevant to notating Arabic music.

18.2.1 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 Section 18.1 [Common notation for non-Western music], page 575.
- Additional key signatures can also be tailored as described in Section 1.3.2 [Key signature], page 24.
- Complex time signatures may require that notes be grouped manually as described in Section 2.4.3 [Manual beams], page 115.
- *Takasim* which are rhythmically free improvisations may be written down omitting bar lines as described in Section 2.3.5 [Unmetered music], page 93.

See also

Notation Reference: Section 18.1 [Common notation for non-Western music], page 575, Section 1.3.2 [Key signature], page 24, Section 2.4.3 [Manual beams], page 115.
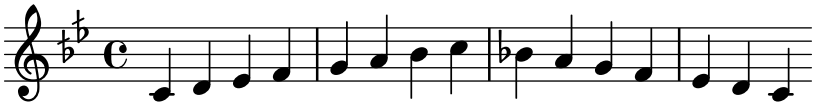
Snippets: Section "World music" in *Snippets*.

18.2.2 Arabic note names

Traditional Arabic note names (like 'rast', 'dukah', 'sikah', etc.) can be quite long and so may not always be suitable for the purpose of music writing.
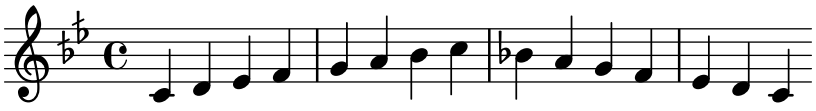
Include the file `arabic.ly` to write Arabic sheet music. The following example demonstrates how to write a 'rast' scale:

```
\include "arabic.ly"
\relative {
  \key do \rast
  do' re misb fa | sol la sisb do | sib la sol fa | misb re do
}
```



The file `arabic.ly` sets the note language to Italian (or Solfege), since that is the modern standard in Arabic music and is widely adopted among Arabic musicians. If you prefer to write sheet music in another language, simply change the language to your preferred language directly after including the file `arabic.ly`. This is a 'rast' scale with English note names:

```
\include "arabic.ly"
\language "english"
\relative {
  \key c \rast
  c' d eqf f | g a bqf c | bf a g f | eqf d c
}
```
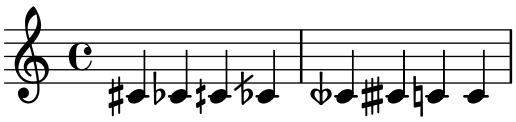


"Rast" is a heptatonic scale that uses quarter tone intervals and is considered the most important and central scale of the "Arabic Maqamat". For the full list of supported Arabic scales, see Section 18.2.3 [Arabic key signatures], page 577.

The use of standard Western notation to notate non-Western music is discussed in Section 18.1 [Common notation for non-Western music], page 575. Also see Section 1.1.5 [Note names in other languages], page 10.

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 {
  \set Staff.extraNatural = ##f
  dod' dob dosd \dwn dob dobsb dodsd do do
}
```



See also

Notation Reference: Section 1.1.5 [Note names in other languages], page 10, Section 18.1 [Common notation for non-Western music], page 575, Section 22.1 [Including LilyPond files], page 628.

Installed Files: `ly/arabic.ly`

Snippets: Section "World music" in *Snippets*.

18.2.3 Arabic key signatures

In addition to the minor and major key signatures, LilyPond provides the most common Arabic key signatures in the file `arabic.ly`. With that being said LilyPond is not aiming at providing a full suite of all possible maqams. It rather defines the most common ones that are frequently used and offers key signatures by grouping maqams together. In general, a maqam uses the key signature of its family, or a neighboring family, and varying accidentals are marked throughout the music. When forming key signatures neighboring maqam families are grouped together. For example, maqam *saba* seldom occurs outside of the context of maqam *bayati* and adds only a single alteration. Although both maqams come from different *maqam families* they are inside the same key signature *group*. Arabic maqams only allow for limited modulations, due to the nature of Arabic musical instruments.

Here is an example of the key signature for a "maqam muhayer" piece of music:

```
\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 also appear in the title.

Other maqams in the same *bayati* group, as shown in the table below (e.g., *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 do not change their fundamental nature, as siblings.

The other maqam in the same group (*nawa*) is related to *bayati* by modulation and is shown in the table in parentheses for those that are modulations of their base maqam. *Nawa*, for example, can be indicated as follows:

```
\key sol \bayati
```

In Arabic music, the same term, for example *bayati*, that is used to indicate a maqam family, is also a maqam that is usually the most important in the family so can also be thought of as a *base maqam*.

Here is the grouping that maps the more common maqams to key signatures as defined in the file `arabic.ly`:
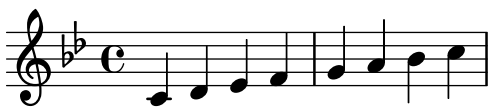
maqam group	**key**	**finalis**	**Other maqams in group (finalis)**
ajam	`major`	sib	jaharka (fa)
bayati	`bayati`	re	hussaini, muhayer, saba, ushaq, nawa (sol)
hijaz	`hijaz`	re	zanjaran (do)
hijaz kar	`hijaz_kar`	do	shahnaz, shad arban (sol)
huzam	`huzam`	misb	-
iraq	`iraq`	sisb	-
kurd	`kurd`	re	hijazkar kurd (do)
nahawand	`minor`	do	busalik (re), farah faza (sol)
nakriz	`nakriz`	do	nawa athar, hisar (re)
rast	`rast`	do	mahur, yakah (sol)
sikah	`sikah`	misb	-

In case you are missing a specific maqam, you can define it yourself in your sheet music before using it. The following example defines and then uses the *zanjaran* maqam.

```
\include "arabic.ly"

% For example on do: do reb mi fa sol la sib do
% reb and sib are FLAT
% You can also use SHARP, SEMI-FLAT, SEMI-SHARP
zanjaran = #`(
  (0 . ,NATURAL)
  (1 . ,FLAT)
  (2 . ,NATURAL)
  (3 . ,NATURAL)
  (4 . ,NATURAL)
  (5 . ,NATURAL)
  (6 . ,FLAT)
)

\relative {
  \key do \zanjaran
  do' reb mi fa sol la sib do
}
```



For special cases, rare maqams are defined in the `hel-arabic.ly` file. Please refer to the file included with LilyPond for a full list of the provided maqams.

Selected snippets

Non-traditional key signatures

The commonly used `\key` command sets the `keyAlterations` 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.keyAlterations =
  #`(((octave . step) . alter) ((octave . step) . alter) ...)
```
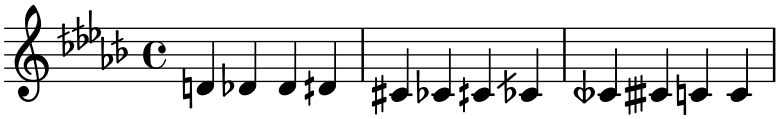
where, for each element in the list, *octave* specifies the octave (0 being the octave from middle C to the B above), *step* specifies the note within the octave (0 means C and 6 means B), and *alter* is one of `SHARP`, `FLAT`, `DOUBLE-SHARP`, etc., preceded by a comma.

Alternatively, you can use the more concise format `(step . alter)` for each item in the list if the same alterations are used in all octaves.

For microtonal scales where a "sharp" is not 100 cents, *alter* refers to the alteration as a proportion of a 200-cent whole tone.

```
\include "arabic.ly"

\relative do' {
  \set Staff.keyAlterations = #`((0 . ,SEMI-FLAT)
                                 (1 . ,SEMI-FLAT)
                                 (2 . ,FLAT)
                                 (5 . ,FLAT)
                                 (6 . ,SEMI-FLAT))
  % \set Staff.extraNatural = ##f
  re reb \dwn reb resd
  dod dob dosd \dwn dob |
  dobsb dodsd do do |
}
```



See also

Music Glossary: Section "maqam" in *Music Glossary*, Section "bayati" in *Music Glossary*, Section "rast" in *Music Glossary*, Section "sikah" in *Music Glossary*, Section "iraq" in *Music Glossary*, Section "kurd" in *Music Glossary*.

Learning Manual: Section "Pitches and key signatures" in *Learning Manual*.

Notation Reference: Section 1.3.2 [Key signature], page 24.

Installed Files: `ly/arabic.ly` `ly/hel-arabic.ly`

Snippets: Section "World music" in *Snippets*, Section "Pitches" in *Snippets*.

Internals Reference: Section "KeySignature" in *Internals Reference*.

18.2.4 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. The alternative is to switch off automatic beaming and beam the notes manually. Even if a match to existing typeset music is not required, it may still be desirable to adjust the automatic beaming behavior and/or use complex time signatures.

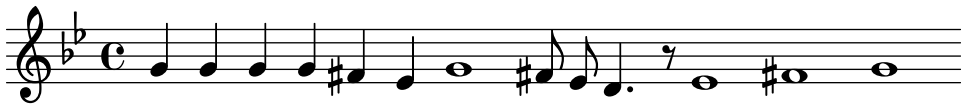
Selected snippets

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 causes 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
  \accidentalStyle forget
  \cadenzaOn
  sol4 sol sol sol fad mib sol1 fad8 mib re4. r8 mib1 fad sol
}
```



See also

Music Glossary: Section "semai" in *Music Glossary*, Section "taqasim" in *Music Glossary*.

Notation Reference: Section 2.4.3 [Manual beams], page 115, Section 2.4.1 [Automatic beams], page 102, Section 2.3.5 [Unmetered music], page 93, Section 1.3.5 [Automatic accidentals], page 31, Section 2.4.2 [Setting automatic beam behavior], page 106, Section 2.3.2 [Time signature], page 80.

Installed Files: `ly/arabic.ly`

Snippets: Section "World music" in *Snippets*.
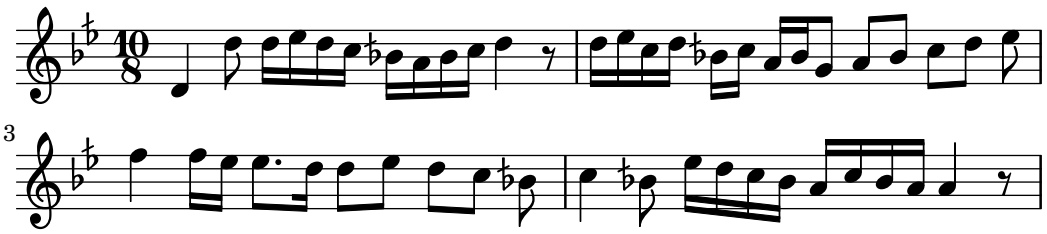
18.2.5 Arabic music example

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 {
  \header {
    title = "Semai Muhayer"
    composer = "Jamil Bek"
  }
  \relative {
    \set Staff.extraNatural = ##f
    \set Staff.autoBeaming = ##f
    \key re \bayati
    \time 10/8

    re'4 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
  }
```

```
}
```



See also

Installed Files: `ly/arabic.ly`

Snippets: Section "World music" in *Snippets*.

18.2.6 Further reading for Arabic music

There are some variations in the details of how maqams are grouped, despite agreement of grouping maqams related through common lower tetra chords or by modulation. There are also some inconsistencies, even within the same texts, on how key signatures for a particular maqam should be specified. However, it is common to use a key signature per 'group' of maqams instead of individual key signatures for each maqam separately.

- *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 some web sites that explain maqams and even provide audio examples:
  - `https://www.maqamworld.com/`
  - `https://www.turath.org/`
- 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

18.3 Turkish classical music

This section highlights issues that are relevant to notating Turkish classical music.

18.3.1 References for Turkish classical music

Turkish classical music developed in the Ottoman Empire at roughly the same time as 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*) are constructed.

Some issues relevant to Turkish classical music are covered elsewhere. Special note names and accidentals are explained in Section 18.1 [Common notation for non-Western music], page 575.

18.3.2 Turkish note names

Pitches in Turkish classical music traditionally have unique names and the basis of pitch on 1/9-tone divisions means that makamlar employ a completely different set of intervals compared to Western scales and modes:

From a modern, notational point of view it is convenient to use standard, Western staff notes (c, d, e, etc.) but with custom accidentals that raise or lower notes by intervals of 1/9, 4/9, 5/9 or 8/9 of a tone.

These custom accidentals are defined in the file `turkish-makam.ly`.

For a more general explanation of non-Western music notation, see Section 18.1 [Common notation for non-Western music], page 575.

See also

Music Glossary: Section “makam” in *Music Glossary*, Section “makamlar” in *Music Glossary*.

Notation Reference: Section 18.1 [Common notation for non-Western music], page 575.

18.3.3 Turkish key signatures

LilyPond supports over 200 makam key signature definitions – well beyond what is used in Turkish classical music – with each makam having its own specific tonic / finalis pitch (known as ‘karar’ in Turkish).

It is important to be aware of the finalis of each makam. Here is an example where *g* is the default tonic and *rast* is the name of the makam.

```
\key g \rast
```

The correct accidentals, koma flat (*b1*) and koma sharp (`f4`), (both in relation to the tonic *g*), will be displayed automatically.

Selected snippets

Turkish Makam example

This template uses the start of a well-known Turkish *Saz Semai* that is familiar in the repertoire in order to illustrate some of the elements of Turkish music notation.

```
#(set-default-paper-size "a6" 'landscape)

\include "turkish-makam.ly"
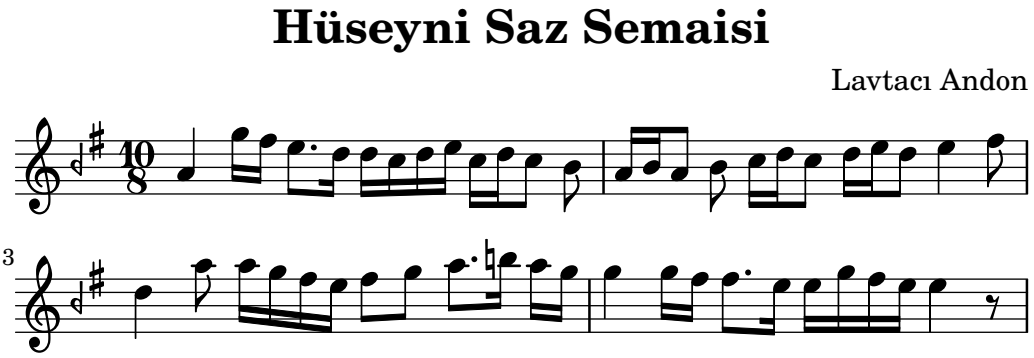
\header {
  title = "Hüseyni Saz Semaisi"
  composer = "Lavtacı Andon"
  tagline = ##f
}

\relative {
  \set Staff.extraNatural = ##f
  \set Staff.autoBeaming = ##f

  \key a \huseyni
  \time 10/8

  a'4 g'16[ fb] e8.[ d16] d[ c d e] c[ d c8] bfc |
  a16[ bfc a8] bfc c16[ d c8] d16[ e d8] e4 fb8 |
  d4 a'8 a16[ g fb e] fb8[ g] a8.[ b16] a16[ g] |
  g4 g16[ fb] fb8.[ e16] e[ g fb e] e4 r8 |
}

\layout {
  indent = 0
}
```



18.3.4 Further reading for Turkish music

- *Turk Musikisi Nazariyati ve Usulleri: Kudum Velveleleri* by Ismail Hakki Ozkan [(Kultur serisi, 41) (Turkish) Paperback – 1986]
  contains information about the theory of makams and usul.
- *Music of the Ottoman Court* by Walter Feldman [VWB Hardback – 1996]
  contains information about the history of Ottoman court music.
- *Turkish Music Makam Guide* by Murat Aydemir [Pan Paperback – 2010]
  contains information in English regarding Turkish makam including two CDs.

18.4 Persian classical music

This section highlights issues that are relevant to notating Persian classical music.

18.4.1 Persian music notation

The notation for Persian classical music commonly uses two accidentals for microtones, *sori* and *koron*. Invented by Ali-Naqi Vaziri around 1935, they indicate raising and lowering a pitch by (approximately) a quarter tone, respectively.



The file `persian.ly`[1] provides support for koron and sori; they can be obtained by appending 'k' (koron) and 'o' (sori) to the English note symbols.

LilyPond supports tunings for all major Persian modes in all keys, sufficient to notate the *gushehs* (central nuclear melodies) of all *dastgahs* (musical modal systems).

The note immediately following a koron is sometimes[2] lowered by about 20 cents. This is not notated but considered part of the tuning. However, for getting better MIDI support you can make a sound flat by appending 'v' to the note name ('*v*lat'). This note should actually also get a strong vibrato, and the vibrato and low tuning are perceptually integrated (*serialism*). This is just for MIDI and has no effect on the notation itself.

There are no further tuning issues in Persian music. Since the music is monophonic, the difference between just intonation (for example) and equal temperament is merely academic – there are no chords where out-of-tune intervals would be noticeable.

The following suffixes to English note names are provided.

[1] There exists another, older support file for Persian classical music also called `persian.ly` (written by Kees van den Doel) that no longer works with the current LilyPond version; while note names are compatible, the selection of key signatures is not.

[2] If the interval defined by the note before the koron and after the koron is a minor third. The same is true for the note below the finalis in the 'Esfahan' dastgah according to some (but not all) Persian musicians.

`ff`	double-flat
`f`	flat
`k`	koron (about quarter flat, -3/10 of whole tone, 60 cents)
`o`	sori (about quarter sharp, 2/10 of whole tone, 40 cents)
`s`	sharp
`x`	double-sharp
`v`	20 cent flat-tuned note ("vlat", not notated)
`fv`	flat, tuned 20 cents down (notated as a normal flat)
`sv`	sharp, tuned 20 cents down (notated as a normal sharp; does never occur in traditional Persian music)

18.4.2 Persian tunings

Dastgahs can have many tunings; it has become common to name tunings after representative dastgahs featuring them. The following Persian tunings are provided to be used as the second argument of the `\key` command. Note that in practice not all tunings are used in all keys.

`\shur`	'Shur' gushehs with natural 5th degree
`\shurk`	'Shur' gushehs with koron 5th degree
`\esfahan`	
`\mokhalefsegah`	
`\chahargah`	
`\mahur`	the same as `\major` but used in a completely different context
`\delkashMahur`	
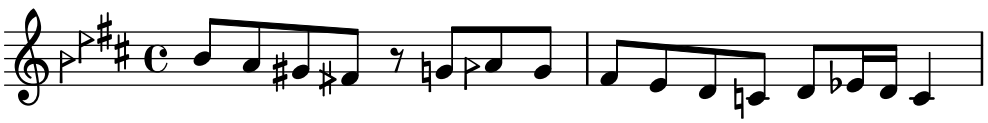
In dastgah 'Shur', the 5th degree can either be natural or koron, so two tunings are provided (`\shur`, `\shurk`). Secondary dastgahs 'Dashti', 'Abuata', and 'Bayat-e-tork' use `\shur`, the derived dastgah 'Afshari' uses `\shurk`. 'Nava' on G is like 'Shur' on D (a 4th apart). The tuning of 'Segah' is `\shurk`. Other tunings are `\esfahan` ('Homayoun' on C is like 'Esfahan' on G, a 5th apart), `\chahargah`, and `\mokhalefsegah`. For dastgahs 'Mahur' and 'Rast-panjgah' the same tuning `\mahur` can be used.

All the various modulatory gushehs in all dastgahs can be notated with these tunings, with the exception of 'Delkash' in 'Mahur', but traditionally the peculiar accidentals there are notated explicitly (however, `persian.ly` also defines a tuning `\delkashMahur`).

As an example, Chahargah in D can be input with the following.

```
\include "persian.ly"

\relative c' {
  \key d \chahargah
  bk'8 a gs fo r g ak g |
  fs ek d c d ef16 d c4 |
}
```
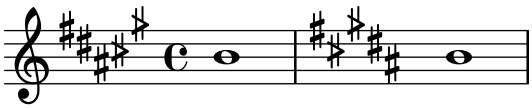


18.4.3 Persian key signatures

By default, the order of accidentals in Persian key signatures is flats followed by korons, then sharps followed by soris (then double flats, then double sharps).

An alternative key order, similar to the one used in `turkish-makam.ly`, can be selected. Here, the order is flats or korons (or double flats), then sharps or soris (or double sharps). In other words, korons and soris are handled equal to flats and sharps, respectively.

```
\include "persian.ly"

{
  \key b \chahargah b'1 |
  \set Staff.keyAlterationOrder = \persianAltKeyAlterationOrder
  \key b \chahargah b'1 |
}
```



18.4.4 Further reading on Persian music

- *Traditional Persian Art Music* by Dariush Tala'i [Bibliotheca Persica, Costa Mesa CA, 2000]

  The provided Persian tunings closely follow this book.
- *The Dastgah Concept in Persian Music* by Hormoz Farhat [Cambridge University Press, Cambridge, 1990]
- *Le répertoire-modèle de la musique iranienne* by Jean During (in French) [Sourush, Tehran, 1995]

  This book contains measurements of the intervals in actual practice, consistent with the tunings of `persian.ly`.
- *Armoni-e Musiqi-e Iran* by Ali-Naqi Vaziri (in Persian) [1935]
- *Scales and Modes Around the World* by Herman Rechberger [Fennica Gehrman, 2018, ISBN 978-952-5489-07-1]

General input and output

19 Input modes

The way in which the notation contained within an input file is interpreted is determined by the current input mode. In general, there are two ways of specifying the mode: a long form, e.g. `\chordmode`, and a short form, e.g. `\chords`. The long form is typically used when supplying input to a variable or when entering input directly into an explicitly created context. The short form implicitly creates a context of the correct type for the input and passes the input directly to it. It is useful in simple situations when there is no requirement to explicitly create the receiving context.

Chord mode

This is activated with the `\chordmode` command, and causes input to be interpreted with the syntax of chord notation, see Chapter 15 [Chord notation], page 508. Music in chord mode is rendered as chords on a staff when entered into a `Staff` context, as chord names when entered into a `ChordNames` context or as fretboards when entered into a `FretBoards` context.

Chord mode is also activated with the `\chords` command. This also causes the following input to be interpreted with the syntax of chord notation but in addition it implicitly creates a new `ChordNames` context and renders the input into it as chord names, see Section 15.2.1 [Printing chord names], page 513.

Drum mode

This is activated with the `\drummode` command, and causes input to be interpreted with the syntax of drum notation, see Section 13.1.2 [Basic percussion notation], page 487. Music in drum mode is rendered as percussion notes when entered into a `DrumStaff` context.

Drum mode is also activated with the `\drums` command. This also causes the following input to be interpreted with the syntax of drum notation but in addition it implicitly creates a new `DrumStaff` context and renders the input into it as percussion notes, see Section 13.1.2 [Basic percussion notation], page 487.

Figure mode

This is activated with the `\figuremode` command, and causes input to be interpreted with the syntax of figured bass, see Section 15.3.2 [Entering figured bass], page 527. Music in figure mode is rendered as figured bass when entered into a `FiguredBass` context or a `Staff` context.

Figure mode is also activated with the `\figures` command. This also causes the following input to be interpreted with the figured bass syntax but in addition it implicitly creates a new `FiguredBass` context and renders the input into it as figured bass, see Section 15.3.1 [Introduction to figured bass], page 526.

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 Section 12.1.3 [Default tablatures], page 433.

To create fret diagrams above a staff, enter notes or chords in either note mode or chord mode and render them in a `FretBoards` context, see Section 12.1.7 [Automatic fret diagrams], page 477. Alternatively, fret diagrams can be entered as markup above the notes using the `\fret-diagram` command, see Section 12.1.5 [Fret diagram markups], page 455.

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 Chapter 9 [Vocal music], page 345. Input in lyric mode is rendered as lyric syllables when entered into a `Lyrics` context.

Lyric mode is also activated with the `\lyrics` command. This also causes the following input to be interpreted as lyric syllables but in addition it implicitly creates a new `Lyrics` context and renders the input into it as lyric syllables.

Lyric mode is also activated with the `\addlyrics` command. This also implicitly creates a new `Lyrics` context and in addition it adds an implicit `\lyricsto` command which associates the following lyrics with the preceding music, see Section 9.1.4 [Automatic syllable durations], page 349.

Markup mode

This is activated with the `\markup` command, and causes input to be interpreted with the syntax of markup, see Section A.1 [Text markup commands], page 809.

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.

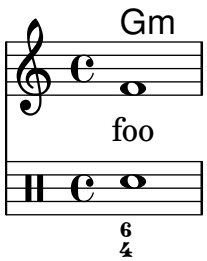
It is not normally necessary to specify note mode explicitly, but it may be useful to do so in certain situations, for example if you are in lyric mode, chord mode or any other mode and want to insert something that only can be done with note mode syntax.

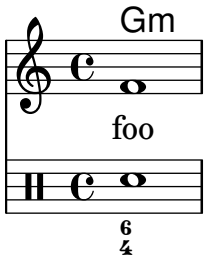
```
% This ...

<<
  \chords { g1:m }
  { f'1 }
  \lyrics { foo1 }
  \drums { sn1 }
  \figures { <6 4>1 }
>>

% ... is equivalent to

<<
  \new ChordNames \chordmode { g1:m }
  \new Voice \notemode { f'1 }
  \new Lyrics \lyricmode { foo1 }
  \new DrumStaff \drummode { sn1 }
  \new FiguredBass \figuremode { <6 4>1 }
>>
```




Gm
foo

20 Input structure

The main format of input for LilyPond are text files. By convention, these files end with `.ly`.

20.1 Structure of a score

A `\score` block must contain a single music expression delimited by curly brackets.[1]
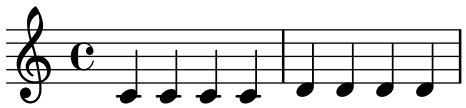
```
\score {
  ...
}
```

> **Note:** There must be **only one** outer music expression in a `\score` block, and it **must** be surrounded by curly brackets.

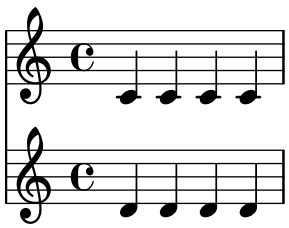
This single music expression may be of any size, and may contain other music expressions to any complexity. All of these examples are music expressions:

```
{ 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 }
    >>
  >>
}
```

[1] Note that there also exists a markup command called `\score`, see [Scores within markup], page 860.

Comments are one exception to this general rule. (For others, see Section 20.5 [File structure], page 596.) 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.

Remember that even in a file containing only a \score block, it is implicitly enclosed in a \book block. A \book block in a source file produces at least one output file, and by default the name of the output file produced is derived from the name of the input file, so fandangoforelephants.ly will produce fandangoforelephants.pdf.

(For more details about \book blocks, see Section 20.2 [Multiple scores in a book], page 593, Section 20.3 [Multiple output files from one input file], page 595, Section 20.5 [File structure], page 596.)

See also

Learning Manual: Section "Working on input files" in *Learning Manual*, Section "Music expressions explained" in *Learning Manual*, Section "A score is a (single) compound musical expression" in *Learning Manual*.

20.2 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 {
  ...
}
```

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 { ... }
  ...
}
```

Variables defined within a `\book` or `\bookpart` block are local to that book or bookpart; books and bookparts have their own 'scope'. In contrast, a `\score` does not accept assignments and has no scope of its own.

If you want to use multiple files to set up your music with variables that should be 'local' to single files, this can be done with the following structure:

```
% movement1.ly
bookpartI = \bookpart {
  variable = { ... }
  \score { ... use \variable ... }
}

% movement2.ly
bookpartII = \bookpart {
  variable = { ... }
  \score { ... use \variable ... }
}

% main.ly
\include "movement1.ly"
\include "movement2.ly"
```

```
\book {
  \bookpart { \bookpartI }
  \bookpart { \bookpartII }
}
```

The values of `variable` in both bookparts are not related to one another. You cannot reference the variables outside of the book or bookpart that they are defined in.

`\layout`, `\paper`, and `\midi` blocks at the outer level of a `\book` or `\bookpart` establish local defaults for scores at that level, similar to how this happens at the top level of a document.

20.3 Multiple output files from one input file

LilyPond creates one output file for each `\book` block. If there is no explicit `\book` block in the input file, LilyPond implicitly treats the whole file as a single `\book` block, see Section 20.5 [File structure], page 596.

By default, LilyPond names the output file using the input file name and, if necessary, suffixes it with an increasing number – i.e., if an output file with the same name has already been created during the run. The default behavior is to append a version number suffix for each name that may clash, so

```
\book {
  \score { ... }
  \paper { ... }
}
\book {
  \score { ... }
  \paper { ... }
}
\book {
  \score { ... }
  \paper { ... }
}
```

in source file `eightminiatures.ly` produces

```
eightminiatures.pdf
eightminiatures-1.pdf
eightminiatures-2.pdf
```

as output files.

20.4 Output file names

It is possible to override both the output file name and the suffix appended to the basic file name in `\paper` blocks.

```
\paper {
  output-filename = "my_special_output"
}

\book {
  \paper {
    output-suffix = "menuetto"
  }
  ...
}
\book {
```

```
    \paper {
      output-suffix = "scherzo"
    }
    ...
}
```

The result are two output files named `my_special_output-menuetto.pdf` and `my_special_output-scherzo.pdf`. Be careful to select values for `output-filename` and `output-suffix` that are valid for file names on your operating system!

If this output file name already exists (this can happen, for example, if both `output-filename` and `output-suffix` are set in the global `\paper` block and not in any `\book` blocks), LilyPond appends an additional suffix with an increasing number.

`output-filename` and `output-suffix` can also be set in a `\midi` block to name the MIDI file generated from that block, which allows creating several MIDI files from a single score. A variable that is not set in the `\midi` block is taken from the book. In particular, an `output-suffix` set in a `\midi` block *replaces* the suffix of the book instead of being appended to it; use `##f` to get no suffix at all.

```
\score {
  ...music...
  \midi {
    output-suffix = "rehearsal"
    \context {
      \Score
      tempoWholesPerMinute = #(ly:make-moment 80 4)
    }
  }
  \midi { output-suffix = ##f }
}
```

In source file `eightminiatures.ly`, this produces `eightminiatures-rehearsal.midi` and `eightminiatures.midi`.

MIDI files are numbered independently of the printed output. If several MIDI files would get the same name, LilyPond appends an increasing number to the names. This number is always present if the output file name of the book has been given one, so the MIDI files of the second of two books in `test.ly` are named `test-1-1.midi`, `test-1-2.midi`, and so on, after `test.midi` and `test-1.midi` of the first book.

Note that predefined `\paper` variables (see Section 26.1 [The \paper block], page 671) must be placed before the `output-filename` and `output-suffix` assignments, for example

```
bigMargin = \paper { top-margin = 10\cm }

\book {
  \paper {
    \bigMargin % must come first
    output-filename = "foo"
  }
}
```

20.5 File structure

A `.ly` file may contain any number of top-level expressions, where a top-level expression is one of the following:

- An output definition, such as `\paper`, `\midi`, and `\layout`. Such a definition at the top-level changes the default book-wide settings. If more than one such definition of the same

type is entered at the top level the definitions are combined, but in conflicting situations the later definitions take precedence. For details of how this affects the `\layout` block see Section 27.1 [The `\layout` block], page 683.

- 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 (i.e., the top of file) header block. This is the block containing the default settings of titling fields like composer, title, etc., for all books within the file (see Section 21.1.1 [Titles explained], page 600).
- A `\score` block. This score will be collected with other top-level scores, and combined as a single `\book`. This behavior can be changed by setting the variable `toplevel-score-handler` at top level. (The default handler is defined in the file `../scm/lily-library.scm` and set in the file `../ly/declarations-init.ly`.)
- A `\book` block logically combines multiple movements (i.e., multiple `\score` blocks) in one document. If there are a number of `\score`s, 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 top level. 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 top-level `\score`s and music expressions. In other words, a file containing only the above music expression will be translated into

```
\book {
  \score {
    \new Staff {
      \new Voice {
        { c'4 d' e'2 }
      }
    }
    \layout { }
  }
  \paper { }
  \header { }
}
```

  This behavior can be changed by setting the variable `toplevel-music-handler` at top level. 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 or a nested structure of Scheme association lists (alists).

```
foo = { c4 d e d }       % normal variable
```

```
"Horn 3" = { c4 d e d }  % quoted variable
verse.1 = { c4 d e d }   % nested alist
```

Variables can be used later on in the file by prepending them with a backslash.

```
\foo
\"Horn 3"
\verse.1
```

The name of a variable should not contain (ASCII) numbers, multiple adjacent underscores, multiple adjacent dashes, or space characters. All other characters Unicode provides are allowed, for example Latin, Greek, Chinese, or Cyrillic. Non-adjacent single underscores and dashes are allowed, too. In other words, variable names like `HornIII` or `Скрипка-II` work.

Any combination of characters is allowed if the variable name is enclosed in double quotation marks (not that you actually should use such a name).[2] Examples: `"foo bar"`, `"a-b-c"`, `"Horn 3"`.

Nested alists provide a means to partially circumvent the abovementioned restriction of variable names not containing a number. A nested alist is a valid variable name followed by one or more sets of a dot and a key. In `mus.violin.1 = { a1 }` we have a variable called `mus`, which has an alist with a key called `violin`, which in turn has a key called `1`, which is eventually assigned the value `{ a1 }`. In `"1.2"."3.4".5` the variable name is `"1.2"` and the keys are `"3.4"` and `5` – such rather illegible constructs might be useful for programmatically generated variable names.

Neither a variable name nor a key in a nested alist is allowed to be a note name. If necessary, enclose those in double quotation marks, or better, change the offending name or key.

For technical reasons, accessing nested alists might fail under some circumstances. In particular, they cannot be accessed at top-level (i.e., outside of a music block). If `lilypond` aborts with the error 'bad expression type', replace the nested alist with a normal or quoted variable.

The following example shows three things that may be entered at top level.

```
\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 `%{ ... %}`.

[2] For this case the standard LilyPond rules for strings apply: backslashes and double quotation marks within the doublequotes need to be escaped with backslashes.

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` before and after `\set` and `\override` commands.

See also

Learning Manual: Section "How LilyPond input files work" in *Learning Manual*.

Notation Reference: Section 21.1.1 [Titles explained], page 600, Section 27.1 [The `\layout` block], page 683.

21 Titles and headers

Almost all printed music includes a title and the composer's name; some pieces include a lot more information.

21.1 Creating titles, headers, and footers

21.1.1 Titles explained

Each `\book` block in a single input file produces a separate output file, see Section 20.5 [File structure], page 596. Within each output file three types of titling areas are provided: *book titles* at the beginning of each book, *bookpart titles* at the beginning of each bookpart, and *score titles* at the beginning of each score.

Values of titling fields such as `title` and `composer` are set in `\header` blocks. (For the syntax of `\header` blocks and a complete list of the fields available by default, see Section 21.1.2 [Default layout of bookpart and score titles], page 603). Book titles, bookpart titles, and score titles can all contain the same fields, although by default the fields in score titles are limited to `piece` and `opus`.

`\header` blocks may be placed in four different places to form a descending hierarchy.

- At the top of the input file, before all `\book`, `\bookpart`, and `\score` blocks.
- Within a `\book` block but outside all the `\bookpart` and `\score` blocks within that book.
- Within a `\bookpart` block but outside all `\score` blocks within that bookpart.
- Within a `\score` block.

The values of the fields filter down this hierarchy, with the values set higher in the hierarchy persisting unless they are overridden by a value set lower in the hierarchy.

- A book title is derived from fields set at the top of the input file, modified by fields set in the `\book` block. The resulting fields are used to print the book title for that book, providing that there is other material that generates a page at the start of the book, before the first bookpart. A single `\pageBreak` suffices.
- A bookpart title is derived from fields set at the top of the input file, modified by fields set in the `\book` block, and further modified by fields set in the `\bookpart` block. The resulting values are used to print the bookpart title for that bookpart.
- A score title is derived from fields set at the top of the input file, modified by fields set in the `\book` block, further modified by fields set in the `\bookpart` block and finally modified by fields set in the `\score` block. The resulting values are used to print the score title for that score. Note, though, that only `piece` and `opus` fields are printed by default in score titles unless the `\paper` variable `print-all-headers` is set to `#t`.

It is not necessary to provide `\header` blocks in all four places: any or even all of them may be omitted. Similarly, simple input files may omit the `\book` and `\bookpart` blocks, leaving them to be created implicitly.

If the book has only a single score, the `\header` block should normally be placed at the top of the file so that just a bookpart title is produced, making all the titling fields available for use.

If the book has multiple scores a number of different arrangements of `\header` blocks are possible, corresponding to the various types of musical publications. For example, if the publication contains several pieces by the same composer, a `\header` block placed at the top of the file specifying the book title and the composer with `\header` blocks in each `\score` block specifying the `piece` and/or `opus` would be most suitable, as here:

```
\header {
  title = "SUITE I."
  composer = "J. S. Bach."
}
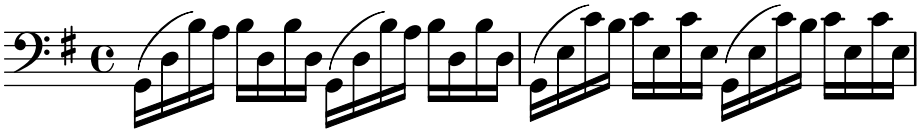
\score {
  \header {
    piece = "Prélude."
  }
  \new Staff \relative {
    \clef bass
    \key g \major
    \*2 { g,16( d' b') a b d, b' d, } |
    \*2 { g,16( e' c') b c e, c' e, } |
  }
}

\score {
  \header {
    piece = "Allemande."
  }
  \new Staff \relative {
    \clef bass
    \key g \major
    \partial 16 b16 |
    <g, d' b'~>4 b'16 a( g fis) g( d e fis) g( a b c) |
    d16( b g fis) g( e d c) b(c d e) fis( g a b) |
  }
}
```
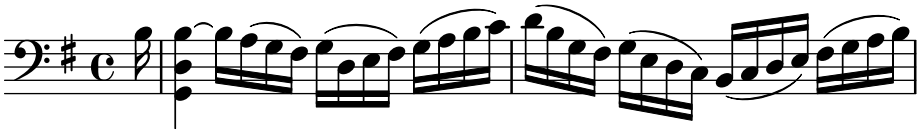
SUITE I.

J. S. Bach.

Prélude.
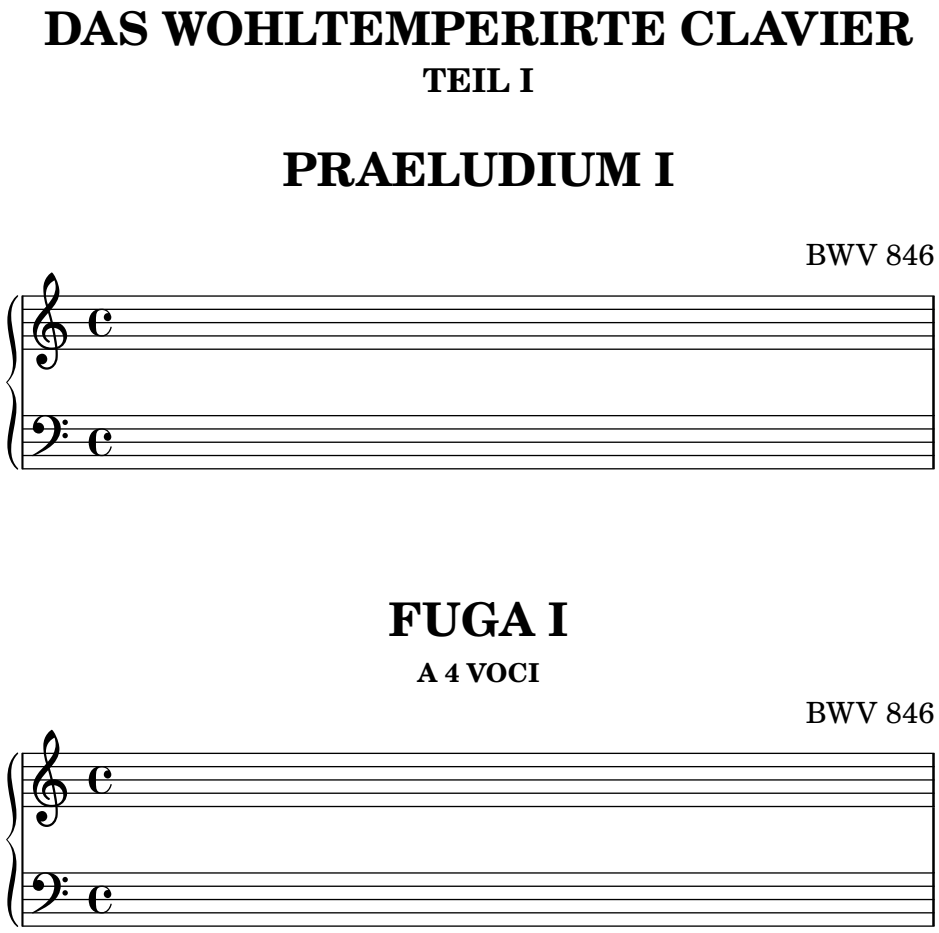


Allemande.



More complicated arrangements are possible. For example, text fields from the `\header` block in a book can be displayed in all score titles, with some fields overridden and some manually suppressed:

```
\book {
  \paper {
    print-all-headers = ##t
  }
  \header {
    title = "DAS WOHLTEMPERIRTE CLAVIER"
    subtitle = "TEIL I"
    % Do not display the default LilyPond footer for this book
    tagline = ##f
  }
  \markup { \vspace #1 }
  \score {
    \header {
      title = "PRAELUDIUM I"
      opus = "BWV 846"
      % Do not display the subtitle for this score
      subtitle = ##f
    }
    \new PianoStaff <<
      \new Staff { s1 }
      \new Staff { \clef "bass" s1 }
    >>
  }
  \score {
    \header {
      title = "FUGA I"
      subsubtitle = "A 4 VOCI"
      opus = "BWV 846"
      % Do not display the subtitle for this score
      subtitle = ##f
    }
    \new PianoStaff <<
      \new Staff { s1 }
      \new Staff { \clef "bass" s1 }
    >>
  }
}
```



See also

Notation Reference: Section 20.5 [File structure], page 596, Section 21.1.2 [Default layout of bookpart and score titles], page 603, Section 21.2.2 [Custom layout for titles], page 608.

21.1.2 Default layout of bookpart and score titles

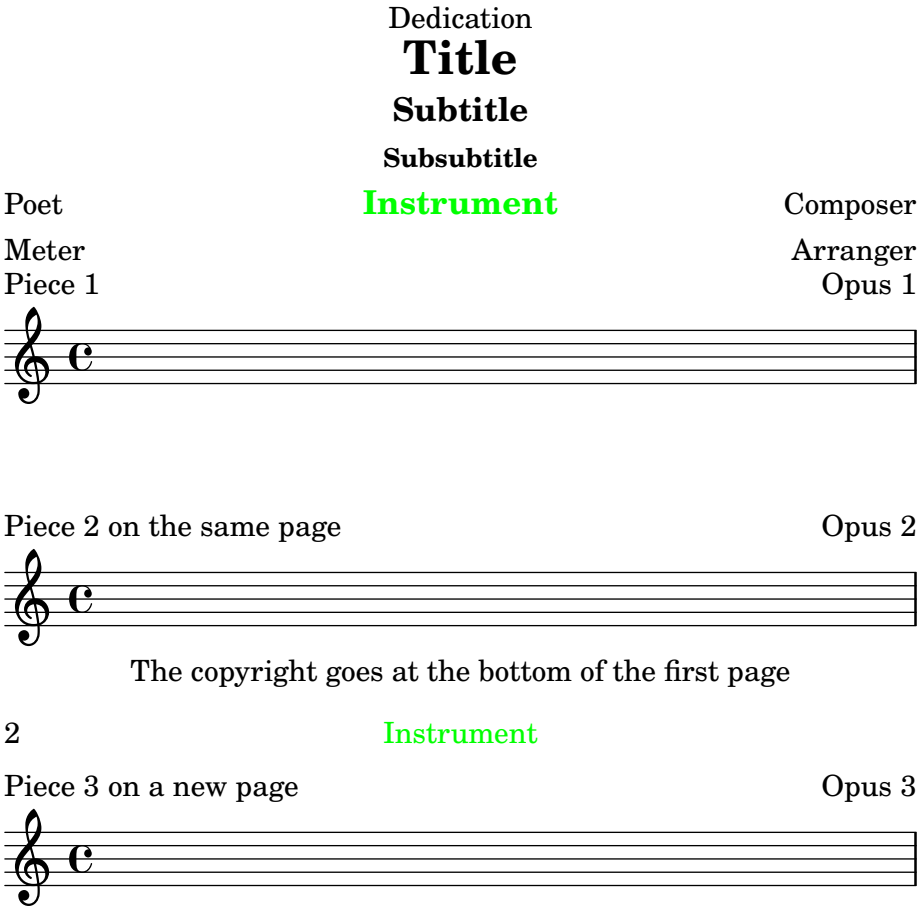
The next example demonstrates all printed `\header` variables. Note that the vertical spacing between the various header elements in the default layout is optimized for single-line entries. If you need multi-line elements, say, a two-line composer entry, try to append `\vspace` to the field if necessary to adjust the vertical spacing. An alternative is to define your own, custom layout, see Section 21.2.2 [Custom layout for titles], page 608.

```
\book {
  \header {
    % The following fields are centered
    dedication = "Dedication"
    title = "Title"
    subtitle = "Subtitle"
    subsubtitle = "Subsubtitle"

    % The following fields are evenly spread on one line;
    % the field "instrument" also appears on following pages
    instrument = \markup \with-color #green "Instrument"
    poet = "Poet"
    composer = "Composer"
```

```
    % The following fields are placed at opposite ends
    %of the same line
    meter = "Meter"
    arranger = "Arranger"

    % The following fields are centered at the bottom
    tagline = "The tagline goes at the bottom of the last page"
    copyright = "The copyright goes at the bottom of the first page"
  }
  \score {
    \header {
      % The following fields are placed at opposite ends
      % of the same line
      piece = "Piece 1"
      opus = "Opus 1"
    }
    { s1 }
  }
  \score {
    \header {
      % The following fields are placed at opposite ends
      % of the same line
      piece = "Piece 2 on the same page"
      opus = "Opus 2"
    }
    { s1 }
  }
  \pageBreak
  \score {
    \header {
      % The following fields are placed at opposite ends
      % of the same line
      piece = "Piece 3 on a new page"
      opus = "Opus 3"
    }
    { s1 }
  }
}
```



Note that

- the instrument name is repeated on every page,
- only `piece` and `opus` are printed in a `\score` when the paper variable `print-all-headers` is set to `#f` (the default),
- text fields left unset in a `\header` block produce no output so that the space is not wasted,
- the default settings for `scoreTitleMarkup` place the `piece` and `opus` text fields at opposite ends of the same line.

To change the default layout, see Section 21.2.2 [Custom layout for titles], page 608.

If a \book block starts immediately with a \bookpart block, no book title gets printed, as there is no page on which to print it. If a book title is required, begin the \book block with some markup material or a \pageBreak command.

Use the breakbefore variable inside a \header block that is itself in a \score block, to make the higher-level \header block titles appear on the first page on their own, with the music (defined in the \score block) starting on the next page.

```
\book {
  \header {
    title = "This is my Title"
    subtitle = "This is my Subtitle"
    copyright = "This is the bottom of the first page"
  }
  \score {
    \header {
      piece = "This is the Music"
      breakbefore = ##t
    }
    \*4 { e'' e'' e'' e'' }
  }
}
```
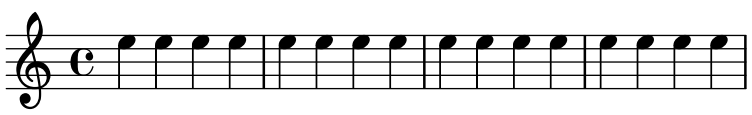
This is my Title

This is my Subtitle

This is the bottom of the first page

2

This is the Music



LilyPond v2.27.5

See also

Learning Manual: Section “How LilyPond input files work” in *Learning Manual*.

Notation Reference: Section 21.2.2 [Custom layout for titles], page 608, Section 20.5 [File structure], page 596.

Installed Files: `ly/titling-init.ly`.

21.1.3 Default layout of headers and footers

Headers and *footers* are lines of text appearing at the top and bottom of pages, separate from the main text of a book. They are controlled by the following `\paper` variables:
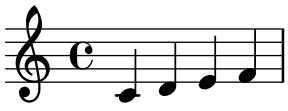
- `oddHeaderMarkup`
- `evenHeaderMarkup`
- `oddFooterMarkup`
- `evenFooterMarkup`

These markup variables are defined in `ly/titling-init.ly` and do the following by default.

- Page numbers are automatically placed on the top far left (if even) or top far right (if odd), starting from the second page.
- The `instrument` header field is placed in the center of every page, starting from the second page.
- The `copyright` header field is centered on the bottom of the first page.
- The `tagline` header field is centered on the bottom of the last page, and below the `copyright` field if there is only a single page.

The following shows an example how to change the tag line.

```
\book {
  \header {
    tagline = "... music notation for Everyone"
  }
  \score {
    \relative {
      c'4 d e f
    }
  }
}
```



... music notation for Everyone

Set `tagline` to `#f` if you don’t want a tag line.

See also

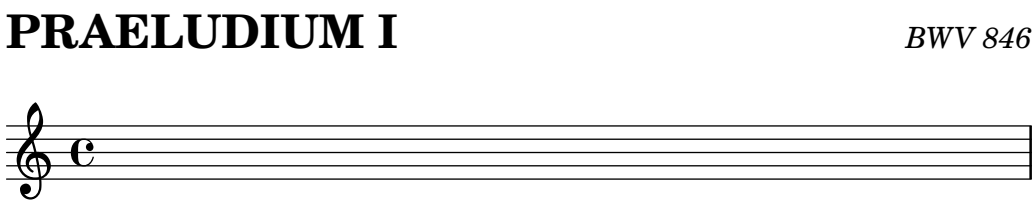
Notation Reference: Section 21.2.3 [Custom layout for headers and footers], page 611.

21.2 Custom titles, headers, and footers

21.2.1 Custom text formatting for titles

Standard \markup commands can be used to customize any header, footer, or title text within the \header block.

```
\score {
  \header {
    piece = \markup { \fontsize #4 \bold "PRAELUDIUM I" }
    opus = \markup { \italic "BWV 846" }
  }
  { s1 }
}
```



See also

Notation Reference: Section 8.2 [Formatting text], page 320.

21.2.2 Custom layout for titles

\markup commands in the \header block are useful for simple text formatting, but they do not allow precise control over the placement of titles. To customize the placement of the text fields, change either or both of the following \paper variables:

- bookTitleMarkup
- scoreTitleMarkup

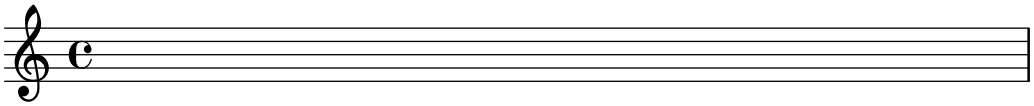
See Section 21.1.2 [Default layout of bookpart and score titles], page 603, for the placement of titles when using the default values of these \markup variables.

The default setting for scoreTitleMarkup defined in ly/titling-init.ly is as follows.

```
scoreTitleMarkup = \markup \column {
  \if \should-print-all-headers { \bookTitleMarkup \hspace #1 }
  \fill-line {
    \fromproperty #'header:piece
    \fromproperty #'header:opus
  }
}
```

This places the piece and opus text fields at opposite ends of the same line.

```
\score {
  \header {
    piece = "PRAELUDIUM I"
    opus = "BWV 846"
  }
  { s1 }
}
```



The next example redefines scoreTitleMarkup so that the piece text field is centered, using a large, bold font.

```
\book {
  \paper {
    indent = 0\mm
    scoreTitleMarkup = \markup {
      \fill-line {
        \null
        \fontsize #4 \bold \fromproperty #'header:piece
        \fromproperty #'header:opus
      }
    }
  }
  \header { tagline = ##f }
  \score {
    \header {
      piece = "PRAELUDIUM I"
      opus = "BWV 846"
    }
    { s1 }
  }
}
```

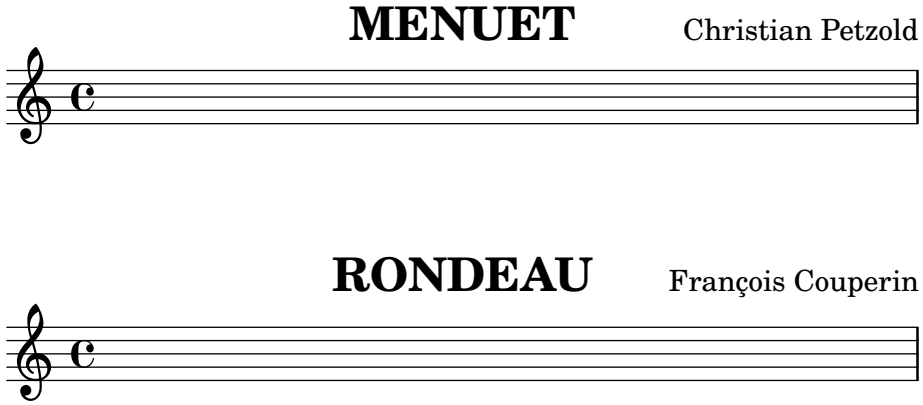


Text fields not normally effective in score \header blocks can be printed in the score title area if print-all-headers is placed inside the \paper block. A disadvantage of using this method is that text fields intended specifically for the bookpart title area need to be manually suppressed in every \score block. See Section 21.1.1 [Titles explained], page 600.

To avoid this, add the desired text field to the scoreTitleMarkup definition. In the following example, the composer text field (normally associated with bookTitleMarkup) is added to scoreTitleMarkup, allowing each score to list a different composer.

```
\book {
  \paper {
    indent = 0\mm
    scoreTitleMarkup = \markup {
      \fill-line {
        \null
        \fontsize #4 \bold \fromproperty #'header:piece
        \fromproperty #'header:composer
      }
    }
  }
  \header { tagline = ##f }
  \score {
```
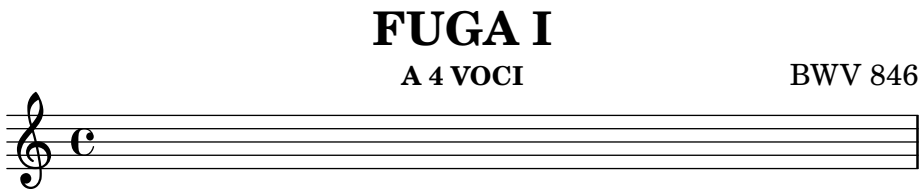
```
    \header {
      piece = "MENUET"
      composer = "Christian Petzold"
    }
    { s1 }
  }
  \score {
    \header {
      piece = "RONDEAU"
      composer = "François Couperin"
    }
    { s1 }
  }
}
```



It is also possible to create your own custom text fields and refer to them in the markup definition.

```
\book {
  \paper {
    indent = 0\mm
    scoreTitleMarkup = \markup {
      \fill-line {
        \null
        \override #`(direction . ,UP)
        \dir-column {
          \center-align \fontsize #-1 \bold
            \fromproperty #'header:mycustomtext %% User-defined field
          \center-align \fontsize #4 \bold
            \fromproperty #'header:piece
        }
        \fromproperty #'header:opus
      }
    }
  }
  \header { tagline = ##f }
  \score {
    \header {
      piece = "FUGA I"
      mycustomtext = "A 4 VOCI" %% User-defined field
      opus = "BWV 846"
```

```
    }
    { s1 }
  }
}
```



See also

Notation Reference: Section 21.1.1 [Titles explained], page 600.

21.2.3 Custom layout for headers and footers

`\markup` commands in the `\header` block are useful for simple text formatting, but they do not allow precise control over the placement of headers and footers. To customize the placement of the text fields, use one or more of the following `\paper` variables.

- `oddHeaderMarkup`
- `evenHeaderMarkup`
- `oddFooterMarkup`
- `evenFooterMarkup`

The `\markup` command `\if` can be used to add markup conditionally to header and footer text defined within the `\paper` block, using the following syntax within `\markup`:

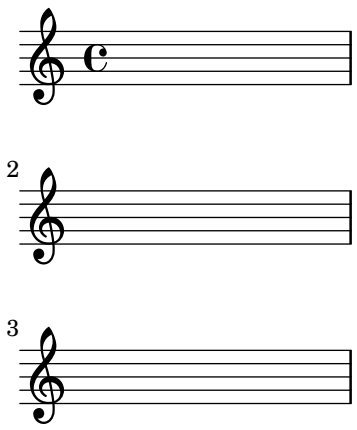
```
\if condition argument
```

The *condition* is tested each time the markup is interpreted. The resulting markup is *argument* if the condition is true, but empty if false. Typical conditions include tests for page numbers (first page, last page, specific page, . . .). To test for the condition being false, replace `\if` with `\unless`.

The following example centers page numbers at the bottom of every page. First, the default settings for `oddHeaderMarkup` and `evenHeaderMarkup` are removed by assigning `#f`. Then, `oddFooterMarkup` is redefined with the page number centered. Finally, `evenFooterMarkup` is given the same layout by defining it as `\oddFooterMarkup`:

```
\book {
  \paper {
    print-page-number = ##t
    print-first-page-number = ##t
    oddHeaderMarkup = ##f
    evenHeaderMarkup = ##f
    oddFooterMarkup = \markup {
      \fill-line {
        \if \should-print-page-number
          \fromproperty #'page:page-number-string
      }
    }
    evenFooterMarkup = \oddFooterMarkup
  }
  \score {
    \new Staff { s1 \break s1 \break s1 }
```

```
  }
}
```



1

Here is a list of all predefined procedures available for use with `\if` and `\unless`.

Syntax	**Condition tested**
`\on-first-page`	First page in the book?
`\on-last-page`	Last page in the book?
`\on-first-page-of-part`	First page in the book part?
`\on-last-page-of-part`	Last page in the book part?
`\on-page` *n*	On page number *n*?
`\single-page`	Does the book fit on just one page?
`\should-print-page-numbers-global`	Print page numbers in the book?[1]
`\should-print-page-number`	Print the number of the current page?
`\should-print-all-headers`	Is `print-all-headers` true?

See also

Notation Reference: Section 21.1.1 [Titles explained], page 600, Section 21.1.2 [Default layout of bookpart and score titles], page 603, Section A.1.5 [Conditional markup], page 864.

Installed Files: `../ly/titling-init.ly`.

21.3 Creating output file metadata

In addition to being shown in the printed output, `\header` variables are also used to set metadata for output files. For example, with PDF files, this metadata could be displayed by PDF readers as the `properties` of the PDF file. For each type of output file, only the `\header` definitions of blocks that define separate files of that type, and blocks higher in the block hierarchy, will be consulted. Therefore, for PDF files, only the `\book` level and the top level `\header` definitions affect the document-wide PDF metadata, whereas for MIDI files, all headers above or at the `\score` level are used.

For example, setting the `title` property of the `header` block to 'Symphony I' will also give this title to the PDF document, and use it as the sequence name of the MIDI file.

```
\header {
```

[1] `\should-print-page-numbers-global` can differ from `\should-print-page-number` for the first page in the book, depending on the `print-first-page-number` setting of the `\paper` block.

```
  title = "Symphony I"
}
```

If you want to set the title of the printed output to one value, but have the title property of the PDF to have a different value, you can use `pdftitle`, as below.

```
\header {
  title = "Symphony I"
  pdftitle = "Symphony I by Beethoven"
}
```

The variables `title`, `subject`, `keywords`, `subtitle`, `composer`, `arranger`, `poet`, `author` and `copyright` all set PDF properties and can all be prefixed with 'pdf' to set a PDF property to a value different from the printed output.

The PDF property `Creator` is automatically set to 'LilyPond' plus the current LilyPond version, and `CreationDate` and `ModDate` are both set to the current date and time. `ModDate` can be overridden by setting the header variable `moddate` (or `pdfmoddate`) to a valid PDF date string.

The `title` variable sets also the sequence name for MIDI. The `miditítle` variable can be used to set the sequence name independently of the value used for typeset output.

21.4 Creating footnotes

Footnotes may be used in many different situations. In all cases, a 'footnote mark' is placed as a reference in text or music, and the corresponding 'footnote text' appears at the bottom of the same page, separated from the music by a horizontal line. The appearance of this separator can be changed by setting the paper variable `footnote-separator-markup`, see Section 26.6.4 [\paper variables concerning headers and markups], page 681.

Footnotes within music expressions and footnotes in stand-alone text outside music expressions are created in different ways.

21.4.1 Footnotes in music expressions

Music footnotes overview

Footnotes in music expressions fall into two categories:

Event-based footnotes
: are attached to a particular event. Examples for such events are single notes, articulations (like fingering indications, accents, dynamics), and post-events (like slurs and manual beams). The general form for event-based footnotes is as follows:

  `[`*direction*`] \footnote [`*mark*`]` *offset footnote music*

Time-based footnotes
: are bound to a particular point of time in a musical context. Some commands like `\time` and `\clef` don't actually use events for creating objects like time signatures and clefs. Neither does a chord create an event of its own: its stem or flag is created at the end of a time step (nominally through one of the note events inside). Exactly which of a chord's multiple note events will be deemed the root cause of a stem or flag is undefined. So for annotating those, time-based footnotes are preferable as well.

  A time-based footnote allows such layout objects to be annotated without referring to an event. The general form for time-based footnotes is:

  `\footnote [`*mark*`]` *offset footnote* `[`*Context*`].`*GrobName*

The elements for both forms are:

direction If (and only if) the `\footnote` is being applied to a post-event or articulation, it must be preceded with a direction indicator (‘`-`’, ‘`_`’, ‘`^`’) in order to attach *music* (with a footnote mark) to the preceding note or rest.

mark is a markup or string specifying the footnote mark which is used for marking both the reference point and the footnote itself at the bottom of the page. It may be omitted (or equivalently replaced with `\default`) in which case a number in sequence will be generated automatically. By default, such numerical sequences restart on each page containing a footnote. Footnotes may be numbered consecutively across page beaks by setting the variable `reset-footnotes-on-new-page` to `#f`, see Section 26.6.4 [`\paper` variables concerning headers and markups], page 681.

offset is a number pair such as ‘`#'(2 . 1)`’ specifying the X and Y offsets in units of staff spaces from the boundary of the object where the mark should be placed. Positive values of the offsets are taken from the right/top edge, negative values from the left/bottom edge and zero implies the mark is centered on the edge.

Context is the context in which the grob being footnoted is created. It may be omitted if the grob is in a bottom context, e.g., a `Voice` context.

GrobName
specifies a type of grob to mark (like ‘`Flag`’). If it is specified, the footnote is not attached to a music expression in particular, but rather to all grobs of the type specified which occur at that moment of musical time.

footnote is the markup or string specifying the footnote text to use at the bottom of the page.

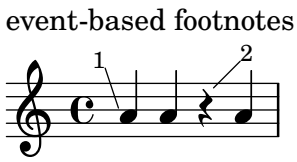
music is the music event or post-event or articulation that is being annotated.

Event-based footnotes

A footnote may be attached to a layout object directly caused by the event corresponding to *music* with the syntax:

```
\footnote [mark] offset footnote music
```
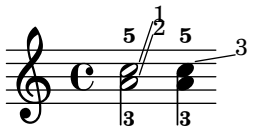
```
\book {
  \header { tagline = ##f }
  \markup "event-based footnotes"
  \markup \null
  \relative c'' {
    \footnote #'(-1 . 3) "A note." a4
    a4
    \footnote #'(2 . 2) "A rest." r4
    a4
  }
}
```



[1] A note.
[2] A rest.

If a chord is marked with an event-based footnote, each chord note gets a separate but identical footnote, which is undesired normally. However, it is possible to create footnotes for individual notes inside of a chord.

```
\book {
  \header { tagline = ##f }
  \markup "event-based footnotes"
  \markup \null
  \relative c'' {
    \footnote #'(1 . 3) "A chord." <a-3 c-5>2
    <a-3 \footnote #'(3 . 0.5) "A note in a chord." c-5>4
  }
}
```



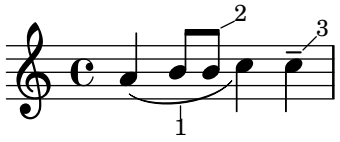
[1] A chord.
[2] A chord.
[3] A note in a chord.

If the footnote is to be attached to a post-event or articulation the `\footnote` command *must* be preceded by a direction indicator ('-', '_', '^'), and followed by the post-event or articulation to be annotated as the *music* argument. In this form the `\footnote` can be considered to be simply a copy of its last argument with a footnote mark attached to it. The syntax is:

direction \footnote [*mark*] *offset footnote music*

```
\book {
  \header { tagline = ##f }
  \markup "event-based footnotes"
  \markup \null
  \relative {
    a'4_\footnote #'(0 . -1) "A slur forced down." (
    b8^\footnote #'(1 . 0.5) "A manual beam forced up." [
    b8 ]
    c4 )
    c-\footnote #'(1 . 1) "Tenuto." --
  }
}
```

event-based footnotes



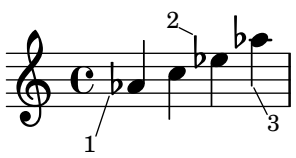
[1]A slur forced down.
[2]A manual beam forced up.
[3]Tenuto.

Time-based footnotes

If the layout object being footmarked is *indirectly* caused by an event (like an `Accidental` or `Stem` caused by a `NoteHead` event), the *GrobName* of the layout object is required after the footnote text instead of *music*:

```
\book {
  \header { tagline = ##f }
  \markup "time-based footnotes"
  \markup \null
  \relative c'' {
    \footnote #'(-1 . -3) "A flat." Accidental
    aes4 c
    \footnote #'(-1 . 0.5) "Another flat." Accidental
    ees
    \footnote #'(1 . -2) "A stem." Stem
    aes
  }
}
```

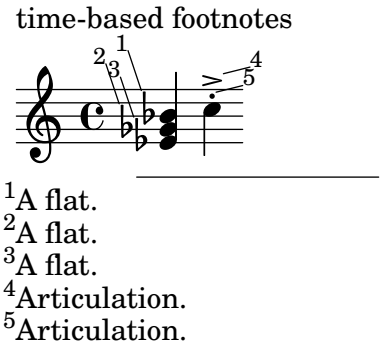
time-based footnotes



[1]A flat.
[2]Another flat.
[3]A stem.

Note, however, that when a *GrobName* is specified, a footnote is attached to all grobs of that type at the current time step:
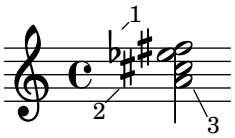
```
\book {
  \header { tagline = ##f }
  \markup "time-based footnotes"
  \markup \null
  \markup \null
  \relative c' {
    \footnote #'(-1 . 3) "A flat." Accidental
    <ees ges bes>4
    \footnote #'(2 . 0.5) "Articulation." Script
    c'->-.
  }
}
```



A note inside of a chord can be given an individual (event-based) footnote. A 'NoteHead' is the only grob directly caused from a chord note, so an event-based footnote command is *only* suitable for adding a footnote to the 'NoteHead' within a chord. All other chord note grobs are indirectly caused. The \footnote command itself offers no syntax for specifying *both* a particular grob type *as well as* a particular event to attach to. However, one can use a time-based \footnote command for specifying the grob type, and then prefix this command with \single in order to have it applied to just the following event:

```
\book {
  \header { tagline = ##f }
  \markup "time-based footnotes"
  \markup \null
  \relative c'' {
    < \footnote #'(1 . -2) "An A." a
      \single \footnote #'(-1 . -1) "A sharp." Accidental
      cis
      \single \footnote #'(0.5 . 0.5) "A flat." Accidental
      ees fis
    >2
  }
}
```

time-based footnotes



[1]A flat.
[2]A sharp.
[3]An A.

Note: When footnotes are attached to several musical elements at the same musical moment, as they are in the example above, the footnotes are numbered from the higher to the lower elements as they appear in the printed output, not in the order in which they are written in the input stream.

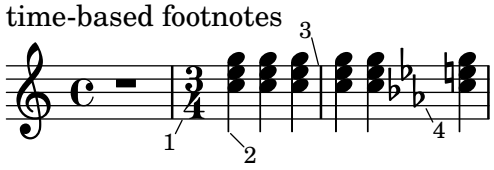
Layout objects like clefs and key change signatures are mostly caused as a consequence of changed properties rather than actual events. Others, like bar lines and bar numbers, are a direct consequence of timing. For this reason, footnotes on such objects have to be based on their musical timing. Time-based footnotes are also preferable when marking features like stems and beams on *chords*: while such per-chord features are nominally assigned to *one* event inside the chord, relying on a particular choice would be imprudent.

The layout object in question must always be explicitly specified for time-based footnotes, and the appropriate context must be specified if the grob is created in a context other than the bottom context.

```
\book {
  \header { tagline = ##f }
  \markup "time-based footnotes"
  \relative c'' {
    r1 |
    \footnote #'(-0.5 . -1) "Meter change." Staff.TimeSignature
    \time 3/4
    \footnote #'(1 . -1) "Chord stem." Stem
    <c e g>4 q q
    \footnote #'(-0.5 . 2) "Bar line." Staff.BarLine
    q q
```

```
    \footnote #'(0.5 . -1) "Key change." Staff.KeySignature
    \key c \minor
    q
  }
}
```

time-based footnotes
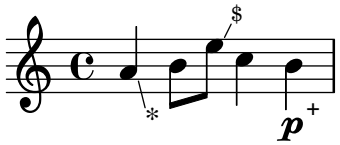


[1]Meter change.
[2]Chord stem.
[3]Bar line.
[4]Key change.

Custom marks can be used as alternatives to numerical marks, and the annotation line joining the marked object to the mark can be suppressed:

```
\book {
  \header { tagline = ##f }
  \markup "footnotes with custom marks"
  \markup \null
  \relative c' {
    \footnote "*" #'(0.5 . -2) \markup { \italic "* The first note" }
    a'4 b8
    \footnote \markup { \super "$" } #'(0.5 . 1)
      \markup { \super "$" \italic " The second note." } e
    c4
    \once \override Score.Footnote.annotation-line = ##f
    b-\footnote \markup \tiny "+" #'(0.1 . 0.1)
      \markup { \super "+" \italic " Editorial." } \p
  }
}
```

footnotes with custom marks



* *The first note*
$ *The second note.*
+ *Editorial.*

More examples of custom marks are shown in Section 21.4.2 [Footnotes in stand-alone text], page 619.

21.4.2 Footnotes in stand-alone text

These are for use in markup outside of music expressions. They do not have a line drawn to their point of reference: their marks simply follow the referenced markup. Marks can be inserted automatically, in which case they are numerical. Alternatively, custom marks can be provided manually.

Footnotes to stand-alone text with automatic and custom marks are created in different ways.

Footnotes in stand-alone text with automatic marks

The syntax of a footnote in stand-alone text with automatic marks is

```
\markup { ... \auto-footnote text footnote ... }
```
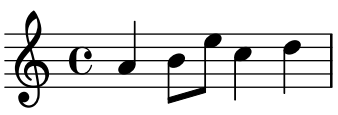
The elements are:

text the markup or string to be marked,

footnote the markup or string specifying the footnote text to use at the bottom of the page.

For example:

```
\book {
  \header { tagline = ##f }
  \markup {
    "A simple"
    \auto-footnote "tune" \italic " By me."
    "is shown below.  It is a"
    \auto-footnote "recent" \italic " Aug 2012."
    "composition."
  }
  \relative {
    a'4 b8 e c4 d
  }
}
```



[1] *By me.*
[2] *Aug 2012.*

Footnotes in stand-alone text with custom marks

The syntax of a footnote in stand-alone text with custom marks is

```
\markup { ... \footnote mark footnote ... }
```

The elements are:
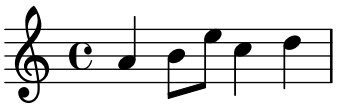
mark is a markup or string specifying the footnote mark which is used for marking the reference point. Note that this mark is *not* inserted automatically before the footnote itself.

footnote is the markup or string specifying the footnote text to use at the bottom of the page, preceded by the *mark*.

Any easy to type character such as '*' or '+' may be used as a mark, as shown in Section 21.4.1 [Footnotes in music expressions], page 613. Alternatively, ASCII aliases may be used (see Section 22.4.3 [ASCII aliases], page 648):

```
\book {
  \paper { #(include-special-characters) }
  \header { tagline = ##f }
  \markup {
    "A simple tune"
    \footnote "*" \italic "* By me."
    "is shown below.  It is a recent"
    \footnote \super &dagger; \concat {
      \super &dagger; \italic " Aug 2012."
    }
    "composition."
  }
  \relative {
    a'4 b8 e c4 d
  }
}
```

A simple tune * is shown below. It is a recent † composition.
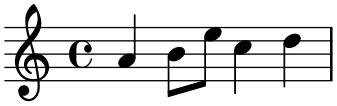


* *By me.*
† *Aug 2012.*

Unicode character codes may also be used to specify marks (see Section 22.4.2 [Unicode], page 648):

```
\book {
  \header { tagline = ##f }
  \markup {
    "A simple tune"
    \footnote \super \char##x00a7 \concat {
      \super \char##x00a7 \italic " By me."
    }
    "is shown below.  It is a recent"
    \footnote \super \char##x00b6 \concat {
      \super \char##x00b6 \italic " Aug 2012."
    }
    "composition."
  }
  \relative {
```

```
    a'4 b8 e c4 d
  }
}
```

A simple tune [§] is shown below. It is a recent [¶] composition.



[§] *By me.*
[¶] *Aug 2012.*

See also

Learning Manual: Section "Objects and interfaces" in *Learning Manual*.

Notation Reference: Section 22.4.3 [ASCII aliases], page 648, Section 7.2.2 [Balloon help], page 300, Section B.12 [List of special characters], page 927, Section 8.1.5 [Text marks], page 314, Section 8.1.2 [Text scripts], page 310, Section 22.4.2 [Unicode], page 648.

Internals Reference: Section "FootnoteEvent" in *Internals Reference*, Section "Footnote" in *Internals Reference*, Section "Footnote_engraver" in *Internals Reference*.

Known issues and warnings

Multiple footnotes for the same page can only be stacked, one above the other; they cannot be printed on the same line.

Footnote marks may collide with staves, `\markup` objects, other footnote marks and annotation lines.

21.5 Creating in-notes

In-notes function like footnotes in that they serve to annotate music, but are different in that they are typeset either above or below the system to which the grob being annotated belongs.

To create an in-note, set the `footnote` property of the `Footnote` grob to `#f`. The distance between two in-notes can be controlled with the paper variable `in-note-padding`, the distance between the in-note and its associated system by `in-note-system-padding`. If you want in-notes positioned below its associated system, set paper variable `in-note-direction` to `DOWN`.

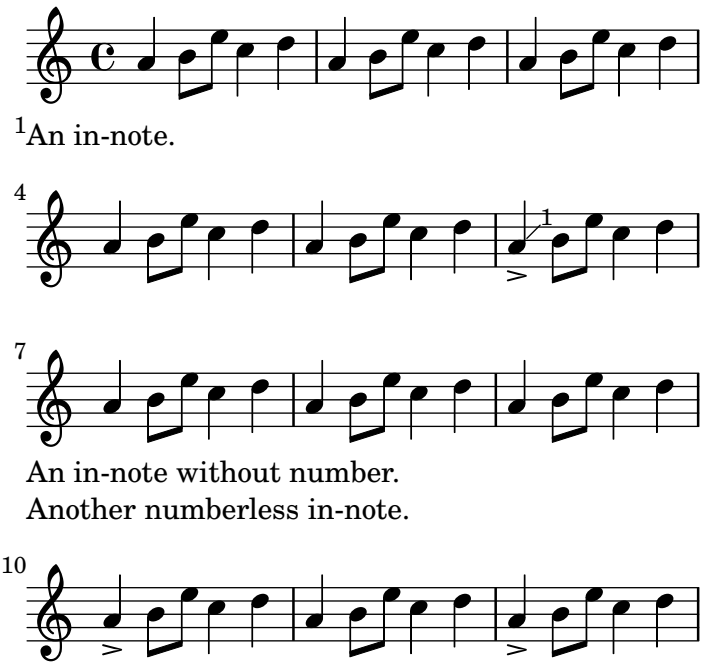
```
music = { a4 b8 e c4 d }

\book {
  \relative c'' {
    \override Score.Footnote.footnote = ##f

    \*5 \music
    \footnote #'(1 . 1) "An in-note." NoteHead
    <>-> \*4 \music
```

```
      \footnote "" #'(0 . 0) "An in-note without number." NoteHead
      <>-> \*2 \music
      \footnote "" #'(0 . 0) "Another numberless in-note." NoteHead
      <>-> \music
    }

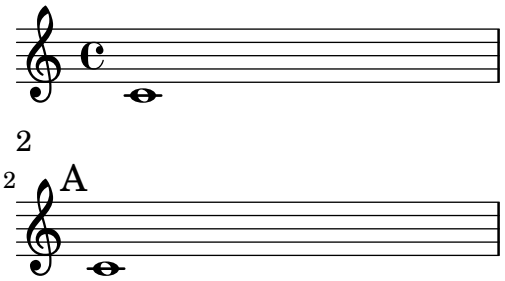
    \paper {
      in-note-system-padding = 5
      in-note-padding = 2
      tagline = ##f
    }
  }
```



21.6 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 2

The `\page-ref` markup command takes three arguments:

1. the label, a Scheme symbol, for example `#'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", i.e., a two digit number.

If the size of the final text is different from the gauge, it could be useful to configure the horizontal alignment relative to the gauge with the property `x-align`. The default is right-aligned.

```
\markup {
  \box
    \page-ref #'foo "???" "?" " right-aligned (default)"
}
\markup {
  \box
    \override #`(x-align . ,LEFT)
    \page-ref #'foo "???" "?" " left-aligned"
}
\markup {
  \box
    \override #'(x-align . -2.5)
    \page-ref #'foo "???" "?" " left outside"
}
```

? right-aligned (default)

? left-aligned

? left outside

In the example the gauge '`???`' is wider than the replacement text '`?`'. The replacement is used because the label `#'foo` does not exist. The property `x-align` can be set with any numbers. The predefined symbols `LEFT`, `CENTER` or `RIGHT` can also be used to set the alignment to left, center or right relative to the gauge.

Predefined commands

`\label`, `\page-ref`.

21.7 Table of contents

A table of contents is included using the `\markuplist \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.

In situations involving one or several books/bookparts, establishing the location for the bookkeeping of a single or multiple tables of contents may be warranted if it doesn't correspond to the first occurence of `\tocItem`. This can be achieved using `\setupTableOfContents`.

For maintaining tables of content in different layers, look up `\newTocItem`'s documentation.

```
\setupTableOfContents
\markuplist \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 % ...
    \tocItem actI \markup "Act I"
    f'4 % ...
    \tocItem actI.sceneI \markup "Scene 1"
    g'4 % ...
    \tocItem actI.sceneI.recitativo \markup "Recit."
    a'4 % ...
  }
}
```

Optionally, a label can be associated with a particular item, or a hierarchical list of existing labels, finishing with that item's label. That latter case allows to mark the item as a 'child' of the preceding labeled items, thus making the score's structure apparent in the table of contents.

Markups used for formatting the table of contents are defined in the `\paper` block. There are three 'predefined' markups already available;

- `tocTitleMarkup`

  Used for formatting the title of the table of contents.

  ```
  tocTitleMarkup = \markup \huge \column {
    \fill-line { \null "Table of Contents" \null }
    \null
  }
  ```

- `tocItemMarkup`

  Used for formatting the elements within the table of contents.

```
tocItemMarkup = \markup \fill-line {
  \fromproperty #'toc:text \fromproperty #'toc:page
}
```

- `tocFormatMarkup`

  How the table's top level entries will be formatted (if there are several hierarchical levels). This is actually a procedure, as explained in Section "Markup construction in Scheme" in *Extending*.

  ```
  tocFormatMarkup = #make-bold-markup
  ```

- `tocIndentMarkup`

  Used to define how the outline's hierarchy will be made apparent. This markup is printed zero, one or several times depending on the level of each entry.

  ```
  tocIndentMarkup = \markup \hspace #4
  ```

Any of these variables can be changed.

Here is an example translating the table of contents' title into French:

```
\paper {
  tocTitleMarkup = \markup \huge \column {
    \fill-line { \null "Table des matières" \null }
    \hspace #1
  }
```

Here is an example changing the font size of the elements in the table of contents:

```
tocItemMarkup = \markup \large \fill-line {
  \fromproperty #'toc:text \fromproperty #'toc:page
}
```

Note how the element text and page numbers are referred to in the `tocItemMarkup` definition.

The `\tocItemWithDotsMarkup` command can be included within the `tocItemMarkup` to fill the line, between a table of contents item and its corresponding page number, with dots:

```
\header { tagline = ##f }
\paper {
  tocItemMarkup = \tocItemWithDotsMarkup
}

\book {
  \markuplist \table-of-contents
  \tocItem \markup { Allegro }
  \tocItem \markup { Largo }
  \markup \null
}
```

Table of Contents

Allegro 1
Largo 1

In addition to the built-in outline mechanism, custom commands can also be defined to build a more personalized table of contents with different markups In the following example, a new style is defined for entering act and scenes in the table of contents of an opera:

A new markup variable (called `tocActMarkup`) is defined in the `\paper` block:

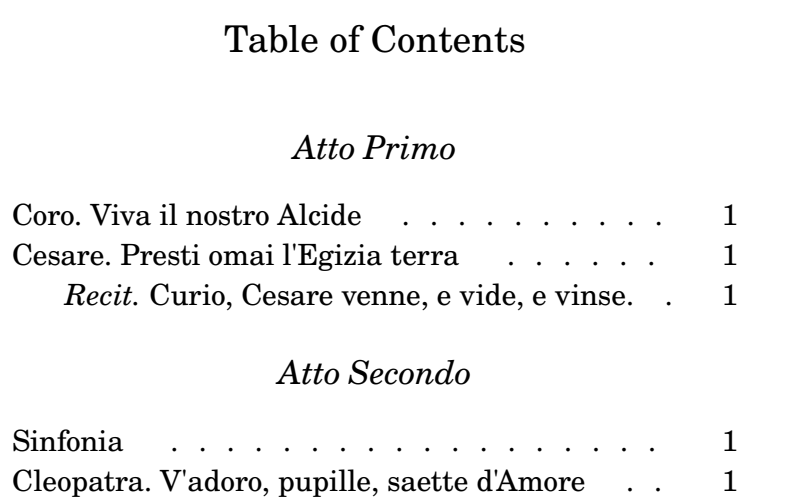
```
\paper {
```

```
  tocActMarkup = \markup \large \column {
    \hspace #1
    \fill-line { \null \italic \fromproperty #'toc:text \null }
    \hspace #1
  }
}
```

A custom music function (`tocAct`) is then created – which uses the new `tocActMarkup` markup definition, and allows to specify a label for each act.

```
tocAct =
  #(define-music-function (label text) (symbol? markup?)
     (add-toc-item! 'tocActMarkup text label))
```

Using these custom definitions and modifying some of the existing definitions, the source file could then be written as follows:



The previous example also demonstrates how to use the `\fill-with-pattern` markup command within the context of a table of contents.

See also

Installed Files: `ly/toc-init.ly`.

Predefined commands

`\table-of-contents`, `\tocItem`, `tocItemMarkup`, `tocTitleMarkup`, `tocFormatMarkup`, `tocIndentMarkup`.

22 Working with input files

22.1 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 22.2 [Different editions from one source], page 629.

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. These second-level `\include` statements are then interpreted relatively to the path of the file containing that command, which is convenient for multiple files located in the same subdirectory. For example, a general library, 'libA', may itself use subfiles which are `\included` by the entry file of that library, like this:

```
libA/
  libA.ly
  A1.ly
  A2.ly
  ...
```

then the entry file, `libA.ly`, will contain

```
\include "A1.ly"
\include "A2.ly"
...
```

Any `.ly` file can then include the entire library simply with

```
\include "~/libA/libA.ly"
```

However, that behavior can be changed globally by passing the command-line option `-drelative-includes=#f`, or by adding `#(ly:set-option 'relative-includes #f)` at the top of the main input file. In that case, each file will be included relatively to the location of the main file, regardless of where its `\include` statement is located. Complex file structures, that require to `\include` *both* files relative to the main directory and files relative to some other directory, may even be devised by setting `relative-includes` to `#f` or `#t` at appropriate places in the files.

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, change 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.

22.2 Different editions from one source

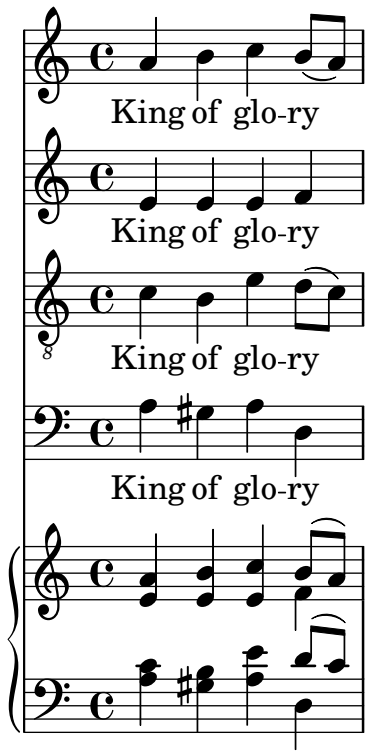
Several methods can be used to generate different versions of a score from the same music source. Variables are perhaps the most useful for combining lengthy sections of music and/or annotation. Tags are more useful for selecting one section from several alternative shorter sections of music, and can also be used for splicing pieces of music together at different points.

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.

22.2.1 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 Section 5.2.5 [Automatic part combining], page 233. Here is an example:

```
sopranoMusic = \relative { a'4 b c b8( a) }
altoMusic = \relative { e'4 e e f }
tenorMusic = \relative { c'4 b e d8( c) }
bassMusic = \relative { 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" {
      \partCombine \sopranoMusic \altoMusic
    }
    \new Staff = "LH" {
      \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 22.1 [Including LilyPond files], page 628.

22.2.2 Using tags

The `\tag` command marks a music expression with a name.

```
\tag #'foo { ... }
```

Expressions tagged in this way can be conviently manipulated as a whole.

Keeping and removing tagged music

The command `\keepWithTag` selects tagged music. On the other hand, `\removeWithTag` filters out tagged music.

Filter	**Result**
Tagged music preceded by `\keepWithTag #'`*name* or `\keepWithTag #'(`*name1 name2...*`)`	Untagged music and music tagged with any of the given tag names is included; music tagged with any other tag name is excluded.
Tagged music preceded by `\removeWithTag #'`*name* or `\removeWithTag #'(`*name1 name2...*`)`	Untagged music and music not tagged with any of the given tag names is included; music tagged with any of the given tag names 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 or a list of symbols such as `#'score` or `#'(violinI violinII)`, followed by a music expression. If *and only if* the symbols are valid LilyPond identifiers (alphabetic characters only, no numbers, underscores, or dashes, and different from any note names), the `#'` prefix may be omitted and, as a shorthand, a list of symbols can use the comma separator. For example, `\tag #'(violinI violinII)` can be written as `\tag violinI,violinII`. The same applies to the arguments of `\keepWithTag` and `\removeWithTag`.

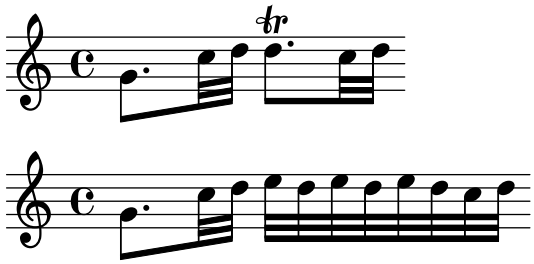
Tagging commands are music functions, thus they cannot be used to filter items that are not music expressions, such as `\book` or `\score` blocks.

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'8. c32 d
  \tag #'trills { d8.\trill }
  \tag #'expand { \*3 { e32 d } }
  c32 d
 }

\score {
  \keepWithTag #'trills \music
}
\score {
  \keepWithTag #'expand \music
```

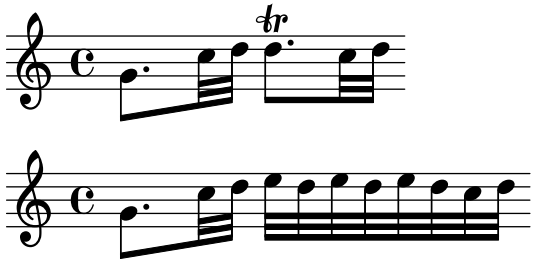
```
}
```



Alternatively, it is sometimes easier to exclude sections of music.

```
music = \relative {
  g'8. c32 d
  \tag #'trills { d8.\trill }
  \tag #'expand { \*3 { e32 d } }
  c32 d
 }

\score {
  \removeWithTag #'expand \music
}
\score {
  \removeWithTag #'trills \music
}
```



If tags mark alternatives that have non-zero duration, the alternatives are often conceptually simultaneous. In this case it is best to put the alternatives in a simultaneous music expression so that the music expression has the same duration no matter which tags are retained. This is especially important if you are using tags in combination with commands like `\cueDuring`.

```
outputTypeTag = "isScore"

firstInstrument = \relative c' {
  <<
    \tag #'isPart {
      \cueDuring "quoteSecondInstrument" #UP { r2 } }
    \tag #'isScore { r2 }
  >>
  e4 f |
  g4 a b c |
}
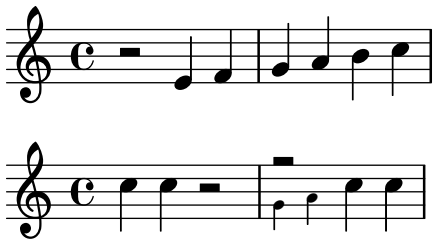
secondInstrument= \relative c'' {
  c4 c r2 |
  \cueDuring "quoteFirstInstrument" #DOWN { r2 }
```

```
  c4 c |
}

\addQuote quoteFirstInstrument \firstInstrument
\addQuote quoteSecondInstrument \secondInstrument

\new Staff {
  \keepWithTag \outputTypeTag \firstInstrument
}

\new Staff {
  \keepWithTag \outputTypeTag \secondInstrument
}
```



Tagged filtering can be applied to articulations, texts, etc., by prepending

```
-\tag #'your-tag
```
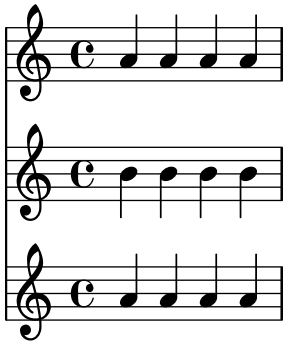
to an articulation. For example, the following defines a note with a conditional fingering indication and a note with a conditional annotation.

```
c1-\tag #'finger ^4
c1-\tag #'warn ^"Watch!"
```

Multiple tags may be placed on expressions with multiple \tag entries, or by combining multiple tags into one symbol list.

```
music = \relative c'' {
  \tag AA \tag AB { a4 a a a }
  \tag BB,both { b4 b b b }
}
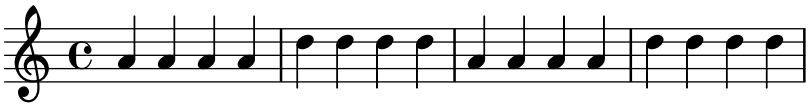
<<
  \keepWithTag AA \music
  \keepWithTag BB \music
  \keepWithTag AB \music
>>
```



Multiple \removeWithTag filters may be applied to a single music expression to remove several differently named tagged sections. Alternatively, you can use a single \removeWithTag with a list of tags.

```
music = \relative c'' {
  \tag AA { a4 a a a }
  \tag BB { b4 b b b }
  \tag CC { c4 c c c }
  \tag DD { d4 d d d }
}
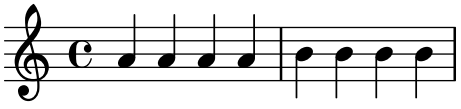
\new Voice {
  \removeWithTag BB \removeWithTag CC \music
  \removeWithTag BB,CC \music
}
```



Using two or more \keepWithTag filters on a single music expression causes *all* of the tagged sections to be removed, which is certainly not what you want: the first filter removes everything except the one named, and any subsequent filter removes the rest. Instead, use one \keepWithTag command with a list of multiple tags to remove all tagged sections that are not specified in that list. As an example, the following code prints music tagged with *violinI* and *violinII* but not tagged with *viola* or *cello*.

```
music = \relative c'' {
  \tag violinI { a4 a a a }
  \tag violinII { b4 b b b }
  \tag viola { c4 c c c }
  \tag cello { d4 d d d }
}

\new Staff {
  \keepWithTag violinI,violinII \music
}
```



Using tag groups

While \keepWithTag is convenient when dealing with *one* set of alternatives, the removal of music tagged with unrelated tags is problematic when using them for more than one purpose. In that case tags can be grouped with \tagGroup. The following code

```
\tagGroup violinI,violinII,viola,cello
```

puts the four tags into a single tag group. As a result, code like
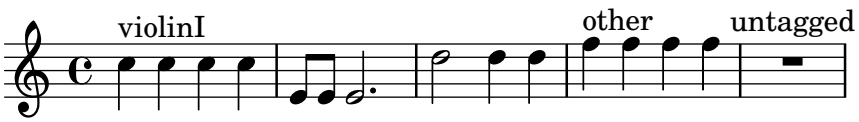
```
\keepWithTag violinI ...
```

now only shows music tagged from violinI's tag group; any music tagged with one of the other tags in the group is removed, as the following example demonstrates.

```
\tagGroup violinI,violinII
\tagGroup original,arranged

music = \relative {
  \tag violinI { c''4^"violinI" c c }
  \tag violinII { a2 a }
  \tag original { e8 e e2. }
  \tag arranged { d'2 d4 d }
  \tag other { f^"other" f f f }
  R1^"untagged"
}

\new Voice {
  \keepWithTag violinI \music
}
```



Note that individual tags cannot be members of more than one tag group.

It is possible to modify tag groups. There is an alternative way of defining them using `\tagGroupRef` to be able to reference the tag group later for modification. The call

```
tg = \tagGroupRef violinI,violinII
```

is essentially the same as

```
tg = #'(violinI violinII)
\tagGroup \tg
```

There are different possibilities to change existing tag groups.

- To undo the effects of every call of `\tagGroup` or `\tagGroupRef` you can just use `\resetTagGroups`.

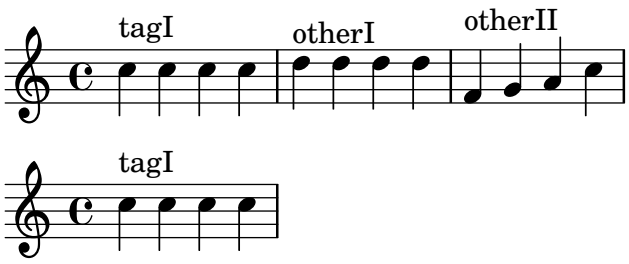
```
\tagGroup tagI,tagII

music = \relative {
  \tag tagI { c''4^"tagI" c c }
  \tag tagII { a2^"tagII" a }
  \tag otherI { d4^"otherI" d d }
  \tag otherII { f,4^"otherII" g a c }
}

\new Voice {
  \keepWithTag tagI \music
}

\resetTagGroups

\new Voice {
  \keepWithTag tagI \music
}
```



- If you want to reset one specific tag group you have to use \resetTagGroup with a given tag group specified by all of its tags. It is handy to use \tagGroupRef for the definition of the tag group.

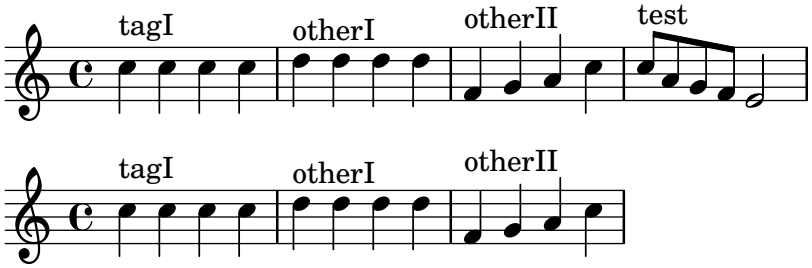
```
tgI = \tagGroupRef tagI,tagII
tgII = \tagGroupRef otherI,otherII

music = \relative {
  \tag tagI { c''4^"tagI" c c c }
  \tag tagII { a2^"tagII" a }
  \tag otherI { d4^"otherI" d d d }
  \tag otherII { f,4^"otherII" g a c }
  \tag test { c8^"test" a g f e2 }
}

% 'otherI', 'otherII' and 'test' are not in the same group
% as 'tagI' so they stay untouched.
\new Voice {
  \keepWithTag tagI \music
}

\resetTagGroup \tgI

% now only 'otherI' and 'otherII' are in a tag group,
% so '\keepWithTag tagI' filters 'tagII' and 'test'.
\new Voice {
  \keepWithTag tagI \music
}
```



- Another way of modifying an existing tag group is to add or remove single tags with the commands \addToTagGroup and \removeFromTagGroup.

```
tgI = \tagGroupRef tagI,tagII
tgII = \tagGroupRef otherI,otherII

music = \relative {
  \tag tagI { c''4^"tagI" c c c }
  \tag tagII { a2^"tagII" a }
```

```
  \tag otherI { d4^"otherI" d d d }
  \tag otherII { f,4^"otherII" g a c }
  \tag test { c8^"test" a g f e2 }
}

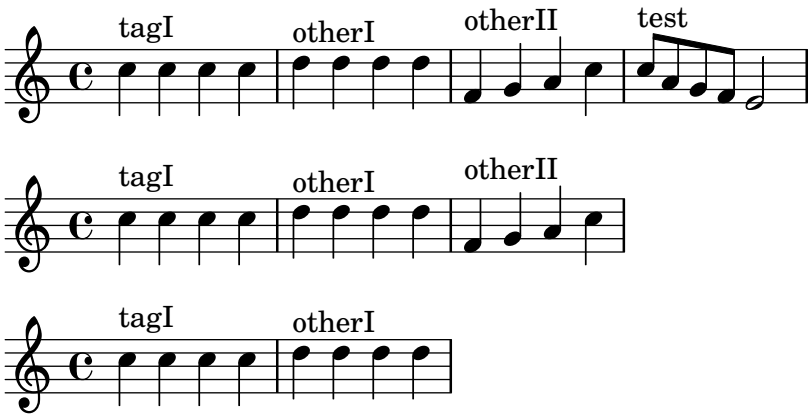
% 'otherI', 'otherII' and 'test' are not in the same group
% as 'tagI' so they stay untouched.
\new Voice {
  \keepWithTag tagI \music
}

\addToTagGroup \tgI test

% now 'tagI', 'tagII' and 'test' are in the same group
% so 'tagII' and 'test' are removed.
\new Voice {
  \keepWithTag tagI \music
}

\removeFromTagGroup \tgII otherII
\addToTagGroup \tgI otherII

% now 'otherII' is removed from 'tgII' and
% added to 'tgI' so only 'otherI' is in its
% own tag group and is not filtered.
\new Voice {
  \keepWithTag tagI \music
}
```



Prepending and appending to tagged music

Sometimes you want to splice some music at a particular place in an existing music expression. You can use \pushToTag and \appendToTag to add material at the front and end of various music constructs, respectively. The supported places are

sequential and simultaneous music
: If you tag an entire { . . . } or << . . . >> construct, you can add music expressions at its front or back.

chords
: If you tag a chord < . . . >, you can either add notes at its front or back, or articulations for the whole chord.

notes and rests
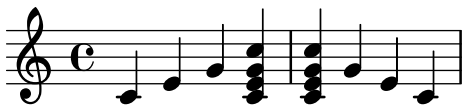
If you tag a note (also inside of a chord) or a rest, you can add articulations to the front or back of its existing articulations. Note that to add other notes, you have to put the note inside of a chord and tag the *chord* instead.

Also note that you cannot tag a single articulation and add to it since it isn't inherently a list. Instead, tag the note.

Both commands get a tag, the material to splice in at every occurrence of the tag, and the tagged expression.

```
music = { \tag #'here { \tag #'here <<c''>> } }

{
  \pushToTag #'here c'
  \pushToTag #'here e'
  \pushToTag #'here g' \music
  \appendToTag #'here c'
  \appendToTag #'here e'
  \appendToTag #'here g' \music
}
```
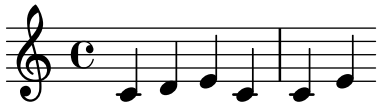


Replace tagged music

It is possible to replace tagged music with other music.

```
music = { c' \tag #'here { d' } e' }
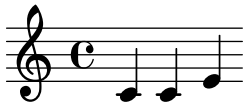
{
  \music
  \replaceWithTag #'here c' \music
}
```



Keep in mind that the tag is gone after the replacement, so you can only replace it once. Filtering is also not possible after replacement.

```
music = { c' \tag #'here { d' } e' }

{
  \removeWithTag #'here % does nothing because tag is replaced
  \replaceWithTag #'here c'
  \music
}
```



Tagging within markups

The \tag command can also be used within \markup. LilyPond provides markup commands \keep-with-tag, \remove-with-tag, \push-to-tag, \append-to-tag and \replace-with-tag in analogy to the corresponding commands for music expressions.

```
test = \markup {
  \tag #'AA a
  \tag #'BB b
  \tag #'CC c
}

\markup { \keep-with-tag #'BB \test }
\markup { \remove-with-tag #'BB \test }
\markup { \push-to-tag #'CC pre \test }
\markup { \append-to-tag #'CC post \test }
\markup { \replace-with-tag #'CC sub \test }
```

b

a c

a b pre c
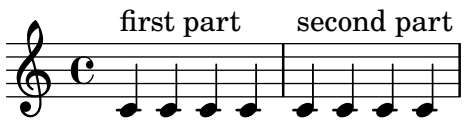
a b c post

a b sub

Music commands like \keepWithTag and \removeWithTag filter tags in \markup parts in the related music, too.

```
music = \relative {
  c'4^\markup { \tag #'one first \tag #'two second part } c c c
}

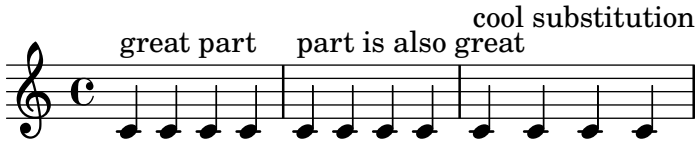
{
  \keepWithTag #'one \music
  \removeWithTag #'one \music
}
```



It is also possible to push and append something to the \markup of musical objects. However, we cannot use \pushToTag, \appendToTag or \replaceWithTag because they only insert some music, so we need commands like \pushToTagMarkup, \appendToTagMarkup or \replaceWithTagMarkup to insert markup.

```
music = \relative {
  c'4^\markup { \tag #'part part } c c c
}
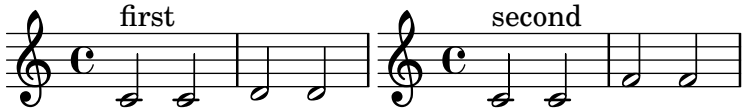
{
  \pushToTagMarkup #'part "great" \music
  \appendToTagMarkup #'part \markup { is also great } \music
  \replaceWithTagMarkup #'part \markup { cool substitution } \music
}
```



The filtering of tags also works for music embedded within \score blocks in markup commands.

```
music = \relative {
  c'2^\markup { \tag #'first first \tag #'second second } c
  \tag #'first { d d }
  \tag #'second { f f }
}

\markup {
  \keep-with-tag #'first \score { \music }
  \remove-with-tag #'first \score { \music }
}
```



Tagging lists of markup need special attention. While the filter functions work as expected, commands like \push-to-tag, \append-to-tag and \replace-with-tag do not.

```
\markup {
  \remove-with-tag #'test { a \tag #'test { b c } d }
}
\markup {
  \push-to-tag #'test "twice" { a \tag #'test { b c } d }
}
\markup {
  \replace-with-tag #'test "twice" { a \tag #'test { b c } d }
}
```

a d

a twice b twice c d

a twice twice d

The reason for this behavior is that LilyPond resolves

```
\tag #'test { b c }
```

internally to

```
\tag #'test b
\tag #'test c
```

and thus the given text is inserted twice. For tagging a list to insert or append something before or after the whole list the command `\tag-list` should be used.

```
\markup {
  \push-to-tag #'test "once" { a \tag-list #'test { b c } d }
}
```

a once b c d

See also

Learning Manual: Section "Organizing pieces with variables" in *Learning Manual*.

Notation Reference: Section 5.2.5 [Automatic part combining], page 233, Section 22.1 [Including LilyPond files], page 628.

Known issues and warnings

Calling `\relative` on a music expression obtained by filtering music through `\keepWithTag` or `\removeWithTag` might cause the octave relations to change, as only the pitches actually remaining in the filtered expression are considered. Applying `\relative` first, before `\keepWithTag` or `\removeWithTag`, avoids this danger as `\relative` then acts on all the pitches as input.

Even when using tag groups, commands like `\keepWithTag` are not commutative. Code like

```
\keepWithTag violinI,original \someMusic
```

does not produce the same output as

```
\keepWithTag violinI \keepWithTag original \someMusic
```

22.2.3 Using global settings

Global settings can be included from a separate file:

```
lilypond -dinclude-settings=MY_SETTINGS.ly MY_SCORE.ly
```

Groups of settings such as page size, font or type face can be stored in separate files and loaded with several `-dinclude-settings` options. This allows different editions from the same score as well as standard settings to be applied to many scores, simply by specifying the proper settings file.

This technique also works well with the use of style sheets, as discussed in Section "Style sheets" in *Learning Manual*.

See also

Learning Manual: Section "Organizing pieces with variables" in *Learning Manual*, Section "Style sheets" in *Learning Manual*.

Notation Reference: Section 22.1 [Including LilyPond files], page 628.

22.3 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*.

22.3.1 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
     (arg1 arg2 ...)
     (type1? type2? ...)
   #{
     ...music...
   #})
```

where

argN	The *n*th argument.
typeN?	A Scheme *type predicate* for which *argN* must return `#t`.
...music...	Normal LilyPond input, using '$' (in places where only LilyPond constructs are allowed) or '#' (to use it as a Scheme value or music function argument or music inside of music lists) to reference arguments (e.g., '`#arg1`').

The list of type predicates is required. Some of the most common type predicates used in music functions are:

boolean?

cheap-list? *(use instead of* 'list?' *for faster processing)*

ly:duration?

ly:music?

ly:pitch?

markup?

number?

pair?

string?

symbol?

For a list of available type predicates, see Section B.25 [Predefined type predicates], page 974. User-defined type predicates are also allowed.

See also

Notation Reference: Section B.25 [Predefined type predicates], page 974.

Extending LilyPond: Section "Music functions" in *Extending*.

Installed Files: `lily/music-scheme.cc`, `scm/c++.scm`, `scm/lily.scm`.
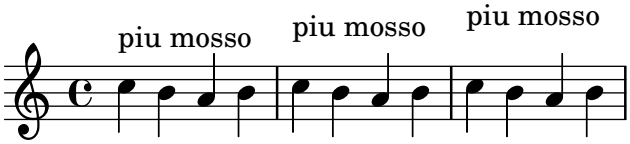
22.3.2 Substitution function examples

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` grob:

```
padText =
#(define-music-function
     (padding)
     (number?)
   #{
     \once \override TextScript.padding = #padding
   #})
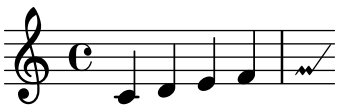
\relative {
  c''4^"piu mosso" b a b
  \padText 1.8
  c4^"piu mosso" b a b
  \padText 2.6
  c4^"piu mosso" b a b
}
```



In addition to numbers, we can use music expressions such as notes for arguments to music functions:

```
custosNote =
#(define-music-function
     (note)
     (ly:music?)
   #{
     \tweak NoteHead.stencil #ly:text-interface::print
     \tweak NoteHead.text
        \markup \musicglyph "custodes.mensural.u0"
     \tweak Stem.stencil ##f
     #note
   #})

\relative { c'4 d e f \custosNote g }
```
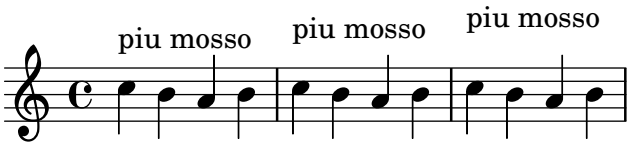


Both of those functions are simple single expressions where only the last element of a function call or override is missing. For those particular function definitions, there is a simpler alternative syntax, namely just writing out the constant part of the expression and replacing its final missing element with \etc:

```
padText =
  \once \override TextScript.padding = \etc
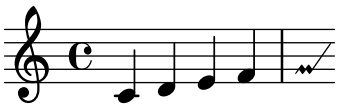
\relative {
  c''4^"piu mosso" b a b
  \padText 1.8
  c4^"piu mosso" b a b
  \padText 2.6
```

```
  c4^"piu mosso" b a b
}
```




```
custosNote =
  \tweak NoteHead.stencil #ly:text-interface::print
  \tweak NoteHead.text
     \markup \musicglyph "custodes.mensural.u0"
  \tweak Stem.stencil ##f
  \etc

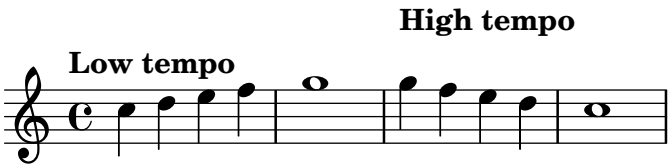
\relative { c'4 d e f \custosNote g }
```



Substitution functions with multiple arguments can be defined:

```
tempoPadded =
#(define-music-function
     (padding tempotext)
     (number? markup?)
   #{
     \once \override Score.MetronomeMark.padding = #padding
     \tempo \markup { \bold #tempotext }
   #})

\relative {
  \tempo \markup { "Low tempo" }
  c''4 d e f g1
  \tempoPadded 4.0 "High tempo"
  g4 f e d c1
}
```
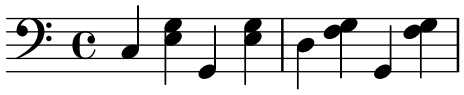


22.3.3 How to prevent sharing of music expressions

When writing music functions, it is important to abide by a rule: the same music expressions must not be shared in several places. As an example, here is a problematic function:

```
simpleAccompaniment =
#(define-music-function
   (bass-1 bass-2 chord) (ly:music? ly:music? ly:music?)
   #{
     #bass-1 #chord #bass-2 #chord
   #})
```
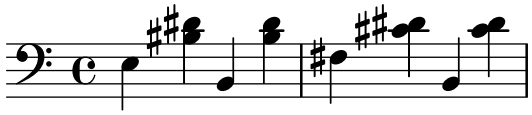
```
{
  \clef bass
  \simpleAccompaniment c g, <e g>
  \simpleAccompaniment d g, <f g>
}
```



The problem with this function becomes clear if the result is transposed:

```
simpleAccompaniment =
#(define-music-function
   (bass-1 bass-2 chord) (ly:music? ly:music? ly:music?)
   #{
     #bass-1 #chord #bass-2 #chord
   #})

\transpose c e {
  \clef bass
  \simpleAccompaniment c g, <e g>
  \simpleAccompaniment d g, <f g>
}
```
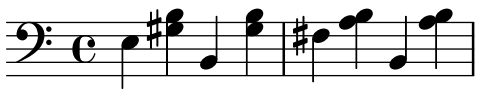


While the bass notes are correct, the chord is not transposed properly – in fact, it is being transposed twice. The reason for this is that the music expression *chord* was used twice in the result of the function, without copying it. Functions such as `\transpose` modify the music object directly (in the case of `\transpose`, the pitches are changed). If the same music object is reused, modifications made in one place where it is used affect both places, since they hold the same object. In this case, `\transpose` encounters the object twice and transposes it twice.

One way to fix this function is to use '`$`' instead of '`#`' to reference the variables, which makes a copy. The difference between '`#`' and '`$`' is detailed in Section "LilyPond Scheme syntax" in *Extending*.

```
simpleAccompaniment =
#(define-music-function
   (bass-1 bass-2 chord) (ly:music? ly:music? ly:music?)
   #{
     $bass-1 $chord $bass-2 $chord
   #})

\transpose c e {
  \clef bass
  \simpleAccompaniment c g, <e g>
  \simpleAccompaniment d g, <f g>
}
```

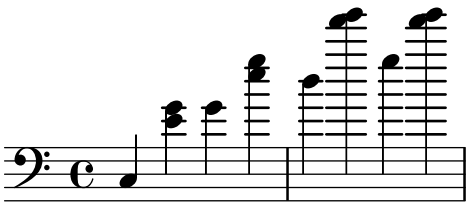


22.3.4 Substitution functions and relative octave entry

When `\relative` is applied to a music expression, it traverses it searching for pitched notes, and modifies the pitches in the order they are found, changing the octave of each pitch according to its octave marks (‘'’ and ‘,’) and the previous pitch. When writing substitution functions, this may lead to the situation that a music expression is ‘relativized’ in a surprising way because the output of the function uses the parameters several times and/or in a different order. Consider this function and how its output reacts to `\relative`:

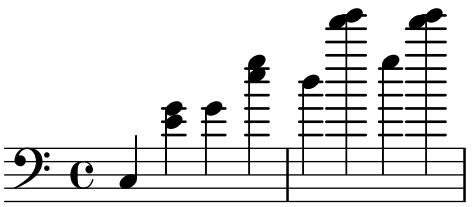
```
simpleAccompaniment =
#(define-music-function
   (bass-1 bass-2 chord) (ly:music? ly:music? ly:music?)
   #{
     $bass-1 $chord $bass-2 $chord
   #})

\relative {
  \clef bass
  \simpleAccompaniment c g <e' g>
  \simpleAccompaniment d g, <f' g>
}
```



In this example, the output is the same as that of
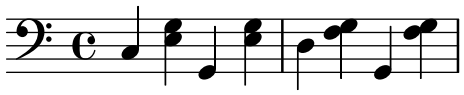
```
\relative {
  \clef bass
  c <e' g> g <e' g>
  d <f' g>  g, <f' g>
}
```



However, this is not the desired output when using the `\simpleAccompaniment` function. The pitch `g,` is relative to the first note of the chord that precedes it, `<e' g>`, although it comes after `c` in the input. Clearly, the pitches should be made relative according to the order in which they are input when using the function, not in the order they appear in the output of the function. This can be achieved using the `make-relative` Scheme macro. Its arguments are: a list of variables, a reference expression, and a main music expression. The reference expression is intended to be a mock-up of how the variables were entered in the input. Most of the time, it can be a simple expression made with `#{ ... #}` containing the variables in order. Beware *not* to make copies in the reference expression (in particular, use ‘#’, not ‘$’). The example above can be fixed using `make-relative` in this way:

```
simpleAccompaniment =
#(define-music-function
   (bass-1 bass-2 chord) (ly:music? ly:music? ly:music?)
   (make-relative
    (bass-1 bass-2 chord)
    #{ #bass-1 #bass-2 #chord #}
    #{ $bass-1 $chord $bass-2 $chord #}))

\relative {
  \clef bass
  \simpleAccompaniment c g <e' g>
  \simpleAccompaniment d g, <f' g>
}
```



22.4 Special characters

22.4.1 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.
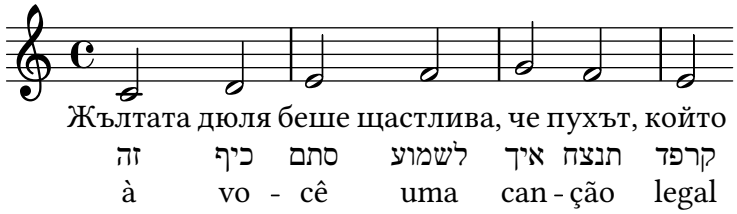
LilyPond does not perform any input encoding conversions. This means that any text, be it title, lyric text, or musical instruction containing non-ASCII characters, must be encoded in UTF-8. The easiest way to enter such text is by using a Unicode-aware editor and saving the file with UTF-8 encoding. 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:



22.4.2 Unicode

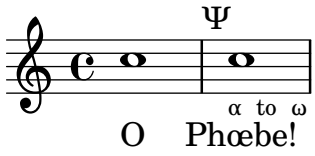
To enter a single character for which the Unicode code point is known but which is not available in the editor being used, use either `\char ##xhhhh` or `\char #dddd` within a `\markup` block, where `hhhh` is the hexadecimal code for the character required and `dddd` is the corresponding decimal value. 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, `https://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 text mark, as articulation text, in lyrics and as stand-alone text below the score:

```
\score {
  \relative {
    c''1
    \textMark \markup { \char ##x03A8 }
    c1_\markup { \tiny { \char ##x03B1 " to " \char ##x03C9 } }
  }
  \addlyrics { O \markup { \concat { Ph \char ##x0153 be! } } }
}
\markup { "Copyright 2008--2026" \char ##x00A9 }
```



Copyright 2008--2026 ©

To enter the copyright sign in the copyright notice use:

```
\header {
  copyright = \markup { \char ##x00A9 "2008" }
}
```

22.4.3 ASCII aliases

A list of ASCII aliases for special characters can be included:

```
\paper {
  #(include-special-characters)
}

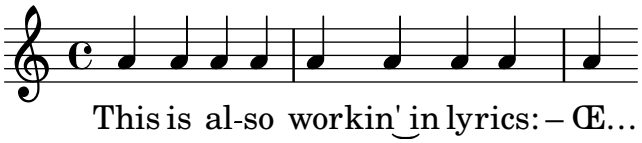
\markup "&flqq; &ndash; &OE;uvre incomplète&hellip; &frqq;"

\score {
  \new Staff { \*9 a'4 }
  \addlyrics {
    This is al -- so wor -- kin'~in ly -- rics: &ndash;_&OE;&hellip;
  }
```

```
}

\markup \column {
  "The replacement can be disabled:"
  "&ndash; &OE; &hellip;"
  \override #'(replacement-alist . ()) "&ndash; &OE; &hellip;"
}
```

« – Œuvre incomplète… »



The replacement can be disabled:
– Œ …
– &OE; …

You can also make your own aliases, either globally:

```
\paper {
  #(add-text-replacements!
    '(("100" . "hundred")
      ("dpi" . "dots per inch")))
}
\markup "A 100 dpi."
```

A hundred dots per inch.

or locally:

```
\markup \replace #'(("100" . "hundred")
                    ("dpi" . "dots per inch")) "A 100 dpi."
```

A hundred dots per inch.

The replacement is not necessarily a string; it can be an arbitrary markup. On the syntax level, this requires using Scheme quasi-quoting syntax, with a backtick ‘`’ instead of a quote ‘'’ to write the alist.

```
\markup \replace
  #`(("2nd" . ,#{ \markup \concat { 2 \super nd } #})) "2nd time"
```

2nd time

Aliases themselves are not further processed for replacements.

See also

Notation Reference: Section B.12 [List of special characters], page 927.

Installed Files: `ly/text-replacements.ly`.

23 Controlling output

23.1 Extracting fragments of music

It is possible to output one or more fragments of a score. To do so, use the `clip-regions` variable within the `\layout` or `\paper` block to define a list of explicit location of the music to be extracted, then execute `lilypond` with the `-dclip-systems` option.

```
\layout {
  clip-regions
  = #(list (cons (make-rhythmic-location 5 1 2)
                 (make-rhythmic-location 7 3 4)))
}
```

This example extracts a single fragment of the input file starting after a half note duration in fifth measure (`5 1 2`) and ending after the third quarter note in the seventh measure (`7 3 4`).

Additional fragments can be extracted by adding more pairs of `make-rhythmic-location` entries to the `clip-regions` list.

If system starts and ends are included, they include extents of the `System` grob, e.g., instrument names.

Grace notes at the end point of the region are not included.

Each music fragment gets written as a separate file. The extracted music is output as if it had been literally 'cut' from the original printed score, so if a fragment runs over one or more lines, a separate output file for each line is generated. Assuming that the above example covers two lines in the PDF output of input file `foo.ly`, the output files for the music fragments would be called `foo-from-5.1.2-to-7.3.4-clip.pdf` and `foo-from-5.1.2-to-7.3.4-clip-1.pdf`.

See also

Notation Reference: Section 27.1 [The `\layout` block], page 683.

Application Usage: Section "Command-line usage" in *Application Usage*.

23.2 Skipping corrected music

When entering or copying music, usually only the music near the end (where new notes are being added) 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 defining a special variable at the source file's top level, as follows:
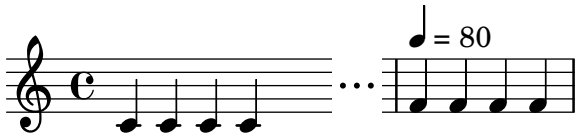
```
showLastLength = R1*5
\score { ... }
```

In this instance, nothing will be rendered but the last five 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 that has already been typeset (for example 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. As a property of the `Score` context, it affects all voices and staves; see Section 33.1.2 [`Score` – the master of all contexts], page 736.

This property is also used to control output to the MIDI file. If some event in the skipped section alters some of its context properties, for example a tempo or instrument change, then that new setting will take effect only at the point in time where `skipTypesetting` is disabled again:

```
\relative c' {
  c4 c c c
  \set Score.skipTypesetting = ##t
  d4 d d d
  \tempo 4 = 80
  e4 e e e
  \set Score.skipTypesetting = ##f
  f4 f f f
}
```



Predefined commands

showLastLength, showFirstLength.

See also

Notation Reference: Chapter 33 [Interpretation contexts], page 736, Section 33.1.2 [Score – the master of all contexts], page 736.

Internals Reference: Section "Tunable context properties" in *Internals Reference*.

23.3 Alternative output formats

The default output formats for the printed score are Portable Document Format (PDF) and PostScript (PS). Portable Network Graphics (PNG), Scalable Vector Graphics (SVG) and Encapsulated PostScript (EPS) output formats are available through command-line options, see Section "Basic command-line options for LilyPond" in *Application Usage*.

23.3.1 SVG Output

SVG output can optionally contain metadata for graphical objects (grobs) like note heads, rests, etc. This metadata can be standard SVG attributes like id and class, or non-standard custom attributes. Specify the attributes and their values by overriding a grob's output-attributes property with a Scheme association list (alist). The values can be numbers, strings, or symbols. For example:

```
{
  \once \override NoteHead.output-attributes =
  #'((id . 123)
     (class . "this that")
     (data-whatever . something))
  c
}
```

The input above will produce the following <g> (group) tag in the SVG file:

```
<g id="123" class="this that" data-whatever="something">
  ...NoteHead grob SVG elements...
</g>
```

The <g> tag contains all of the SVG elements for a given grob. (Some grobs generate multiple SVG elements.) In SVG syntax the data- prefix is used for non-standard custom metadata attributes.

23.4 Embedding files in PDF output

Command-line option `-dembed-source-code` makes LilyPond embed all (user) source files needed for compilation in the final output PDF (see Section "Advanced command-line options for LilyPond" in *Application Usage*); a PDF viewer can then extract these attachments for further use.

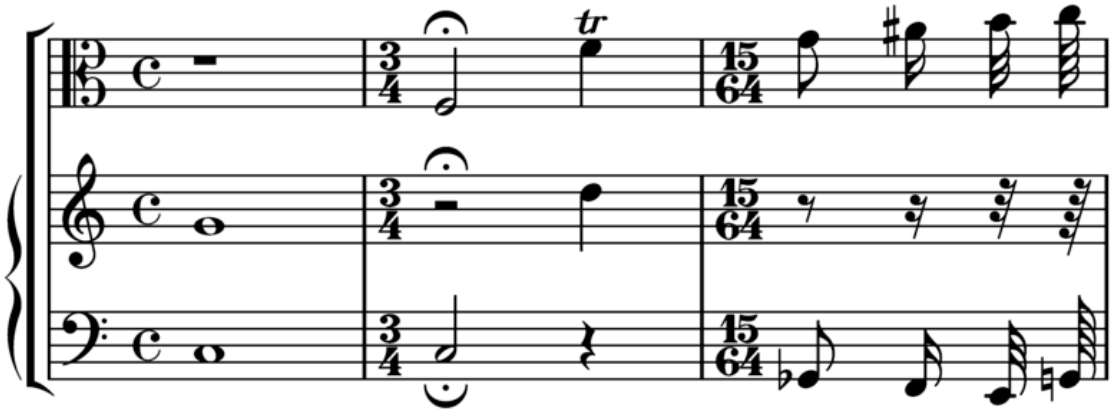
In a similar vein it is possible to embed arbitrary files in the PDF output with function `ly:note-extra-source-file` (see Section "Scheme functions" in *Internals Reference*).

23.5 Replacing the notation font
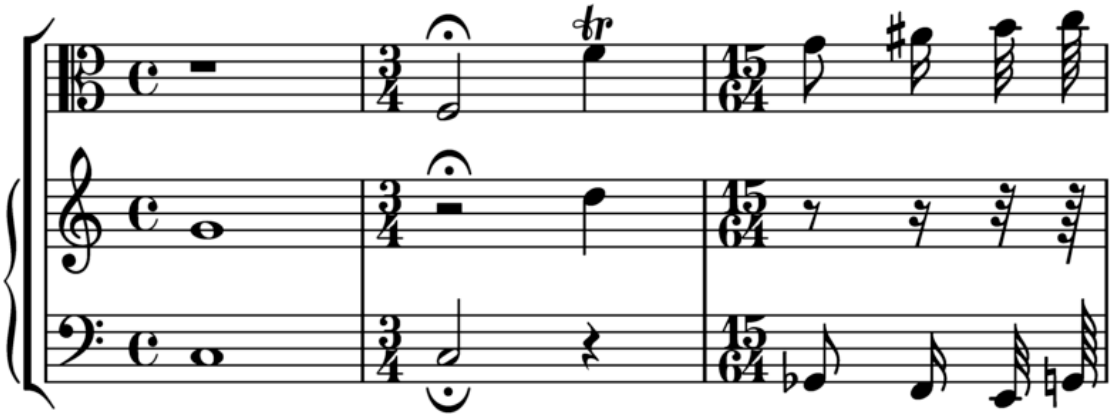
Gonville is an alternative set of glyphs to *Feta* – part of the Emmentaler font – and used in LilyPond. They can be downloaded from:

`http://www.chiark.greenend.org.uk/~sgtatham/gonville/` (`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 glyphs:



Installation Instructions

- Download and extract the font files.
- Copy[1] the files

  `gonville-11.otf`

[1] Currently it is necessary to repeat these steps after installing a new LilyPond version. If you are running the `lilypond` binary directly from the build directory, see Section "Replacing the notation fonts in development versions" in *Contributor's Guide* for more information.

```
gonville-13.otf
gonville-14.otf
gonville-16.otf
gonville-18.otf
gonville-20.otf
gonville-23.otf
gonville-26.otf
gonville-brace.otf
```

to directory .../share/lilypond/*X.Y.Z*/fonts/otf.

- If you have gonville-*.svg files, copy them to directory .../share/lilypond/*X.Y.Z*/fonts/svg.

For more information, see Section "Other sources of information" in *Learning Manual*.

Note: gonville-*.otf files are for the ps and cairo backend (for PDF and PostScript outputs, as well as all output formats when using the Cairo backend). gonville-*.svg files are for the svg backend. For more information, see Section "Advanced command-line options for LilyPond" in *Application Usage*.

The following code changes the notation font to the Gonville font.

```
\paper {
  property-defaults.fonts.music = "gonville"
}
```

For more information, see Section 8.3.5 [Changing fonts], page 341.

See also

Learning Manual: Section "Other sources of information" in *Learning Manual*.

Notation Reference: Section B.8 [The Emmentaler font], page 907, Section 8.3.5 [Changing fonts], page 341.

Known issues and warnings

Gonville cannot be used to typeset 'Ancient Music' notation and it is likely newer glyphs in later releases of LilyPond may not exist in the Gonville font family. Please refer to the author's website for more information on these and other specifics, including licensing of Gonville.

Other notation fonts

If you have other notation fonts like *fontname*-*.otf and *fontname*-*.svg, you can use them in the same way as Gonville.

That is, copy the *fontname*-*.otf files to .../share/lilypond/*X.Y.Z*/fonts/otf. If you have *fontname*-*.svg files, copy them to .../share/lilypond/*X.Y.Z*/fonts/svg.

Note: At the moment, LilyPond expects the font file names to have the following suffixes, all of which must be present in the above installation folder(s) to work properly: -11, -13, -14, -16, -18, -20, -23, -26, -brace. For example, emmentaler-11.otf, emmentaler-20.svg, etc.

The following code changes the notation font to the *fontname* font.

```
\paper {
  % font file name without suffix and extension
  property-defaults.fonts.music = "fontname"
}
```

24 Creating MIDI output

LilyPond can produce files that conform to the MIDI (Musical Instrument Digital Interface) standard and so allow for the checking of the music output aurally (with the help of an application or device that understands MIDI). Listening to MIDI output may also help in spotting errors such as notes that have been entered incorrectly or are missing accidentals and so on.

MIDI files do not contain sound (like AAC, MP3 or Vorbis files) but require additional software to produce sound from them.

24.1 Supported notation for MIDI

The following musical notation can be used with LilyPond's default capabilities to produce MIDI output;

- Breath marks
- Chords entered as chord names
- Crescendi, decrescendi over multiple notes. The volume is altered linearly between the two extremes
- Dynamic markings from `ppppp` to `fffff`, including `mp`, `mf` and `sf`
- Lyrics
- Markers: rehearsal marks, segni, coda marks, and section labels
- Microtones but *not* microtonal chords. A MIDI player that supports pitch bending will also be required.
- Pitches
- Rhythms entered as note durations, including tuplets
- 'Simple' articulations; staccato, staccatissimo, accent, marcato and portato
- Tempo changes using the `\tempo` function, including for fractional metronome values
- Ties
- Tremolos that are *not* entered with a ':[*number*]' value

Panning, balance, expression, reverb and chorus effects can also be controlled by setting context properties, see Section 24.8 [Context properties for MIDI effects], page 664.

When combined with the `articulate` script the following, additional musical notation can be output to MIDI;

- Appoggiaturas. These are made to take half the value of the note following (without taking dots into account). For example;

  ```
  \appoggiatura c8 d2.
  ```

  The c will take the value of a crotchet.
- Ornaments (i.e., mordents, trills and turns et al.)
- Rallentando, accelerando, ritardando and a tempo
- Slurs, including phrasing slurs
- Tenuto

See Section 24.9 [Enhancing MIDI output], page 665.

24.2 Unsupported notation for MIDI

The following items of musical notation cannot be output to MIDI;

- Articulations other than staccato, staccatissimo, accent, marcato and portato
- Crescendi and decrescendi over a *single* note
- Fermata
- Figured bass
- Glissandi
- Falls and doits
- Microtonal chords
- Rhythms entered as annotations, e.g., swing
- Tempo changes without `\tempo` (e.g., entered as annotations)
- Tremolos that *are* entered with a ':[*number*]' value

24.3 The MIDI block

To create a MIDI output file from a LilyPond input file, insert a `\midi` block, which can be empty, within the `\score` block.[1]

```
\score {
  ... music ...
  \layout { }
  \midi { }
}
```

> **Note:** A `\score` block that contains music and a `\midi` block but no `\layout` block produces a MIDI output file only. No notation gets printed.
>
> A `\midi` block at the top level can be used to change MIDI settings globally; however, the generation of an actual MIDI file only happens when a `\midi` block is part of a `\score` block.
>
> Similarly, a `\layout` block at the top level affects layout settings globally but does not influence whether printed output is produced or not.

The default output file extension (`.midi`) can be changed by using the `-dmidi-extension` option with the `lilypond` command:

```
lilypond -dmidi-extension=mid MyFile.ly
```

Alternatively, add the following Scheme expression before the start of either the `\book`, `\bookpart` or `\score` blocks. See Section 20.5 [File structure], page 596.

```
#(ly:set-option 'midi-extension "mid")
```

See also

Notation Reference: Section 20.5 [File structure], page 596, Section 20.4 [Output file names], page 595, Section 21.3 [Creating output file metadata], page 612.

Installed Files: `scm/midi.scm`.

[1] Note that there also exists a markup command called `\score` that doesn't produce MIDI output, even if a `\midi` block is present. See [Scores within markup], page 860.

Known issues and warnings

There are fifteen MIDI channels available and one additional channel (#10) for drums. Staves are assigned to channels in sequence, so a score that contains more than fifteen staves will result in the extra staves sharing (but not overwriting) the same MIDI channel. This may be a problem if the sharing staves have conflicting, channel-based, MIDI properties – such as different MIDI instruments – set.

24.4 Controlling MIDI dynamics

It is possible to control the overall MIDI volume, the relative volume of dynamic markings and the relative volume of different instruments.

Dynamic marks translate automatically into volume levels in the available MIDI volume range whereas crescendi and decrescendi vary the volume linearly between their two extremes. It is possible to control the relative volume of dynamic markings, and the overall volume levels of different instruments.

24.4.1 Dynamic marks in MIDI

Only the dynamic markings from `ppppp` to `fffff`, including `mp`, `mf` and `sf` have values assigned to them. This value is then applied to the value of the overall MIDI volume range to obtain the final volume included in the MIDI output for that particular dynamic marking. The default fractions range from 0.25 for *ppppp* to 0.95 for *fffff*. The complete set of dynamic marks and their associated fractions can be found in `ly/midi-init.ly`.

Selected snippets

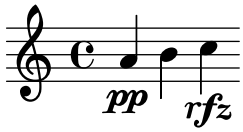
Creating custom dynamics in MIDI output

The following example shows how to create a dynamic marking, not included in the default list, and assign a specific value to it so that it affects MIDI output.

The dynamic mark `\rfz` gets value `0.9`.

```
#(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 {
        a'4\pp b c-\rfz
      }
    }
  }
  \layout {}
  \midi {}
}
```



Installed Files: `ly/script-init.ly` `scm/midi.scm`.

Snippets: Section "MIDI" in *Snippets*.

Internals Reference: Section "Dynamic_performer" in *Internals Reference*.

24.4.2 Setting 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 at the start of a voice and on dynamic marks. 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 {
      \set Staff.midiInstrument = "flute"
      ... music ...
    }
    \new Staff {
      \set Staff.midiInstrument = "clarinet"
      ... music ...
    }
  >>
  \midi {
    \context {
      \Score
      midiMinimumVolume = 0.2
      midiMaximumVolume = 0.5
    }
  }
}
```

Simple MIDI instrument equalization can be achieved by setting `midiMinimumVolume` and `midiMaximumVolume` properties within the `Staff` context.

```
\score {
  \new Staff {
    \set Staff.midiInstrument = "flute"
    \set Staff.midiMinimumVolume = 0.7
    \set Staff.midiMaximumVolume = 0.9
    ... music ...
  }
  \midi { }
}
```

For scores with multiple staves and multiple MIDI instruments, the relative volumes of each instrument can be set individually;

```
\score {
  <<
    \new Staff {
      \set Staff.midiInstrument = "flute"
      \set Staff.midiMinimumVolume = 0.7
      \set Staff.midiMaximumVolume = 0.9
```

```
      ... music ...
    }
    \new Staff {
      \set Staff.midiInstrument = "clarinet"
      \set Staff.midiMinimumVolume = 0.3
      \set Staff.midiMaximumVolume = 0.6
      ... music ...
    }
  >>
  \midi { }
}
```

In this example the volume of the clarinet is reduced relative to the volume of the flute.

If these volumes properties are not set then LilyPond still applies a 'small degree' of equalization to certain instruments. See `scm/midi.scm`.

Installed Files: `scm/midi.scm`.

See also

Notation Reference: Chapter 27 [Score layout], page 683.

Internals Reference: Section "Dynamic_performer" in *Internals Reference*.

Selected snippets

Replacing default MIDI instrument equalization

The default MIDI instrument equalizer can be replaced by setting the `instrumentEqualizer` property in the `Score` context to a user-defined Scheme procedure that uses a MIDI instrument name as its argument along with a pair of fractions indicating the minimum and maximum volumes, respectively, to be applied to that specific instrument.

The following example sets the minimum and maximum volumes for flute and clarinet.
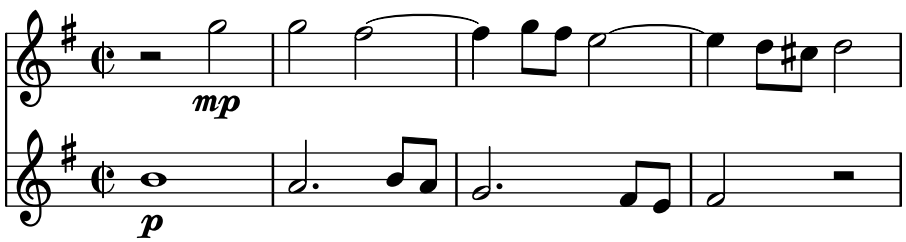
```
#(define my-instrument-equalizer-alist '())

#(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 {
        r2 g''\mp g fis~
```

```
        4 g8 fis e2~
        4 d8 cis d2
      }
    }
    \new Staff {
      \key g \major
      \set Staff.midiInstrument = "clarinet"
      \new Voice \relative {
        b'1\p a2. b8 a
        g2. fis8 e
        fis2 r
      }
    }
  >>
  \layout { }
  \midi {  }
}
```



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.

24.4.3 Setting MIDI block properties

The `\midi` block can contain context rearrangements, new context definitions or code that sets the values of certain properties.

```
\score {
  ... music ...
  \midi {
    \tempo 4 = 72
  }
}
```

Here the tempo is set to 72 quarter note beats per minute. The tempo mark in the `\midi` block will not appear in the printed score. Although any other `\tempo` indications specified within the `\score` block will also be reflected in the MIDI output.

In a `\midi` block the `\tempo` command is setting properties during the interpretation of the music and in the context of output definitions; so it is interpreted *as if* it were a context modification.

Context definitions follow the same syntax as those in a `\layout` block;

```
\score {
  ... music ...
  \midi {
    \context {
      \Voice
```

```
        \remove Dynamic_performer
      }
    }
  }
```

This example removes the effect of dynamics from the MIDI output. Note: LilyPond's translation modules used for sound are called 'performers'.

See also

Learning Manual: Section "Other sources of information" in *Learning Manual*.

Notation Reference: Chapter 3 [Expressive marks], page 156, Chapter 27 [Score layout], page 683.

Installed Files: `ly/performer-init.ly`.

Snippets: Section "MIDI" in *Snippets*.

Internals Reference: Section "Dynamic_performer" in *Internals Reference*.

Known issues and warnings

Some MIDI players do not always correctly handle tempo changes in the midi output.

Changes to the `midiInstrument`, as well as some MIDI options, at the *beginning* of a staff may appear twice in the MIDI output.

24.5 Using MIDI instruments

MIDI instruments are set using the `midiInstrument` property within a `Staff` context.

```
\score {
  \new Staff {
    \set Staff.midiInstrument = "glockenspiel"
    ... music ...
  }
  \midi { }
}
```

or

```
\score {
  \new Staff \with {midiInstrument = "cello"} {
    ... music ...
  }
  \midi { }
}
```

If the instrument name does not match any of the instruments listed in the 'MIDI instruments' section, the `acoustic grand` instrument will be used instead. See Section B.6 [MIDI instruments], page 901.

See also

Learning Manual: Section "Other sources of information" in *Learning Manual*.

Notation Reference: Section B.6 [MIDI instruments], page 901, Chapter 27 [Score layout], page 683.

Installed Files: `scm/midi.scm`.

Known issues and warnings

Percussion instruments that are notated in a `DrumStaff` context will be output, correctly, to MIDI channel 10 but some pitched, percussion instruments like the xylophone, marimba, vibraphone or timpani, are treated as "normal" instruments so the music for these should be entered in a `Staff` (not `DrumStaff`) context to obtain correct MIDI output. A full list of `channel 10 drum-kits` entries can be found in `scm/midi.scm`. See Section "Other sources of information" in *Learning Manual*.

24.6 Using repeats with MIDI

Repeats can be represented in the MIDI output by applying the `\unfoldRepeats` command.

```
\score {
  \unfoldRepeats {
    \repeat tremolo 8 { c'32 e' }
    \repeat percent 2 { c''8 d'' }
    \repeat volta 2 { c'4 d' e' f' }
    \alternative {
      \volta 1 { g' a' a' g' }
      \volta 2 { f' e' d' c' }
    }
  }
  \midi { }
}
```

In order to restrict the effect of `\unfoldRepeats` to the MIDI output only, while also generating printable scores, it is necessary to make *two* `\score` blocks; one for MIDI (with unfolded repeats) and one for the notation (with volta, tremolo, and percent repeats);

```
\score {
  ... music ...
  \layout { }
}
\score {
  \unfoldRepeats {
    ... music ...
  }
  \midi { }
}
```

When using multiple voices, each of the voices must contain completely unfolded repeats for correct MIDI output.

See also

Notation Reference: Chapter 4 [Repeats], page 189.

24.7 MIDI channel mapping

When generating a MIDI file from a score, LilyPond will automatically assign every note in the score to a MIDI channel, the one on which it should be played when it is sent to a MIDI device. A MIDI channel has a number of controls available to select, for example, the instrument to be used to play the notes on that channel, or to request the MIDI device to apply various effects to the sound produced on the channel. At all times, every control on a MIDI channel can have only a single value assigned to it (which can be modified, however, for example, to switch to another instrument in the middle of a score).

The MIDI standard supports only 16 channels per MIDI device. This limit on the number of channels also limits the number of different instruments which can be played at the same time.

LilyPond creates separate MIDI tracks for each staff, (or discrete instrument or voice, depending on the value of `Score.midiChannelMapping`), and also for each lyrics context. There is no limit to the number of tracks.

To work around the limited number of MIDI channels, LilyPond supports a number of different modes for MIDI channel allocation, selected using the `Score.midiChannelMapping` context property. In each case, if more MIDI channels than the limit are required, the allocated channel numbers wrap around back to 0, possibly causing the incorrect assignment of instruments to some notes. This context property can be set to one of the following values:

`'staff`

Allocate a separate MIDI channel to each staff in the score (this is the default). All notes in all voices contained within each staff will share the MIDI channel of their enclosing staff, and all are encoded in the same MIDI track.

The limit of 16 channels is applied to the total number of staff and lyrics contexts, even though MIDI lyrics do not take up a MIDI channel.

`'instrument`

Allocate a separate MIDI channel to each distinct MIDI instrument specified in the score. This means that all the notes played with the same MIDI instrument will share the same MIDI channel (and track), even if the notes come from different voices or staves.

In this case the lyrics contexts do not count towards the MIDI channel limit of 16 (as they will not be assigned to a MIDI instrument), so this setting may allow a better allocation of MIDI channels when the number of staves and lyrics contexts in a score exceeds 16.

`'voice`

Allocate a separate MIDI channel to each voice in the score that has a unique name among the voices in its enclosing staff. Voices in different staves are always assigned separate MIDI channels, but any two voices contained within the same staff will share the same MIDI channel if they have the same name. Because `midiInstrument` and the several MIDI controls for effects are properties of the staff context, they cannot be set separately for each voice. The first voice will be played with the instrument and effects specified for the staff, and voices with a different name from the first will be assigned the default instrument and effects.

Note: different instruments and/or effects can be assigned to several voices on the same staff by moving the `Staff_performer` from the `Staff` to the `Voice` context, and leaving `midiChannelMapping` to default to `'staff` or set to `'instrument`; see the snippet below.

For example, the default MIDI channel mapping of a score can be changed to the `'instrument` setting as shown:

```
\score {
  ...music...
  \midi {
    \context {
      \Score
      midiChannelMapping = #'instrument
    }
  }
}
```
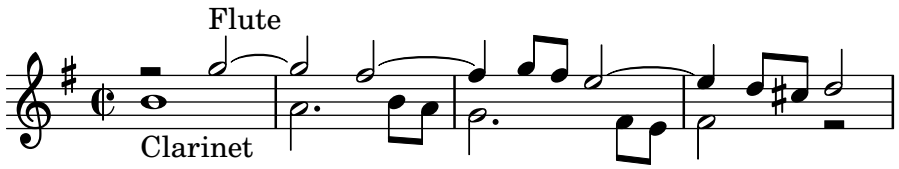
Selected snippets

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"
    }
    \tempo 2 = 72
  }
}
```



24.8 Context properties for MIDI effects

The following context properties can be used to apply various MIDI effects to notes played on the MIDI channel associated with the current staff, MIDI instrument or voice (depending on the value of the `Score.midiChannelMapping` context property and the context in which the `Staff_performer` is located; see Section 24.7 [MIDI channel mapping], page 661).

Changing these context properties will affect all notes played on the channel after the change, however some of the effects may even apply also to notes which are already playing (depending on the implementation of the MIDI output device).

The following context properties are supported:

`Staff.midiPanPosition`
: The pan position controls how the sound on a MIDI channel is distributed between left and right stereo outputs. The context property accepts a number between -1.0 (`#LEFT`) and 1.0 (`#RIGHT`); the value -1.0 will put all sound power to the left stereo output (keeping the right output silent), the value 0.0 (`#CENTER`) will distribute the sound evenly between the left and right stereo outputs, and the value 1.0 will move all sound to the right stereo output. Values between -1.0 and 1.0 can be used to obtain mixed distributions between left and right stereo outputs.

`Staff.midiBalance`
: The stereo balance of a MIDI channel. Similarly to the pan position, this context property accepts a number between -1.0 (`#LEFT`) and 1.0 (`#RIGHT`). It varies the relative volume sent to the two stereo speakers without affecting the distribution of the stereo signals.

`Staff.midiExpression`
: Expression level (as a fraction of the maximum available level) to apply to a MIDI channel. A MIDI device combines the MIDI channel's expression level with a voice's current dynamic level (controlled using constructs such as `\p` or `\ff`) to obtain the total volume of each note within the voice. The expression control could be used, for example, to implement crescendo or decrescendo effects over single sustained notes (not supported automatically by LilyPond).

  The expression level ranges from 0.0 (no expression, meaning zero volume) to 1.0 (full expression).

`Staff.midiReverbLevel`
: Reverb level (as a fraction of the maximum available level) to apply to a MIDI channel. This property accepts numbers between 0.0 (no reverb) and 1.0 (full effect).

`Staff.midiChorusLevel`
: Chorus level (as a fraction of the maximum available level) to apply to a MIDI channel. This property accepts numbers between 0.0 (no chorus effect) and 1.0 (full effect).

Known issues and warnings

As MIDI files do not contain any actual audio data, changes in these context properties translate only to requests for changing MIDI channel controls in the outputted MIDI files. Whether a particular MIDI device (such as a software MIDI player) can actually handle any of these requests in a MIDI file is entirely up to the implementation of the device: a device may choose to ignore some or all of these requests. Also, how a MIDI device will interpret different values for these controls (generally, the MIDI standard fixes the behavior only at the endpoints of the value range available for each control), and whether a change in the value of a control will affect notes already playing on that MIDI channel or not, is also specific to the MIDI device implementation.

When generating MIDI files, LilyPond will simply transform the fractional values within each range linearly into values in a corresponding (7-bit, or 14-bit for MIDI channel controls which support fine resolution) integer range (0-127 or 0-16383, respectively), rounding fractional values towards the nearest integer away from zero. The converted integer values are stored as-is in the generated MIDI file. Please consult the documentation of your MIDI device for information about how the device interprets these values.

24.9 Enhancing MIDI output

The default MIDI output is basic but can be improved by setting MIDI instruments and various `\midi` block properties.

Additional scripts allow to fine-tune the way dynamics, articulations and rhythm are rendered in MIDI: the `articulate` script and the `swing` script.

24.9.1 The `articulate` script

To use the `articulate` script add the appropriate `\include` command at the top of the input file;

```
\include "articulate.ly"
```

The script creates MIDI output into appropriately 'time-scaled' notes to match many articulation and tempo indications. Engraved output however, will also be altered to literally match the MIDI output.

```
\score {
  \articulate <<
    ... music ...
  >>
  \midi { }
}
```

The `\articulate` command enables abbreviatures (such as trills and turns) to be processed. A full list of supported items can be found in the script itself. See `ly/articulate.ly`.

See also

Learning Manual: Section "Other sources of information" in *Learning Manual*.

Notation Reference: Chapter 27 [Score layout], page 683.

Installed Files: `ly/articulate.ly`.

> **Note:** The `articulate` script may shorten chords, which might not be appropriate for some types of instrument, such as organ music. Notes that do not have any articulations attached to them may also be shortened; so to allow for this, restrict the use of the `\articulate` function to shorter segments of music, or modify the values of the variables defined in the `articulate` script to compensate for the note-shortening behavior.

24.9.2 The `swing` script

The `swing` script provides additional functions allowing for regular durations to be played with an unequal rhythm. The most obvious example is 'swing' interpretation commonly found in jazz music where binary eighth notes should be played in a ternary fashion; however additional interpretations are also supported.

This script has to be `\include`-d at the beginning of the input file:

```
\include "swing.ly"
```

Three commands are provided:

- `\tripletFeel` creates a triplet-feel swing. It takes two arguments: the durations that should be affected by it (typically `8` for eighth notes), and then the music expression to which it should be applied.
- `\applySwing` takes an additional argument prior to the music expression: a 'weight list' of *n* number ratios expressing the way regular notes should be played: for example, `#'(2 1)` indicates that every other note should be played twice as long as the following note (in fact, `\tripletFeel` *duration* {*music*} is actually a shortcut for `\applySwing` *duration* `#'(2 1)` {*music*}). Smoother swung eighths may be obtained with a weight list of `#'(3 2)`, or other values depending on taste.

  That list may include more than two values, which allows for longer and more sophisticated groove patterns; for example, a samba feel for sixteenth notes may be obtained as follows:

```
\score {
  \applySwing 16 #'(3 2 2 3) {
    ... music ...
  }
  \midi { }
}
```

- `\applySwingWithOffset` adds yet another argument between the 'weight list' and the music expression: an offset length. This command should be used when the music expression has to start off-beat, with a partial swing cycle.

> **Note:** As with the `articulate` script, all swing commands are also rendered in the engraved output, which results in irregular note spacing. This can be avoided by using them only in a `\score` block dedicated to MIDI output, rather than to printed music.

Additional help and information is included in the script file: see `ly/swing.ly`.

See also

Learning Manual: Section "Other sources of information" in *Learning Manual*.

Notation Reference: Chapter 2 [Rhythms], page 52.

Installed Files: `ly/swing.ly`.

Known issues and warnings

- `\repeat` constructs in music (even `\repeat unfold`) are not taken into consideration when determining note timing. This will lead to problems unless the durations of all repeated parts are integer multiples of the swing cycle duration.
- These functions are oblivious to time signatures and measures. That is why offsets need to be supplied by using `\applySwingWithOffset` if music starts off-beat.
- Grace notes are ignored and simply left unaffected; so are tuplets.

25 Extracting musical information

In addition to creating graphical output and MIDI, LilyPond can display musical information as text.

25.1 Displaying LilyPond notation

Displaying a music expression in LilyPond notation can be done with the music function `\displayLilyMusic`. To see the output, you will typically want to call LilyPond using the command line. For example,

```
{
  \displayLilyMusic \transpose c a, { c4 e g a bes }
}
```

will display

```
{ a,4 cis4 e4 fis4 g4 }
```

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 `\displayLilyMusic`, redirect the output to a file.

```
lilypond file.ly >display.txt
```

Note that LilyPond does not just display the music expression, but also interprets it (since `\displayLilyMusic` returns it in addition to displaying it). Just insert `\displayLilyMusic` into the existing music in order to get information about it.

To interpret and display a music section in the console but, at the same time, remove it from the output file use the `\void` command.

```
{
  \void \displayLilyMusic \transpose c a, { c4 e g a bes }
  c1
}
```

25.2 Displaying Scheme music expressions

See Section “Displaying music expressions” in *Extending*.

25.3 Saving music events to a file

Music events can be saved to a file on a per-staff basis by including a file in your main score.

```
\include "event-listener.ly"
```

This creates file(s) called `FILENAME-STAFFNAME.notes` or `FILENAME-unnamed-staff.notes` for each staff. Note that if you have multiple unnamed staves, the events for all staves are mixed together in the same file. The output looks like this:

```
0.000   note     57       4   p-c 2 12
0.000   dynamic  f
0.250   note     62       4   p-c 7 12
0.500   note     66       8   p-c 9 12
0.625   note     69       8   p-c 14 12
0.750   rest     4
0.750   breathe
```

The syntax is a tab-delimited line, with two fixed fields on each line followed by optional parameters.

```
time  type  ...params...
```

This information can easily be read into other programs such as python scripts, and can be very useful for researchers wishing to perform musical analysis or playback experiments with LilyPond.

Known issues and warnings

Not all lilypond music events are supported by `event-listener.ly`. It is intended to be a well-crafted "proof of concept". If some events that you want to see are not included, copy `event-listener.ly` into your lilypond directory and modify the file so that it outputs the information you want.

Spacing issues

26 Page layout

The global paper layout is determined by three factors: the page layout, the line breaks, and the spacing. These all influence each other. The choice of spacing determines how densely each system of music is set. This influences where line breaks are chosen, and thus ultimately, how many pages a piece of music takes.

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.

Two types of blocks can contain layout settings: `\paper {...}` and `\layout {...}`. The `\paper` block contains page layout settings that are expected to be the same for all scores in a book or book part, such as the paper height, or whether to print page numbers, etc. See Chapter 26 [Page layout], page 671. The `\layout` block contains score layout settings, such as the number of systems to use, or the space between staff groups, etc. See Chapter 27 [Score layout], page 683.

26.1 The \paper block

`\paper` blocks may be placed in three different places to form a descending hierarchy of `\paper` blocks:

- At the top of the input file, before all `\book`, `\bookpart`, and `\score` blocks.
- Within a `\book` block but outside all the `\bookpart` and `\score` blocks within that book.
- Within a `\bookpart` block but outside all `\score` blocks within that book part.

A `\paper` block cannot be placed within a `\score` block.

The values of the fields filter down this hierarchy, with the values set higher in the hierarchy persisting unless they are overridden by a value set lower in the hierarchy.

Several `\paper` blocks can appear at each of the levels, for example as parts of several `\included` files. If so, the fields at each level are merged, with values encountered last taking precedence if duplicated fields appear.

Settings that can appear in a `\paper` block include:

- the `set-paper-size` Scheme function,
- `\paper` variables used for customizing page layout, and
- markup definitions used for customizing the layout of headers, footers, and titles.

The `set-paper-size` function is discussed in the next section, Section 26.2 [Paper size and automatic scaling], page 672. The `\paper` variables that deal with page layout are discussed in later sections. The markup definitions that deal with headers, footers, and titles are discussed in Section 21.2 [Custom titles, headers, and footers], page 608.

Most `\paper` variables will only work in a `\paper` block. The few that will also work in a `\layout` block are listed in Section 27.1 [The \layout block], page 683.

Except when specified otherwise, all `\paper` variables that correspond to distances on the page are measured in millimeters, unless a different unit is specified by the user. For example, the following declaration sets `top-margin` to ten millimeters:

```
\paper {
  top-margin = 10
}
```

To set it to 0.5 inches, use the \in unit suffix:

```
\paper {
  top-margin = 0.5\in
}
```

The available unit suffixes are \mm, \cm, \in, \pt, and \bp. These units are simple values for converting from millimeters; they are defined in ly/paper-defaults-init.ly. For the sake of clarity, when using millimeters, the \mm is typically included in the code, even though it is not technically necessary.

It is also possible to define \paper values using Scheme. The Scheme equivalent of the above example is:

```
\paper {
  #(define top-margin (* 0.5 in))
}
```

Finally, you can also predefine paper variables.

```
bigMargin = \paper { top-margin = 10\cm }

\paper {
  \bigMargin
  indent = 0\mm
}
```

See also

Notation Reference: Section 26.2 [Paper size and automatic scaling], page 672, Section 21.2 [Custom titles, headers, and footers], page 608, Section 27.1 [The \layout block], page 683.

Installed Files: ly/paper-defaults-init.ly.

26.2 Paper size and automatic scaling

26.2.1 Setting the paper size

'A4' is the default value when no explicit paper size is set. However, there are two functions that can be used to change it:

set-default-paper-size

```
#(set-default-paper-size "quarto")
```

which must always be placed at the top-level scope, and

set-paper-size

```
\paper {
  #(set-paper-size "tabloid")
}
```

which must always be placed in a \paper block.

If the set-default-paper-size function is used in the top-level scope, it must come before any \paper block. set-default-paper-size sets the paper size for all pages, whereas set-paper-size only sets the paper size for the pages that the \paper block applies to. For example, if the \paper block is at the top of the file, then it applies the paper size to all pages. If the \paper block is inside a \book, then the paper size applies only to that book.

When the set-paper-size function is used, it must be placed *before* any other functions used within the same \paper block. See Section 26.2.2 [Automatic scaling to paper size], page 673.

Paper sizes are defined in file scm/paper.scm; see Section B.5 [Predefined paper sizes], page 899, for a complete list.

Both set-default-paper-size and set-paper-size also accept a quoted number pair as its argument to set a custom paper size. For example,

```
#(set-default-paper-size '(cons (* 100 mm) (* 50 mm)))
```

sets the paper width and height to 100 mm and 50 mm, respectively.

Possible units are in (inches), cm (centimeters), mm (millimeters), pt (points), and bp (big points).

If the symbol 'landscape is added to the paper size function as a second argument, pages are rotated by 90 degrees, and wider line widths are set accordingly.

```
#(set-default-paper-size "a6" 'landscape)
```

Swapping the paper dimensions *without* having the print rotated (like when printing to postcard size, or creating graphics for inclusion rather than a stand-alone document) can be achieved by appending 'landscape' to the name of the paper size itself:

```
#(set-default-paper-size "a6landscape")
```

When the paper size ends with an explicit 'landscape' or 'portrait', the presence of a 'landscape symbol *only* affects print orientation, not the paper dimensions used for layout.

See also

Notation Reference: Section 26.2.2 [Automatic scaling to paper size], page 673, Section B.5 [Predefined paper sizes], page 899.

Installed Files: scm/paper.scm.

26.2.2 Automatic scaling to paper size

If the paper size is changed with one of the Scheme functions (set-default-paper-size or set-paper-size), the values of several \paper variables are automatically scaled to the new size. To bypass the automatic scaling for a particular variable, set the variable after setting the paper size. Note that the automatic scaling is not triggered by setting the paper-height or paper-width variables, even though paper-width can influence other values (this is separate from scaling and is discussed below). The set-default-paper-size and set-paper-size functions are described in Section 26.2.1 [Setting the paper size], page 672.

The vertical dimensions affected by automatic scaling are top-margin and bottom-margin (see Section 26.3 [Fixed vertical spacing \paper variables], page 673). The horizontal dimensions affected by automatic scaling are left-margin, right-margin, inner-margin, outer-margin, binding-offset, indent, and short-indent (see Section 26.5 [Horizontal spacing \paper variables], page 676).

The default values for these dimensions are set in ly/paper-defaults-init.ly, using internal variables named top-margin-default, bottom-margin-default, etc. These are the values that result at the default paper size a4. For reference, with a4 paper the paper-height is 297\mm and the paper-width is 210\mm.

See also

Notation Reference: Section 26.3 [Fixed vertical spacing \paper variables], page 673, Section 26.5 [Horizontal spacing \paper variables], page 676.

Installed Files: ly/paper-defaults-init.ly, scm/paper.scm.

26.3 Fixed vertical spacing \paper variables

> **Note:** Some \paper dimensions are automatically scaled to the paper size, which may lead to unexpected behavior. See Section 26.2.2 [Automatic scaling to paper size], page 673.

Default values (before scaling) are defined in file `ly/paper-defaults-init.ly`.

`paper-height`
: The height of the page, unset by default. Note that the automatic scaling of some vertical dimensions is not affected by this.

`top-margin`
: The margin between the top of the page and the top of the printable area. If the paper size is modified, this dimension's default value is scaled accordingly.

`bottom-margin`
: The margin between the bottom of the printable area and the bottom of the page. If the paper size is modified, this dimension's default value is scaled accordingly.

`ragged-bottom`
: If this is set to `#t`, systems will be set at their natural spacing, neither compressed nor stretched vertically to fit the page.

`ragged-last-bottom`
: If this is set to `#f`, then the last page, and the last page in each section created with a \bookpart block, will be vertically justified in the same way as the earlier pages.

See also

Notation Reference: Section 26.2.2 [Automatic scaling to paper size], page 673.

Installed Files: `ly/paper-defaults-init.ly`.

Snippets: Section "Spacing" in *Snippets*.

Known issues and warnings

The titles (from the \header block) 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.

Explicitly defined paper sizes will override any user-defined top or bottom margin settings.

26.4 Flexible vertical spacing \paper variables

In most cases, it is preferable for the vertical distances between certain items (such as margins, titles, systems, and separate scores) to be flexible, so that they stretch and compress nicely according to each situation. A number of \paper variables (listed below) are available to fine-tune the stretching behavior of these dimensions.

Note that the \paper variables discussed in this section do not control the spacing of staves within individual systems. Within-system spacing is controlled by grob properties, with settings typically entered inside a \score or \layout block, and not inside a \paper block. See Section 29.1 [Flexible vertical spacing within systems], page 697.

26.4.1 Structure of flexible vertical spacing alists

Each of the flexible vertical spacing \paper variables is an alist (association list) containing four *keys*:

- `basic-distance` – the vertical distance, measured in staff spaces, between the *reference points* of the two items, when no collisions would result, and no stretching or compressing

is in effect. The reference point of a (title or top-level) markup is its highest point, and the reference point of a system is the vertical center of the nearest `StaffSymbol` – even if a non-staff line (such as a `Lyrics` context) is in the way. Values for `basic-distance` that are less than either `padding` or `minimum-distance` are not meaningful, since the resulting distance will never be less than either `padding` or `minimum-distance`.

- `minimum-distance` – the smallest allowable vertical distance, measured in staff spaces, between the reference points of the two items, when compressing is in effect. Values for `minimum-distance` that are less than `padding` are not meaningful, since the resulting distance will never be less than `padding`.
- `padding` – the minimum required amount of unobstructed vertical whitespace between the bounding boxes (or skylines) of the two items, measured in staff spaces.
- `stretchability` – a unitless measure of the dimension's relative propensity to stretch. If zero, the distance will not stretch (unless collisions would result). When positive, the significance of a particular dimension's `stretchability` value lies only in its relation to the `stretchability` values of the other dimensions. For example, if one dimension has twice the `stretchability` of another, it will stretch twice as easily. Values should be non-negative and finite. The value `+inf.0` triggers a `programming_error` and is ignored, but `1.0e7` can be used for an almost infinitely stretchable spring. If unset, the default value is set to `basic-distance`. Note that the dimension's propensity to *compress* cannot be directly set by the user and is equal to (`basic-distance` − `minimum-distance`).

If a page has a ragged bottom, the resulting distance is the largest of:

- `basic-distance`,
- `minimum-distance`, and
- `padding` plus the smallest distance necessary to eliminate collisions.

For multi-page scores with a ragged bottom on the last page, the last page uses the same spacing as the preceding page, provided there is enough space for that.

Specific methods for modifying alists are discussed in Section 35.9 [Modifying alists], page 771. The following example demonstrates the two ways these alists can be modified. The first declaration updates one key value individually, and the second completely redefines the variable:

```
\paper {
  system-system-spacing.basic-distance = 8
  score-system-spacing =
    #'((basic-distance . 12)
       (minimum-distance . 6)
       (padding . 1)
       (stretchability . 12))
}
```

26.4.2 List of flexible vertical spacing \paper variables

The names of these variables follow the format *upper*`-`*lower*`-spacing`, where *upper* and *lower* are the items to be spaced. Each distance is measured between the reference points of the two items (see the description of the alist structure above). Note that in these variable names, the term 'markup' refers to both *title markups* (`bookTitleMarkup` or `scoreTitleMarkup`) and *top-level markups* (see Section 20.5 [File structure], page 596). All distances are measured in staff spaces.

Default settings are defined in `ly/paper-defaults-init.ly`.

`markup-system-spacing`
: the distance between a (title or top-level) markup and the system that follows it.

`score-markup-spacing`
: the distance between the last system of a score and the (title or top-level) markup that follows it.

`score-system-spacing`
: the distance between the last system of a score and the first system of the score that follows it, when no (title or top-level) markup exists between them.

`system-system-spacing`
: the distance between two systems in the same score.

`markup-markup-spacing`
: the distance between two (title or top-level) markups.

`last-bottom-spacing`
: the distance from the last system or top-level markup on a page to the bottom of the printable area (i.e., the top of the bottom margin).

`top-system-spacing`
: the distance from the top of the printable area (i.e., the bottom of the top margin) to the first system on a page, when there is no (title or top-level) markup between the two.

`top-markup-spacing`
: the distance from the top of the printable area (i.e., the bottom of the top margin) to the first (title or top-level) markup on a page, when there is no system between the two.

See also

Notation Reference: Section 29.1 [Flexible vertical spacing within systems], page 697.

Installed Files: `ly/paper-defaults-init.ly`.

Snippets: Section "Spacing" in *Snippets*.

26.5 Horizontal spacing \paper variables

> **Note:** Some `\paper` dimensions are automatically scaled to the paper size, which may lead to unexpected behavior. See Section 26.2.2 [Automatic scaling to paper size], page 673.

26.5.1 \paper variables for widths and margins

Default values (before scaling) that are not listed here are defined in file `ly/paper-defaults-init.ly`.

`paper-width`
: The width of the page, unset by default. While `paper-width` has no effect on the automatic scaling of some horizontal dimensions, it does influence the `line-width` variable. If both `paper-width` and `line-width` are set, then `left-margin` and `right-margin` will also be updated. Also see `check-consistency`.

`line-width`
: When specified in a `\paper` block this defines the horizontal extent available for the staff lines in unindented systems. If left unspecified, the paper's `line-width` is determined from `(paper-width − left-margin − right-margin)`. If the paper's `line-width` is specified, and both `left-margin` and `right-margin` are not, then

the margins will be updated to center the systems on the page automatically. Also see `check-consistency`.

`line-widths` for individual scores can be specified in the scores' `\layout` blocks. These values control the width of the lines produced on a score-by-score basis. If `line-width` is not specified for a score, it defaults to the paper's `line-width`. Setting a score's `line-width` has no effect on the paper margins. Staff lines, of a length determined by the score's `line-width`, are left-aligned within the paper area defined by the paper's `line-width`. If the score and paper `line-widths` are equal, the staff lines will extend exactly from the left margin to the right margin, but if the score's `line-width` is greater than the paper's `line-width` the staff lines will run over into the right margin.

`left-margin`
: The margin between the left edge of the page and the start of the staff lines in unindented systems. If the paper size is modified, this dimension's default value is scaled accordingly. If `left-margin` is unset, and both `line-width` and `right-margin` are set, then `left-margin` is set to `(paper-width − line-width − right-margin)`. If only `line-width` is set, then both margins are set to `((paper-width − line-width) / 2)`, and the systems are consequently centered on the page. Also see `check-consistency`.

`right-margin`
: The margin between the right edge of the page and the end of the staff lines in non-ragged systems. If the paper size is modified, this dimension's default value is scaled accordingly. If `right-margin` is unset, and both `line-width` and `left-margin` are set, then `right-margin` is set to `(paper-width − line-width − left-margin)`. If only `line-width` is set, then both margins are set to `((paper-width − line-width) / 2)`, and the systems are consequently centered on the page. Also see `check-consistency`.

`check-consistency`
: If this is true (the default value), print a warning if `left-margin`, `line-width`, and `right-margin` do not exactly add up to `paper-width`, and replace each of these (except `paper-width`) with their default values (scaled to the paper size if necessary). If set to `#f`, ignore any inconsistencies and allow systems to run off the edge of the page.

`ragged-right`
: If set to `#t`, systems will not fill the line width. Instead, systems end at their natural horizontal length. Default: `#t` for scores with only one system, and `#f` for scores with two or more systems. This variable can also be set in a `\layout` block.

`ragged-last`
: If set to `#t`, the last system in the score will not fill the line width. Instead the last system ends at its natural horizontal length. Default: `#f`. This variable can also be set in a `\layout` block.

See also

Notation Reference: Section 26.2.2 [Automatic scaling to paper size], page 673.

Installed Files: `ly/paper-defaults-init.ly`.

Known issues and warnings

Explicitly defined paper sizes will override any user-defined left or right margin settings.

26.5.2 \paper variables for two-sided mode

Default values (before scaling) are defined in `ly/paper-defaults-init.ly`.

`two-sided`
: If set to `#t`, 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`.

`inner-margin`
: The margin all pages have at the inner side if they are part of a book. If the paper size is modified, this dimension's default value is scaled accordingly. Works only with `two-sided` set to `#t`.

`outer-margin`
: The margin all pages have at the outer side if they are part of a book. If the paper size is modified, this dimension's default value is scaled accordingly. Works only with `two-sided` set to `#t`.

`binding-offset`
: The amount `inner-margin` is increased to make sure nothing will be hidden by the binding. If the paper size is modified, this dimension's default value is scaled accordingly. Works only with `two-sided` set to `#t`.

See also

Notation Reference: Section 26.2.2 [Automatic scaling to paper size], page 673.

Installed Files: `ly/paper-defaults-init.ly`.

26.5.3 \paper variables for shifts and indents

Default values (before scaling) that are not listed here are defined in `ly/paper-defaults-init.ly`.

`horizontal-shift`
: The amount that all systems (including titles and system separators) are shifted to the right. Default: `0.0\mm`.

`indent`
: The level of indentation for the first system in a score. If the paper size is modified, this dimension's default value is scaled accordingly. The space within `line-width` available for the first system is reduced by this amount. `indent` may also be specified in `\layout` blocks to set indents on a score-by-score basis.

`short-indent`
: The level of indentation for all systems in a score besides the first system. If the paper size is modified, this dimension's default value is scaled accordingly. The space within `line-width` available for systems other than the first is reduced by this amount. `short-indent` may also be specified in `\layout` blocks to set short indents on a score-by-score basis.

See also

Notation Reference: Section 26.2.2 [Automatic scaling to paper size], page 673.

Installed Files: `ly/paper-defaults-init.ly`.

Snippets: Section "Spacing" in *Snippets*.

26.6 Other \paper variables

26.6.1 \paper variables for line breaking

`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.

`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.

`system-count`
: The number of systems to be used for a score. Default: unset. This variable can also be set in a `\layout` block.

See also

Notation Reference: Section 28.1 [Line breaking], page 689.

26.6.2 \paper variables for page breaking

Default values not listed here are defined in `ly/paper-defaults-init.ly`

`page-breaking`
: The page breaking algorithm to use. Choices are `ly:minimal-breaking`, `ly:page-turn-breaking`, `ly:one-page-breaking`, `ly:one-line-breaking`, `ly:one-line-auto-height-breaking`, and `ly:optimal-breaking`. Default: `ly:optimal-breaking`.

`page-breaking-system-system-spacing`
: Tricks the page breaker into thinking that `system-system-spacing` is set to something different than it really is. For example, if `page-breaking-system-system-spacing.padding` is set to something substantially larger than `system-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-spacing-weight`
: When using the `ly:optimal-breaking` algorithm for page breaking, LilyPond has to make trade-offs between horizontal and vertical stretching so that the overall spacing is more acceptable. This parameter controls the relative importance of page (vertical) spacing and line (horizontal) spacing. High values will make page spacing more important. Default: `10`.

The following variables are effective only when `page-breaking` is set to `ly:page-turn-breaking`. Page breaks are then chosen to minimize the number of page turns. Since page turns are required on moving from an odd-numbered page to an even-numbered one, a layout in which the last page is odd-numbered will usually be favored. Places where page turns are preferred can be indicated manually by inserting `\allowPageTurn` or automatically by including the `Page_turn_engraver` (see Section 28.2.7 [Optimal page turning], page 695).

If there are insufficient choices available for making suitable page turns, LilyPond may insert a blank page either within a score, between scores (if there are two or more scores), or by ending a score on an even-numbered page. The values of the following three variables may be increased to make these actions less likely.

The values are penalties, i.e., the higher the value the less likely will be the associated action relative to other choices.

`blank-page-penalty`
: The penalty for having a blank page in the middle of a score. If `blank-page-penalty` is large and `ly:page-turn-breaking` is selected, then LilyPond will be less likely to insert a page in the middle of a score. Instead, it will space out the music further to fill the blank page and the following one. Default: 5.

`blank-last-page-penalty`
: The penalty for ending the score on an even-numbered page. If `blank-last-page-penalty` is large and `ly:page-turn-breaking` is selected, then LilyPond will be less likely to produce a score in which the last page is even-numbered. Instead, it will adjust the spacing in order to use one page more or one page less. Default: 0.

`blank-after-score-page-penalty`
: The penalty for having a blank page after the end of one score and before the next. By default, this is smaller than `blank-page-penalty`, so that blank pages after scores are inserted in preference to blank pages within a score. Default: 2.

See also

Notation Reference: Section 28.2 [Page breaking], page 693, Section 28.2.2 [Optimal page breaking], page 694, Section 28.2.7 [Optimal page turning], page 695, Section 28.2.3 [Minimal page breaking], page 694, Section 28.2.4 [One-page page breaking], page 694, Section 28.2.5 [One-line page breaking], page 695, Section 28.2.6 [One-line-auto-height page breaking], page 695.

Installed Files: `ly/paper-defaults-init.ly`.

26.6.3 `\paper` variables for page numbering

Default values not listed here are defined in `ly/paper-defaults-init.ly`

`auto-first-page-number`
: The page breaking algorithm is affected by the first page number being odd or even. If set to `#t`, the page breaking algorithm will decide whether to start with an odd or even number. This will result in the first page number remaining as is or being increased by one. Default: `#f`.

`first-page-number`
: The value of the page number on the first page.

`print-first-page-number`
: If set to `#t`, a page number is printed on the first page.

`print-page-number`
: If set to `#f`, page numbers are not printed.

`page-number-type`
: The type of numerals used for page numbers. Choices include `'arabic`, `'roman-ij-lower`, `'roman-ij-upper`, `'roman-lower`, and `'roman-upper`. Default: `'arabic`.

bookpart-level-page-numbering
: If set to #t, each book part has its independent sequence of page numbers, starting at first-page-number (1 by default).

  This may also be used for one book part only. The typical use case is numbering pages of the first book part independently and in roman numerals, as may be wished for an analytical introduction to the work being published.

```
\book {
  \bookpart {
    \paper {
      bookpart-level-page-numbering = ##t
      page-number-type = #'roman-lower
    }
    \markuplist \wordwrap-lines {
      Lorem ipsum dolor sit amet.
    }
  }
  \bookpart {
    ...
  }
}
```

See also

Installed Files: ly/paper-defaults-init.ly.

Known issues and warnings

Odd page numbers are always on the right. If you want the music to start on page 1 there must be a blank page on the back of the cover page so that page 1 is on the right-hand side.

26.6.4 \paper variables concerning headers and markups

print-all-headers
: If set to #t, this will print all headers for each \score in the output. Normally only the piece and opus header variables are printed. For use cases see Chapter 21 [Titles and headers], page 600. Default: #f.

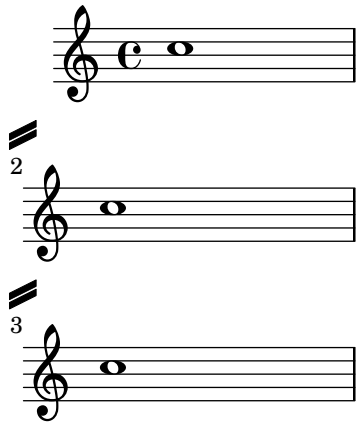
reset-footnotes-on-new-page
: If set to #t, footnote numbers are reset on each page break. For footnotes numbered consecutively across page breaks, set to #f. Default: #t.

system-separator-markup
: A markup object that is inserted between systems, often used for orchestral scores. Default: unset. The \slashSeparator markup, defined in ly/titling-init.ly, is provided as a sensible default, for example:

```
#(set-default-paper-size "a8")

\book {
  \paper {
    system-separator-markup = \slashSeparator
  }
  \header {
    tagline = ##f
  }
  \score {
```

```
    \relative { c''1 \break c1 \break c1 }
  }
}
```



`footnote-separator-markup`
: A markup object that is inserted above the footnote texts at the bottom of the page. Default: a centered horizontal line, defined in `ly/paper-defaults-init.ly`.

See also

Installed Files: `ly/titling-init.ly`, `ly/paper-defaults-init.ly`.

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.

26.6.5 \paper variables for debugging

The variables `debug-beam-scoring`, `debug-slur-scoring` and `debug-tie-scoring` allow to print debugging output for beam, slur and tie scoring. See Section "Debugging scoring algorithms" in *Contributor's Guide* for a detailed explanation, what these variables do.

27 Score layout

This section discusses score layout options for the \layout block.

27.1 The \layout block

While the \paper block contains settings that relate to the page formatting of the whole document, the \layout block contains settings for score-specific layout. To set score layout options globally, enter them in a top-level \layout block. To set layout options for an individual score, enter them in a \layout block inside the \score block, after the music. Settings that can appear in a \layout block include:

- the layout-set-staff-size Scheme function,
- context modifications in \context blocks, and
- \paper variables that affect score layout.

The layout-set-staff-size function is discussed in the next section, Section 27.2 [Setting the staff size], page 685. Context modifications are discussed in a separate chapter; see Section 33.4 [Modifying context plug-ins], page 745, and Section 33.5 [Changing context default settings], page 746.

The \paper variables that can appear in a \layout block are as follows.

- line-width, ragged-right and ragged-last (see Section 26.5.1 [\paper variables for widths and margins], page 676)
- indent and short-indent (see Section 26.5.3 [\paper variables for shifts and indents], page 678)
- system-count (see Section 26.6.1 [\paper variables for line breaking], page 679)

The default values of the above variables are taken from the \paper block.

Here is an example \layout block:

```
\layout {
  indent = 2\cm
  \context {
    \StaffGroup
    \override StaffGrouper.staff-staff-spacing.basic-distance = 8
  }
  \context {
    \Voice
    \override TextScript.padding = 1
    \override Glissando.thickness = 3
  }
}
```

Multiple \layout blocks can be entered as top-level expressions. This can, for example, be useful if different settings are stored in separate files and included optionally. Internally, when a \layout block is evaluated, a copy of the current \layout configuration is made, then any changes defined within the block are applied and the result is saved as the new current configuration. From the user's perspective the \layout blocks are combined, but in conflicting situations (when the same property is changed in different blocks) the later definitions take precedence.

For example, if this block

```
\layout {
  \context {
    \Voice
```

```
    \override TextScript.color = #magenta
    \override Glissando.thickness = 1.5
  }
}
```

is placed after the one from the preceding example the `padding` and `color` overrides for `TextScript` are combined, but the later `thickness` override for `Glissando` replaces (or hides) the earlier one.

`\layout` blocks may be assigned to variables for reuse later, but the way this works is slightly but significantly different from writing them literally.

If a variable is defined like this

```
layoutVariable = \layout {
  \context {
    \Voice
    \override NoteHead.font-size = 4
  }
}
```

it holds the current `\layout` configuration with the `NoteHead.font-size` override added, but this combination is *not* saved as the new current configuration. Be aware that the 'current configuration' is read when the variable is defined and not when it is used, so the content of the variable is dependent on its position in the source.

The variable can then be used inside another `\layout` block, for example:

```
\layout {
  \layoutVariable
  \context {
    \Voice
    \override NoteHead.color = #red
  }
}
```

A `\layout` block containing a variable, as in the example above, does *not* copy the current configuration but instead uses the content of `\layoutVariable` as the base configuration for further additions. This means that any changes defined between the definition and the use of the variable are lost.

If `layoutVariable` is defined (or `\included`) immediately before being used, its content is just the current configuration plus the overrides defined within it. So in the example above showing the use of `\layoutVariable` the final `\layout` block would consist of

```
TextScript.padding = 1
TextScript.color = #magenta
Glissando.thickness = 1.5
NoteHead.font-size = 4
NoteHead.color = #red
```

plus the `indent` and the `StaffGrouper` overrides.

However, if the variable had already been defined before the first `\layout` block the current configuration would now contain only the following.

```
NoteHead.font-size = 4  % written in the variable definition
NoteHead.color = #red   % added after the use of the variable
```

If carefully planned, `\layout` variables can be a valuable tool to structure the layout design of sources, and also to reset the `\layout` configuration to a known state.

See also

Notation Reference: Section 33.5 [Changing context default settings], page 746.

Snippets: Section "Spacing" in *Snippets*.

27.2 Setting the staff size

The default *staff size* is 20 points, which corresponds to a staff height of 7.03 mm (one point is equal to 100/7227 of an inch, or 2540/7227 mm). The staff size may be changed in three ways:

1. To set the staff size globally for all scores in a file, use `set-global-staff-size`.

   ```
   #(set-global-staff-size 14)
   ```

   The above example sets the global default staff size to 14 pt (4.92 mm) and scales all fonts accordingly.

   The function can also be used to set various staff sizes for different `\book` blocks:

   ```
   #(set-global-staff-size 30)
   \book {
     { c' }
   }

   #(set-global-staff-size 10)
   \book {
     { c' }
   }
   ```

2. To set the staff size for a single score within a book, use `layout-set-staff-size` inside that score's `\layout` block:

   ```
   \score {
     ...
     \layout {
       #(layout-set-staff-size 14)
     }
   }
   ```

3. To set the staff size for a single staff within a system, use the `\magnifyStaff` command. For example, traditionally engraved chamber music scores with piano often used 7 mm piano staves while the other staves were typically between 3/5 and 5/7 as large (between 60% and 71%). To achieve the 5/7 proportion, use:

   ```
   \score {
     <<
       \new Staff \with {
         \magnifyStaff #5/7
       } { ... }
       \new PianoStaff { ... }
     >>
   }
   ```

   If you happen to know which `fontSize` you wish to use, you could use the following form:

   ```
   \score {
     <<
       \new Staff \with {
         \magnifyStaff #(magstep -3)
       } { ... }
       \new PianoStaff { ... }
   ```

```
    >>
  }
```

To emulate the look of traditional engraving, it is best to avoid reducing the thickness of the staff lines.

Automatic font weight at different sizes

The Emmentaler font provides the set of *Feta* musical glyphs in eight different sizes; each one tuned for a different staff size. The smaller the glyph size, the "heavier" it becomes, so as to match the relatively thicker staff lines. Recommended glyphs 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	

See also
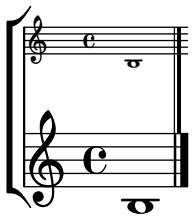
Notation Reference: Section 7.1.1 [Selecting notation font size], page 278, Section B.8 [The Emmentaler font], page 907.

Snippets: Section "Spacing" in *Snippets*.

Known issues and warnings

When using `\magnifyStaff` only for some staves in a `StaffGroup`, `BarLine` grobs do not align any more, due to the changed `BarLine` properties `thick-thickness`, `hair-thickness` and `kern`.

```
\new StaffGroup
  <<
    \new Staff \with { \magnifyStaff #1/2 } { b1 \bar "|." }
    \new Staff { b }
  >>
```



You may want to cancel magnifying `BarLine` grobs, mimic them on the other staves or apply intermediate values for every `Staff`.

```
#(define bar-line-props
  '((BarLine thick-thickness)
    (BarLine hair-thickness)
    (BarLine kern)))

mus = { b1 \bar "|."}

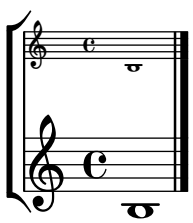
\markup "Cancel \\magnifyStaff for bar lines:"
```

```
\new StaffGroup
  <<
    \new Staff
      \with {
        \magnifyStaff #1/2
        #(revert-props 'magnifyStaff 0 bar-line-props)
      }
      \mus
    \new Staff
      \mus
  >>
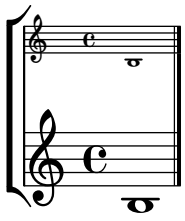
\markup "Mimic \\magnifyStaff on other staves:"
\new StaffGroup
  <<
    \new Staff
      \with { \magnifyStaff #1/2 }
      \mus
    \new Staff
      \with {
        #(scale-props 'magnifyStaff 1/2 #t bar-line-props)
      }
      \mus
  >>

\markup "Apply an intermediate value to all staves:"
\new StaffGroup
  <<
    \new Staff
      \with {
        \magnifyStaff #1/2
        #(scale-props 'magnifyStaff 3/2 #t bar-line-props)
      }
      \mus
    \new Staff
      \with {
        #(scale-props 'magnifyStaff 3/4 #t bar-line-props)
      }
      \mus
  >>
```

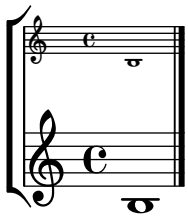
Cancel \magnifyStaff for bar lines:



Mimic \magnifyStaff on other staves:



Apply an intermediate value to all staves:
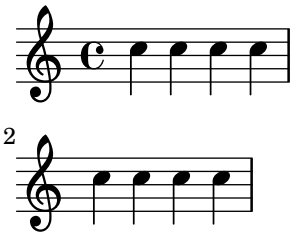


28 Breaks

28.1 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.

To manually force a line break at a bar line, use the `\break` command:

```
\relative c'' {
  c4 c c c | \break
  c4 c c c |
}
```



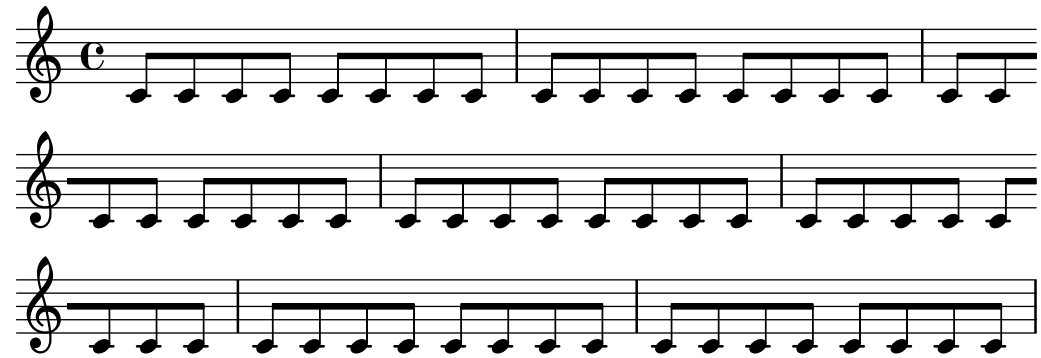
By default, breaks are only allowed at bar lines. There are also a few other factors that can prevent a break from being allowed at a certain bar line:

- a note head or rest continuing over the bar line,
- the presence of an 'unbreakable' spanner, such as a beam or a glissando, crossing the bar line.

The `\break` command forces a break in all cases, regardless of the presence of a bar line or any other factor. It is also possible to bypass all these factors using the `\allowBreak` command. In the following example, breaks are allowed everywhere, even in the middle of a measure, and despite the presence of beams.

```
\*56 { c'8 \allowBreak }

\paper {
  indent = 0
  line-width = 135\mm
}
```



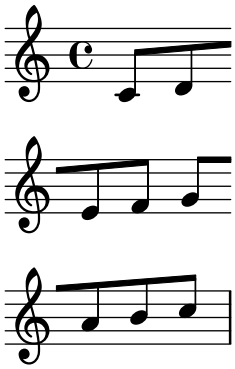
If you find yourself using `\allowBreak` often, you may want to prevent some of the factors mentioned above from disabling breaks.

- `Bar_engraver` forbids breaks between bar lines when `forbidBreakBetweenBarLines` is set to `#t`. To inhibit this, set the property to `#f`.

```
\fixed c' {
  c8 d e f g a b c'
}

\layout {
  \context {
    \Score
    forbidBreakBetweenBarLines = ##f
  }
}
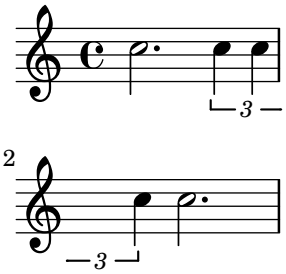
\paper {
  indent = 0
  line-width = 30\mm
}
```



- Note heads and rests extending over bar lines can be made not to suppress breaks by removing the `Forbid_line_break_engraver` from the `Voice` context.

```
\new Voice \with {
  \remove Forbid_line_break_engraver
} \relative {
  c''2. \tuplet 3/2 { c4 c c } c2.
}
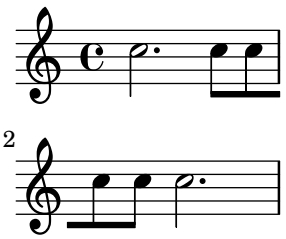
\paper {
  indent = 0
  line-width = 35\mm
}
```



- The presence of beams and other unbreakable spanners over bar lines is ignored if their `breakable` property is set to `#t`.

```
\relative c'' {
  \override Beam.breakable = ##t
  c2. c8[ c |
  c8 c] c2. |
}

\paper {
  indent = 0
  line-width = 35\mm
}
```



The \noBreak command will prevent a line break at the bar line where it is inserted.

Within a score, automatic line breaking is prevented within music lying between \autoLineBreaksOff and \autoLineBreaksOn commands. If automatic page breaks should also be prevented, the commands \autoBreaksOff and \autoBreaksOn should be used. Manual breaks are unaffected by these commands. Note that inhibiting automatic line breaks may cause music to run over the right margin if it cannot all be contained within one line.

Automatic line breaks (but not page breaks) may be enabled at single bar lines by using \once \autoLineBreaksOn at a bar line. This identifies a permitted rather than a forced line break.

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 #t 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\mm
  line-width = 150\mm
  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 actual music... }
>>
```

Predefined commands

\break, \allowBreak, \noBreak, \autoBreaksOff, \autoBreaksOn, \autoLineBreaksOff, \autoLineBreaksOn.

Selected snippets

Using an extra voice for breaks

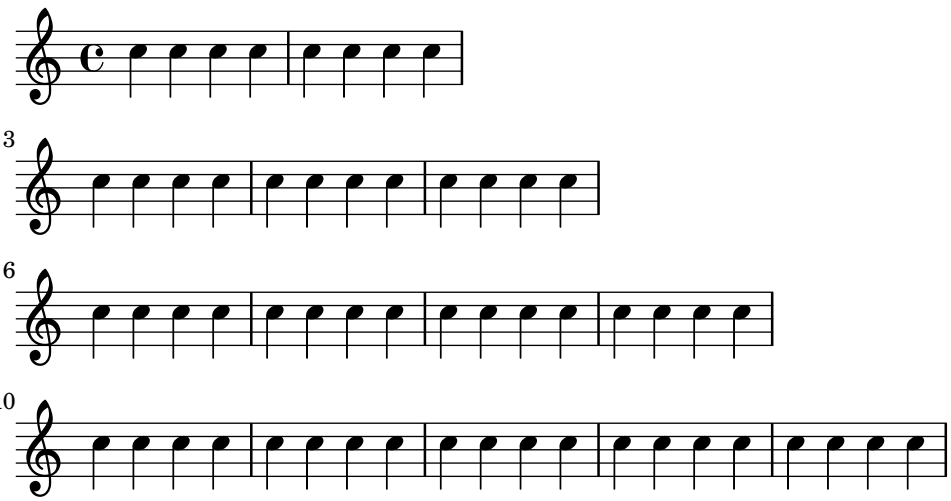
Often it is easier to manage line and page-breaking information by keeping it separate from the music by introducing an extra voice containing only skips along with the \break, \pageBreak, and other layout information.

This pattern becomes especially helpful when overriding line-break-system-details and the other useful but long properties of the NonMusicalPaperColumn grob.

```
music = \relative c'' { c4 c c c }

\score {
  \new Staff <<
    \new Voice {
      s1*2 \break
      s1*3 \break
      s1*4 \break
      s1*5 \break
    }
    \new Voice {
      \*2 \music
      \*3 \music
      \*4 \music
      \*5 \music
    }
  >>
}

\paper {
  indent = 0
  line-width = 140\mm
  ragged-right = ##t
}
```



See also

Notation Reference: Section 26.6.1 [\paper variables for line breaking], page 679, Section 27.1 [The \layout block], page 683.

Snippets: Section "Spacing" in *Snippets*.

Internals Reference: Section "LineBreakEvent" in *Internals Reference*.

Known issues and warnings

Placing \autoLineBreaksOff or \autoBreaksOff before any music will cause error messages to appear. Always place these commands after some music.

28.2 Page breaking

This section describes the different page breaking methods, and how to modify them.

28.2.1 Manual 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 at that bar line. 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.

Within a score, automatic page breaks are prevented within music lying between \autoPageBreaksOff and \autoPageBreaksOn commands. Manual page breaks are unaffected by these commands.

There are also analogous settings to ragged-right and ragged-last which have the same effect on vertical spacing. If ragged-bottom is set to #t the systems will not be justified vertically. When ragged-last-bottom is set to #t, as it is by default, empty space is allowed at the bottom of the final page (or the final page in each \bookpart). See Section 26.3 [Fixed vertical spacing \paper variables], page 673.

Page breaks are computed by the page-breaking function. LilyPond provides several algorithms for computing page breaks, including 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 {
  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
     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`, `\autoPageBreaksOn`, `\autoPageBreaksOff`.

See also

Notation Reference: Section 26.6.2 [`\paper` variables for page breaking], page 679.

Snippets: Section "Spacing" in *Snippets*.

Known issues and warnings

The `\once` prefix is ineffective with `\autoPageBreaksOn` and `\autoPageBreaksOff`. If automatic page breaking is off and is then turned on to permit a page break, it must remain on for a few bars (the precise number of bars depends on the score) before being turned off, else the opportunity to break the page will not be taken.

28.2.2 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*.

28.2.3 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 {
  page-breaking = #ly:minimal-breaking
}
```

See also

Snippets: Section "Spacing" in *Snippets*.

28.2.4 One-page page breaking

The `ly:one-page-breaking` function is a special-purpose page breaking algorithm that automatically adjusts the page height to fit the music, so that everything fits on a single page. The `paper-height` variable in the paper block is ignored, but other settings work as usual. In particular, the spacing between the last system (or top level markup) and the footer can be customized

with last-bottom-spacing in the paper block. The width of the page is left unmodified by default but can be set with paper-width in the paper block.

Known issues and warnings

ly:one-page-breaking is not currently compatible with \bookpart.

28.2.5 One-line page breaking

The ly:one-line-breaking function is a special-purpose page breaking algorithm that puts each score on its own page, and on a single line. No titles or margins are typeset; only the score is displayed.

The page width is adjusted so that the longest score fits on one line. In particular, paper-width, line-width and indent variables in the \paper block are ignored, although left-margin and right-margin are still honored. The height of the page is left unmodified.

28.2.6 One-line-auto-height page breaking

The ly:one-line-auto-height-breaking function works just like ly:one-line-breaking except the page height is automatically modified to fit the height of the music. Specifically, the paper-height variable in the \paper block is set so that it spans the height of the tallest score plus the top-margin and bottom-margin.

Note that the top-system-spacing setting will affect the vertical position of the music. Set it to #f in a paper block to simply place the music between the top and bottom margins.

28.2.7 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 28.2 [Page breaking], page 693. 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 pageTurnMinimumRestLength to determine how long a note-free section must be before a page turn is considered. The default value for pageTurnMinimumRestLength is 1. If you want to disable page turns, 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.pageTurnMinimumRestLength = #5/2
```

```
    R1 | % a page turn will not be allowed here
    a4 b r2 |
    R1*2 | % a page turn will be allowed here
    a1
  }
```

When using volta repeats, the Page_turn_engraver 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. If the repeat is too short then the Page_turn_engraver can be used to *disable* page turns by setting an appropriate value for the context property pageTurnMinimumRepeatLength. In this case the Page_turn_engraver will only allow turns in repeats whose duration is longer than the value specified.

The page turning commands, \pageTurn, \noPageTurn and \allowPageTurn, may also be used at top-level, in top-level markups and between scores.

Predefined commands

\pageTurn, \noPageTurn, \allowPageTurn.

See also

Notation Reference: Section 26.6.1 [\paper variables for line breaking], page 679.

Snippets: Section "Spacing" in *Snippets*.

Known issues and warnings

Use only one Page_turn_engraver per score. If there are more, they will interfere with each other.

See also

Notation Reference: Chapter 29 [Vertical spacing], page 697.

Snippets: Section "Spacing" in *Snippets*.

29 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.

29.1 Flexible vertical spacing within systems

Three separate mechanisms control the flexible vertical spacing within systems, one for each of the following categories:

- *ungrouped staves*,
- *grouped staves* (staves within a staff group such as `ChoirStaff`, etc.), and
- *non-staff lines* (such as `Lyrics`, `ChordNames`, etc.).

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 are distributed between the staves.

Note that the spacing mechanisms discussed in this section only control the vertical spacing of staves and non-staff lines within individual systems. The vertical spacing between separate systems, scores, markups, and margins is controlled by `\paper` variables, which are discussed in Section 26.4 [Flexible vertical spacing \paper variables], page 674.

29.1.1 Within-system spacing properties

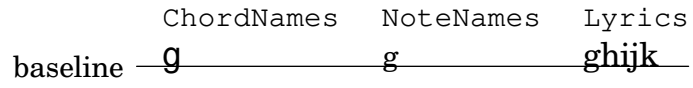
The within-system vertical spacing mechanisms are controlled by two sets of grob properties. The first set is associated with the `VerticalAxisGroup` grob, which is created by all staves and non-staff lines. The second set is associated with the `StaffGrouper` grob, which can be created by staff groups, but only if explicitly called. These properties are described individually at the end of this section.

The names of these properties (except for `staff-affinity`) follow the format *`item1`*`-`*`item2`*`-spacing`, where *`item1`* and *`item2`* are the items to be spaced. Note that *`item2`* is not necessarily below *`item1`*; for example, `nonstaff-relatedstaff-spacing` will measure upwards from the non-staff line if `staff-affinity` is `UP`.

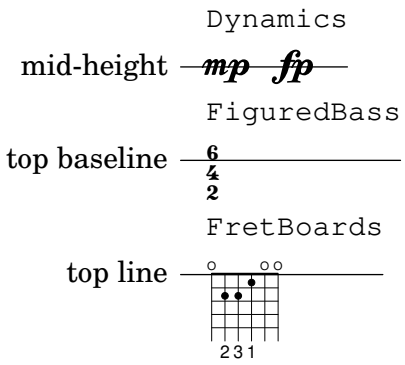
Each distance is measured between the *reference points*[1] of the two items. The reference point for a staff is the vertical center of its `StaffSymbol` (i.e., the middle line if `line-count` is odd; the middle space if `line-count` is even). The reference points for individual non-staff lines are given in the following table:

Non-staff line	**Reference point**
`ChordNames`	baseline
`NoteNames`	baseline
`Lyrics`	baseline
`Dynamics`	mid-height of 'm'
`FiguredBass`	baseline of topmost element
`FretBoards`	top line

In the following image, horizontal lines indicate the positions of these reference points:



[1] This is a sloppy formulation. For contexts, the positions of grobs along the x-axis are computed by LilyPond's spacing algorithm (using `PaperColumn` objects and the like for alignment); this means that the only relevant information here is the vertical position. In other words, a reference 'point' of a context is the y-coordinate position to which other grobs are aligned to.



Each of the vertical spacing grob properties (except `staff-affinity`) uses the same alist structure as the `\paper` spacing variables discussed in Section 26.4 [Flexible vertical spacing \paper variables], page 674. Specific methods for modifying alists are discussed in Section 35.9 [Modifying alists], page 771. Grob properties should be adjusted with an `\override` inside a `\score` or `\layout` block, and not inside a `\paper` block.

The following example demonstrates the two ways these alists can be modified. The first declaration updates one key value individually, and the second completely redefines the property:

```
\new Staff \with {
  \override VerticalAxisGroup
            .default-staff-staff-spacing.basic-distance = 10
} { ... }

\new Staff \with {
  \override VerticalAxisGroup.default-staff-staff-spacing =
    #'((basic-distance . 10)
       (minimum-distance . 9)
       (padding . 1)
       (stretchability . 10))
} { ... }
```

To change any spacing settings globally, put them in the `\layout` block:

```
\layout {
  \context {
    \Staff
    \override VerticalAxisGroup
              .default-staff-staff-spacing
              .basic-distance = 10
  }
}
```

Standard settings for the vertical spacing grob properties are listed in Section "VerticalAxisGroup" in *Internals Reference* and Section "StaffGrouper" in *Internals Reference*. Default overrides for specific types of non-staff lines are listed in the relevant context descriptions in Section "Contexts" in *Internals Reference*.

Properties of the `VerticalAxisGroup` grob

`VerticalAxisGroup` properties are typically adjusted with an `\override` at the `Staff` level (or equivalent).

`staff-staff-spacing`
: Used to determine the distance between the current staff and the staff just below it in the same system, even if one or more non-staff lines (such as `Lyrics`) are placed between the two staves. Does not apply to the bottom staff of a system.

Initially, the staff-staff-spacing of a VerticalAxisGroup is a Scheme function that applies the properties of the StaffGrouper if the staff is part of a group, or the default-staff-staff-spacing of the staff otherwise. This allows staves to be spaced differently when they are grouped. For uniform spacing regardless of grouping, this function may be replaced by a flexible-spacing alist, using the complete-redefinition form of override shown above. If only some values are specified in an override, missing values will be taken from default-staff-staff-spacing (if it has values for them).

default-staff-staff-spacing
: A flexible-spacing alist defining the staff-staff-spacing used for ungrouped staves, unless staff-staff-spacing has been explicitly set with an \override.

staff-affinity
: The direction of the staff to use for spacing the current non-staff line. Choices are UP, DOWN, and CENTER. If CENTER, the non-staff line will be placed equidistant between the two nearest staves on either side, unless collisions or other spacing constraints prevent this. Adjacent non-staff lines should have non-increasing staff-affinity from top to bottom, e.g., a non-staff line set to UP should not immediately follow one that is set to DOWN. Non-staff lines at the top of a system should use DOWN; those at the bottom should use UP. Setting staff-affinity for a staff causes it to be treated as a non-staff line. Setting staff-affinity to #f causes a non-staff line to be treated as a staff. Setting staff-affinity to UP, CENTER, or DOWN causes a staff to be spaced as a non-staff line.

nonstaff-relatedstaff-spacing
: The distance between the current non-staff line and the nearest staff in the direction of staff-affinity, if there are no non-staff lines between the two, and staff-affinity is either UP or DOWN. If staff-affinity is CENTER, then nonstaff-relatedstaff-spacing is used for the nearest staves on *both* sides, even if other non-staff lines appear between the current one and either of the staves. This means that the placement of a non-staff line depends on both the surrounding staves and the surrounding non-staff lines. Setting the stretchability of one of these types of spacing to a small value will make that spacing dominate. Setting the stretchability to a large value will make that spacing have little effect.

nonstaff-nonstaff-spacing
: The distance between the current non-staff line and the next non-staff line in the direction of staff-affinity, if both are on the same side of the related staff, and staff-affinity is either UP or DOWN.

nonstaff-unrelatedstaff-spacing
: The distance between the current non-staff line and the staff in the opposite direction from staff-affinity, if there are no other non-staff lines between the two, and staff-affinity is either UP or DOWN. 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.

Properties of the StaffGrouper grob

StaffGrouper properties are typically adjusted with an \override at the StaffGroup level (or equivalent).

staff-staff-spacing
: The distance between consecutive staves within the current staff group. The staff-staff-spacing property of an individual staff's VerticalAxisGroup grob can be overridden with different spacing settings for that staff.

staffgroup-staff-spacing
: The distance between the last staff of the current staff group and the staff just below it in the same system, even if one or more non-staff lines (such as Lyrics) exist between the two staves. Does not apply to the bottom staff of a system. The staff-staff-spacing property of an individual staff's VerticalAxisGroup grob can be overridden with different spacing settings for that staff.

See also

Notation Reference: Section 26.4 [Flexible vertical spacing \paper variables], page 674, Section 35.9 [Modifying alists], page 771.

Installed Files: ly/engraver-init.ly, scm/define-grobs.scm.

Internals Reference: Section "Contexts" in *Internals Reference*, Section "VerticalAxisGroup" in *Internals Reference*, Section "StaffGrouper" in *Internals Reference*.

29.1.2 Spacing of ungrouped staves

Staves (such as Staff, DrumStaff, TabStaff, etc.) are contexts that can contain one or more voice contexts, but cannot contain any other staves.

The following properties affect the spacing of *ungrouped* staves:

- VerticalAxisGroup properties:
  - default-staff-staff-spacing
  - staff-staff-spacing

These grob properties are described individually above; see Section 29.1.1 [Within-system spacing properties], page 697.

Additional properties are involved for staves that are part of a staff group; see Section 29.1.3 [Spacing of grouped staves], page 701.

The following example shows how the default-staff-staff-spacing property can affect the spacing of ungrouped staves. The same overrides applied to staff-staff-spacing would have the same effect, but would also apply in cases where the staves are combined in a group or groups.
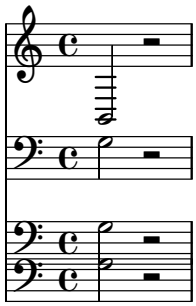
```
\layout {
  \context {
    \Staff
    \override VerticalAxisGroup.default-staff-staff-spacing =
      #'((basic-distance . 8)
         (minimum-distance . 7)
         (padding . 1))
  }
}

<<
  % The very low note here needs more room than 'basic-distance
  % can provide, so the distance between this staff and the next
  % is determined by 'padding.
  \new Staff { b,2 r | }

  % Here, 'basic-distance provides enough room, and there is no
  % need to compress the space (towards 'minimum-distance) to make
  % room for anything else on the page, so the distance between
  % this staff and the next is determined by 'basic-distance.
```

```
  \new Staff { \clef bass g2 r | }

  % By setting 'padding to a negative value, staves can be made to
  % collide.  The lowest acceptable value for 'basic-distance is 0.
  \new Staff \with {
    \override VerticalAxisGroup.default-staff-staff-spacing =
      #'((basic-distance . 3.5)
         (padding . -10))
  } { \clef bass g2 r | }
  \new Staff { \clef bass g2 r | }
>>
```



See also

Installed Files: `scm/define-grobs.scm`.

Snippets: Section "Spacing" in *Snippets*.

Internals Reference: Section "VerticalAxisGroup" in *Internals Reference*.

29.1.3 Spacing of grouped staves

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.

Staff-groups (such as `StaffGroup`, `ChoirStaff`, etc.) are contexts that can contain one or more staves simultaneously.

The following properties affect the spacing of staves inside staff groups:

- `VerticalAxisGroup` properties:
  - `staff-staff-spacing`
- `StaffGrouper` properties:
  - `staff-staff-spacing`
  - `staffgroup-staff-spacing`

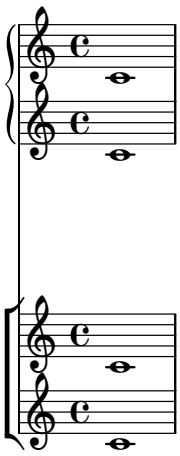
These grob properties are described individually above; see Section 29.1.1 [Within-system spacing properties], page 697.

The following example shows how properties of the `StaffGrouper` grob can affect the spacing of grouped staves:

```
\layout {
  \context {
    \Score
    \override StaffGrouper.staff-staff-spacing.padding = 0
    \override StaffGrouper.staff-staff-spacing.basic-distance = 1
  }
}
```

```
<<
  \new PianoStaff \with {
    \override StaffGrouper
              .staffgroup-staff-spacing
              .basic-distance = 20
  } <<
    \new Staff { c'1 }
    \new Staff { c'1 }
  >>

  \new StaffGroup <<
    \new Staff { c'1 }
    \new Staff { c'1 }
  >>
>>
```



See also

Installed Files: `scm/define-grobs.scm`.

Snippets: Section "Spacing" in *Snippets*.

Internals Reference: Section "VerticalAxisGroup" in *Internals Reference*, Section "StaffGrouper" in *Internals Reference*.

29.1.4 Spacing of non-staff lines

Non-staff lines (such as `Lyrics`, `ChordNames`, etc.) are contexts whose layout objects are engraved like staves (i.e., in horizontal lines within systems). Specifically, non-staff lines are non-staff contexts that contain the Section "Axis_group_engraver" in *Internals Reference*.

The following properties affect the spacing of non-staff lines:

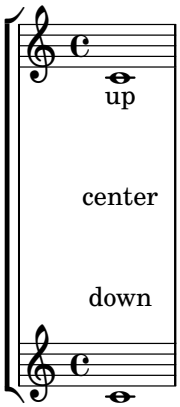
- `VerticalAxisGroup` properties:
  - `staff-affinity`
  - `nonstaff-relatedstaff-spacing`
  - `nonstaff-nonstaff-spacing`
  - `nonstaff-unrelatedstaff-spacing`

These grob properties are described individually above; see Section 29.1.1 [Within-system spacing properties], page 697.

The following example shows how the nonstaff-nonstaff-spacing property can affect the spacing of consecutive non-staff lines. Here, by setting the stretchability key to a very high value, the lyrics are able to stretch much more than usual:

```
\layout {
  \context {
    \Lyrics
    \override VerticalAxisGroup
              .nonstaff-nonstaff-spacing
              .stretchability = 1000
  }
}

\new StaffGroup
<<
  \new Staff \with {
    \override VerticalAxisGroup.staff-staff-spacing =
      #'((basic-distance . 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

Installed Files: ly/engraver-init.ly, scm/define-grobs.scm.

Snippets: Section “Spacing” in *Snippets*.

Internals Reference: Section “Contexts” in *Internals Reference*, Section “VerticalAxisGroup” in *Internals Reference*.
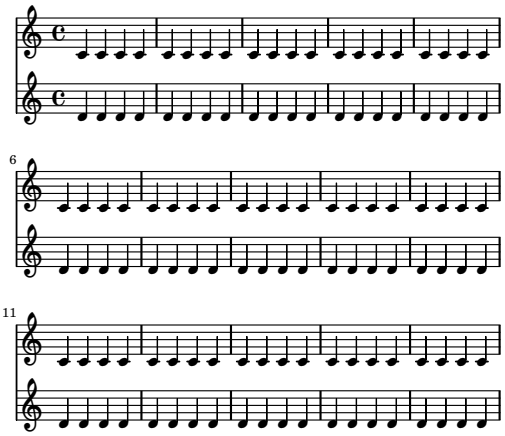
29.2 Explicit staff and system positioning

It is possible to approach vertical spacing in a different way (in comparison to what was demonstrated in the previous sections) by using the property `line-break-system-details` of the `NonMusicalPaperColumn` grob. While the flexible vertical spacing mechanisms specify vertical padding, `line-break-system-details` can specify exact vertical positions on the page.

This property contains five different subproperties: `X-offset`, `Y-offset`, `extra-offset`, `alignment-distances`, and `bottom-padding`.

To understand how each of these different subproperties work, we begin by looking at an example that includes no overrides at all. We isolate both line breaking and page breaking information in a dedicated voice, dividing the music into five measures per line. This technique of creating an otherwise empty voice for breaks helps in keeping layout separate from music entry as our example becomes more complicated. Also see Chapter 28 [Breaks], page 689.
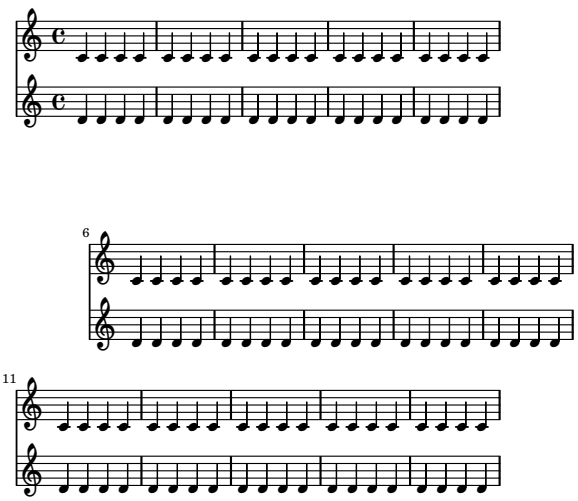
```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          s1*5 \break
          s1*5 \break
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new Staff { \*15 { d'4 d' d' d' } }
    >>
  }
}
```



LilyPond v2.27.5

We now set the vertical and horizontal start points of each system explicitly by using the `Y-offset` and `X-offset` subproperties of `line-break-system-details`.

```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          \once \override
            Score.NonMusicalPaperColumn
                 .line-break-system-details.Y-offset = 0
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn
                 .line-break-system-details.Y-offset = 30
          \once \override
            Score.NonMusicalPaperColumn
                 .line-break-system-details.X-offset = 10
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn
                 .line-break-system-details.Y-offset = 50
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new Staff { \*15 { d'4 d' d' d' } }
    >>
  }
}
```
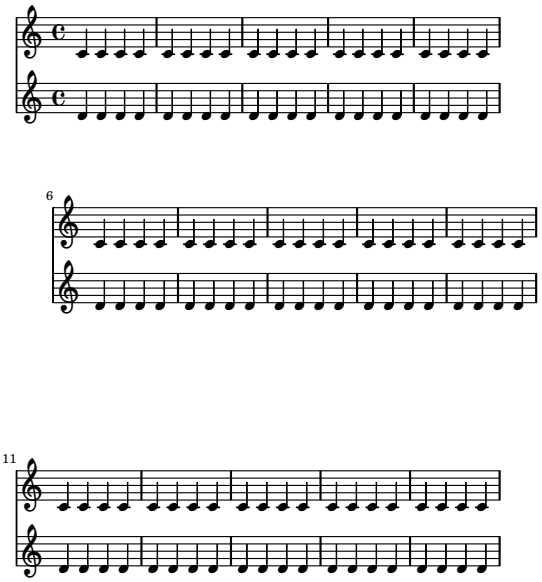


LilyPond v2.27.5

In contrast to the absolute positioning available through `Y-offset` and `X-offset`, relative positioning is possible with the `extra-offset` subproperty. Placement is relative to the de-
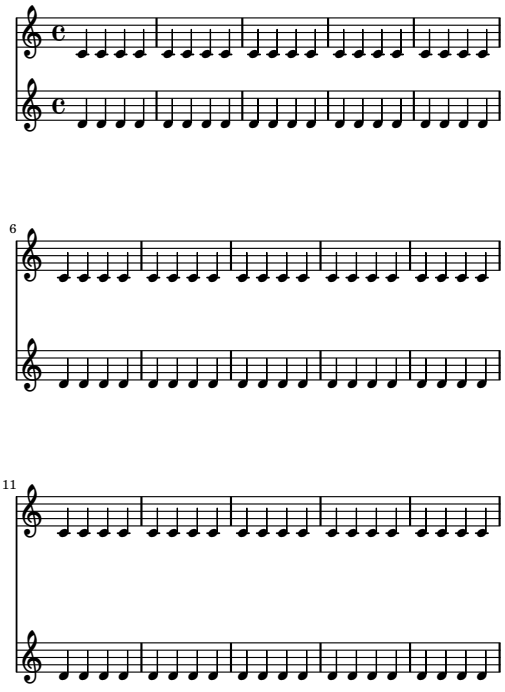
fault layout or to the absolute positioning created by setting `X-offset` and `Y-offset`. The subproperty `extra-offset` accepts a `pair` consisting of displacements along the X- and Y-axis.

```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn
              .line-break-system-details.extra-offset = #'(5 . 5)
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn
              .line-break-system-details.extra-offset = #'(0 . 20)
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new Staff { \*15 { d'4 d' d' d' } }
    >>
  }
}
```



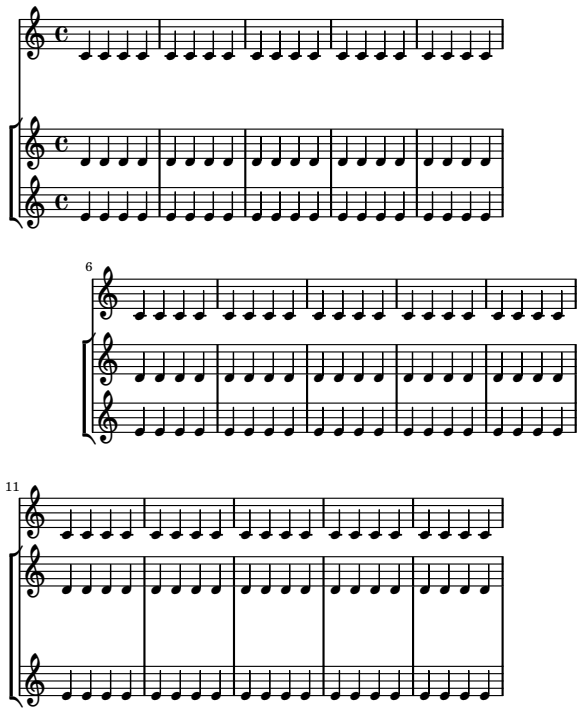
Now that we have set the vertical start point of each system explicitly, we can also set the vertical distances between staves within each system manually by modifying the `alignment-distances` subproperty. As can be seen, we use an alist to set two different `line-break-system-details` subproperties at the same time.

```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
              #'((Y-offset . 0)
                 (alignment-distances . (10)))
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
              #'((Y-offset . 30)
                 (alignment-distances . (15)))
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
              #'((Y-offset . 65)
                 (alignment-distances . (20)))
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new Staff { \*15 { d'4 d' d' d' } }
    >>
  }
}
```
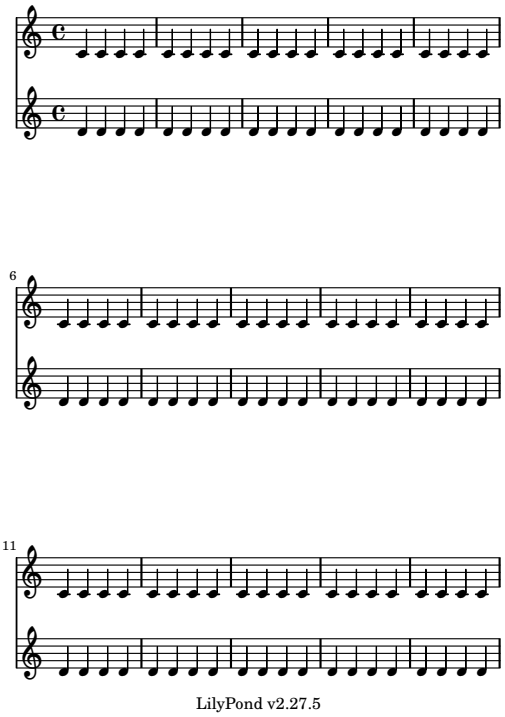


The next example shows how to set vertical positions of multiple staves in a system. Note that `alignment-distances` directly applies to staves, not taking care of grouping with `StaffGroup` or the like. Value `#f` indicates that the default value should be used.

```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
              #'((Y-offset . 0)
                 (alignment-distances . (15 8)))
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
              #'((Y-offset . 35)
                 (X-offset . 10)
                 (alignment-distances . (#f 8)))
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
              #'((Y-offset . 65)
                 (alignment-distances . (8 15)))
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new StaffGroup <<
        \new Staff { \*15 { d'4 d' d' d' } }
        \new Staff { \*15 { e'4 e' e' e' } }
      >>
    >>
  }
}
```



The last subproperty of `line-break-system-details`, `bottom-padding`, has a similar effect as `last-bottom-spacing`. Note, however, that it doesn't hold a compressible or stretchable spacing dimension but a rigid, fixed padding value from the bottom of the page to the lowest staff. Here is the test snippet without a ragged bottom, showing the default distance.
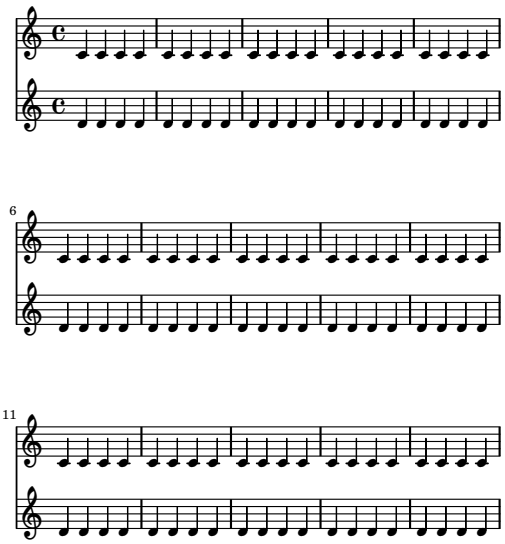
```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          s1*5 \break
          s1*5 \break
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new Staff { \*15 { d'4 d' d' d' }
      }
    >>
  }
  \paper { ragged-last-bottom = ##f }
}
```



Setting `bottom-padding` yields the following result.

```
\book {
  \score {
    <<
      \new Staff <<
        \new Voice {
          s1*5 \break
          s1*5 \break
          \once \override
            Score.NonMusicalPaperColumn.line-break-system-details =
```

```
            #'((bottom-padding . 20))
          s1*5 \break }
        \new Voice { \*15 { c'4 c' c' c' } }
      >>
      \new Staff { \*15 { d'4 d' d' d' } }
    >>
  }
  \paper { ragged-last-bottom = ##f }
}
```



LilyPond v2.27.5

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 the `X-offset`, `Y-offset`, `extra-offset`, and `alignment-distances` subproperties 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. Use reasonable values for the subproperties to avoid this.

See also

Snippets: Section "Spacing" in *Snippets*.

29.3 Vertical collision avoidance

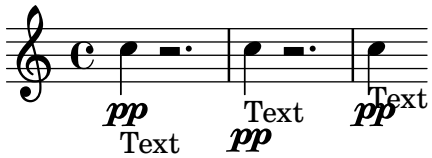
Intuitively, there are some objects in musical notation that belong to the staff and there are other objects that should be placed outside the staff. Objects belonging outside the staff include things such as rehearsal marks, text and dynamic markings (from now on, these will be called outside-staff 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.
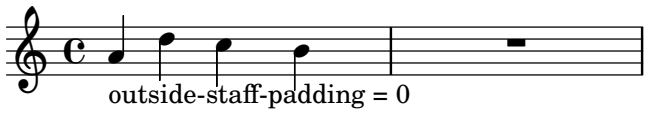
A listing of defaults for `outside-staff-priority` may be found in Section B.17 [Default values for outside-staff-priority], page 937.

```
\relative c'' {
  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 around outside-staff objects can be controlled with property `outside-staff-padding`.

```
\relative {
  \once \override TextScript.outside-staff-padding = 0
  a'4-"outside-staff-padding = 0"
  \once \override TextScript.outside-staff-padding = 3
  d-"outside-staff-padding = 3"
  c-"default outside-staff-padding"
  b-"default outside-staff-padding"
  R1
}
```



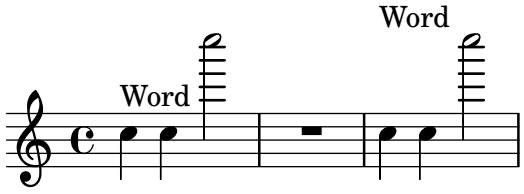
outside-staff-padding = 3

default outside-staff-padding
default outside-staff-padding

By default, outside-staff objects are placed so they avoid a horizontal collision with previously-positioned grobs. This can lead to situations in which objects are placed close to each other

horizontally. As shown in the example below, setting `outside-staff-horizontal-padding` increases the horizontal spacing required, and in this case moves the text up to prevent it from getting too close to the ledger lines.

```
\relative {
  c''4^"Word" c c''2
  R1
  \once \override TextScript.outside-staff-horizontal-padding = 1
  c,,4^"Word" c c''2
}
```



See also

Snippets: Section “Spacing” in *Snippets*.
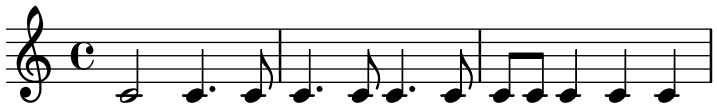
30 Horizontal spacing

30.1 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 `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.

```
\relative c' {
  c2 c4. c8
  c4. c8 c4. c8
  c8 c c4 c c
}
```



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.
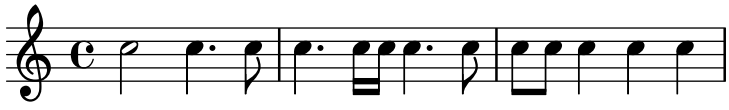
If one would follow the above procedure exactly, then adding a single 32nd note to a score that uses 8th and 16th notes, would widen up the entire score a lot. The shortest note is no longer a 16th, but a 32nd, thus adding 1 NHW to every note. To prevent this, the shortest duration for spacing is not 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.

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:

```
\relative { c''2 c4. c8 | c4. c16[ c] c4. c8 | c8 c c4 c c }
```
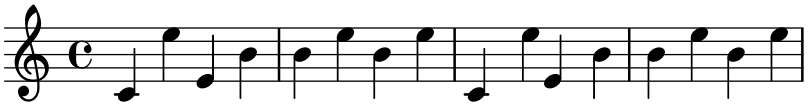


As explained in the *Essay on automated music engraving*, stem directions will influence spacing (see Section "Optical spacing" in *Essay*) and can be adjusted using the

stem-spacing-correction property of the Section "NoteSpacing" in *Internals Reference* object (which 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 this; once with the default settings and once with an exaggerated adjustment:



Proportional notation is supported; see Section 30.6 [Proportional notation], page 722.

See also

Essay on automated music engraving: Section "Optical spacing" in *Essay*.

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 workaround may be used to insert extra space into a score, adjusting the padding value as necessary.

```
\override Score.NonMusicalPaperColumn.padding = 10
```
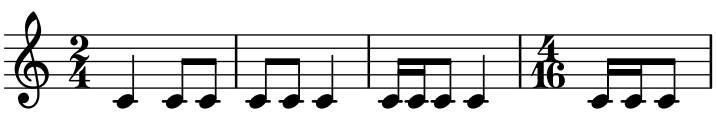
No workaround exists for decreasing the amount of space.

30.2 New spacing section

New sections with different spacing parameters can be started with the newSpacingSection command. This is useful for sections with different notions of 'long' and 'short' notes. The \newSpacingSection command creates a new SpacingSpanner object at that musical moment.
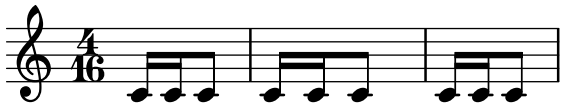
In the following example the time signature change introduces a new section, and the 16ths notes are automatically spaced slightly wider apart.

```
\relative c' {
  \time 2/4
  c4 c8 c
  c8 c c4 c16[ c c8] c4
  \newSpacingSection
  \time 4/16
  c16[ c c8]
}
```



If the automatic spacing adjustments do not give the required spacing, manual \overrides may be applied to its properties. These must be applied at the same musical moment as the \newSpacingSection command itself and will then affect the spacing of all the following music until the properties are changed in a new spacing section, for example:

```
\relative c' {
  \time 4/16
  c16[ c c8]
  \newSpacingSection
  \override Score.SpacingSpanner.spacing-increment = 2
  c16[ c c8]
  \newSpacingSection
  \revert Score.SpacingSpanner.spacing-increment
  c16[ c c8]
}
```



See also
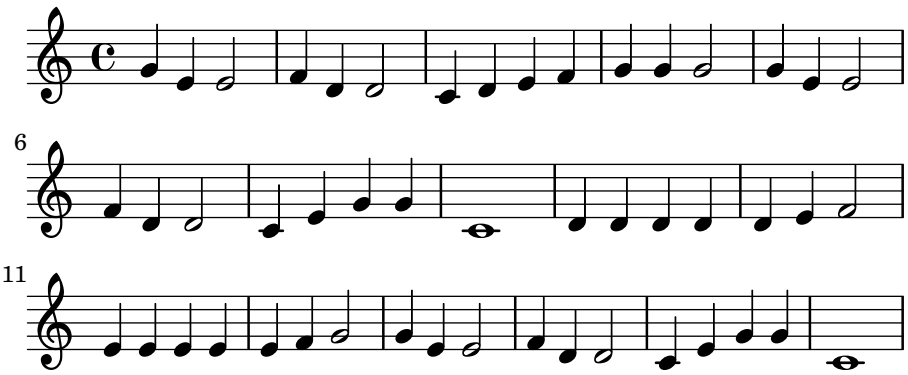
Snippets: Section “Spacing” in *Snippets*.

Internals Reference: Section “SpacingSpanner” in *Internals Reference*.

30.3 Changing horizontal spacing globally
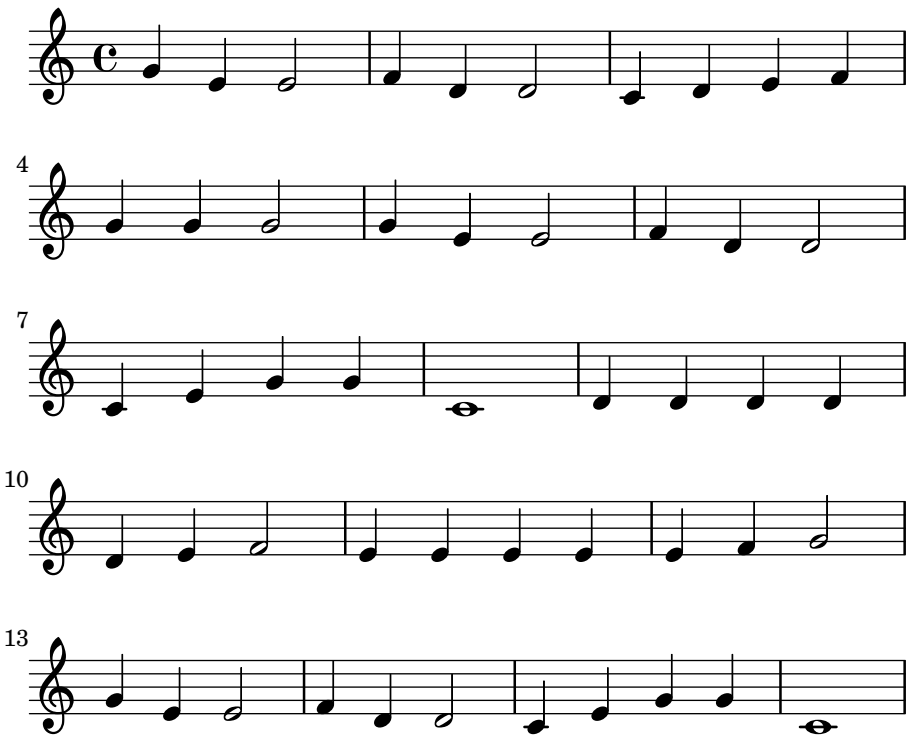
Horizontal spacing may be altered with the `base-shortest-duration` property. Here we compare the same music; once without altering the property, and then altered. Larger values yield tighter spacing.

```
\score {
  \relative {
    g'4 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 {
    g'4 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 {
```

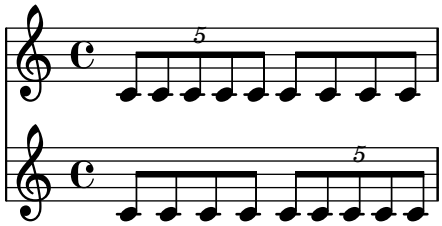
```
    \context {
      \Score
      \override SpacingSpanner.base-shortest-duration =
        \musicLength 16
    }
  }
}
```



30.3.1 Uniform stretching of tuplets

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,

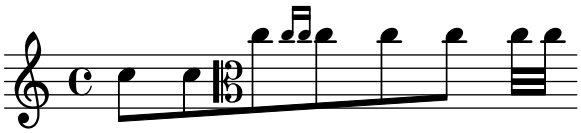
```
\score {
  <<
    \new Staff \relative c' {
      \tuplet 5/4 { c8 c c c c } c8 c c c
    }
    \new Staff \relative c' {
      c8 c c c \tuplet 5/4 { c8 c c c c }
    }
  >>
  \layout {
    \context {
      \Score
      \override SpacingSpanner.uniform-stretching = ##t
    }
  }
}
```



30.3.2 Strict note spacing

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 \relative {
  c''8[ c \clef alto c \grace { c16 c } c8 c c]  c32[ c] }
```



See also

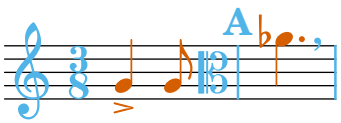
Snippets: Section "Spacing" in *Snippets*.

30.4 Adjusting horizontal spacing for specific layout objects

In addition to the general-purpose parameters of the default spacing algorithm that apply to all elements of the score or spacing section, several properties serve to operate adjustments on a per-object-type-basis. Examples include adjusting the distance from clefs to time signatures, but not from clefs to notes when there is no time signature, or stretching notes further apart in the presence of a printed text so that it does not overlap with the next note. Tweaking these first requires identifying the type of spacing at play.

30.4.1 Overview of object-specific horizontal spacing tweaks

Layout objects that have a horizontally fixed position are called *items* (as opposed to *spanners*) and, for the purpose of horizontal spacing, are grouped into *columns*. Note heads and rests, forming the main musical material, together with those objects that are logically linked to them – accidentals, articulations, stems, dots, etc. – are all part of 'musical columns' (represented by `NoteColumn` grobs). *Prefatory matter*, such as clefs, time signatures and bar lines, is grouped into 'non-musical columns' (represented by `NonMusicalPaperColumn` grobs). In the following example, musical items are colored red, while non-musical items are blue.
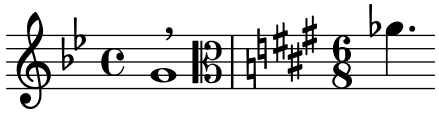


This example shows that there is an alternation between musical and non-musical columns. The first non-musical column contains a clef and a time signature. The first musical column has a note head with its stem and articulation. The second non-musical column is empty and thus removed during the layout process. The second musical column has a note again. The third non-musical column contains a clef, a bar line and a rehearsal mark, etc.

Within one column, spacing is fixed. On the other hand, the amount of space between consecutive columns is flexible. As we shall see, the methods to adjust spacing within a column and between columns are different.

30.4.2 Spacing between adjacent non-musical items

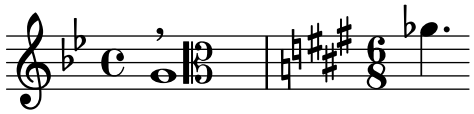
Within a non-musical column, items are laid out in a specific order. For instance, with the set of items in the picture below, the default order places the breathing sign first, then the clef, then the bar line, the key cancellation and key signature, and finally the time signature (this is controlled by the `BreakAlignment.break-align-orders` property, which may conveniently be changed using `\breakAlignInsert`).

```
\relative {
  \key g \minor
  g'1
  \breathe
  \clef alto
  \time 6/8
  \key a \major
  aes4.
}
```



The distance between two adjacent items from the same non-musical column is controlled by the value of the `space-alist` property of the leftmost one of the two. `space-alist` has the form of an associative list mapping break-align symbols to (*`spacing-style . value`*) pairs. A breakable item's break-align symbol is given by the value of its `break-align-symbol` property; standard choices are listed in Section "break-alignment-interface" in *Internals Reference*. Spacing styles are listed in Section "break-aligned-interface" in *Internals Reference*. Among the available options, only `extra-space` and `minimum-space` are relevant for tweaking the space between non-musical items. The difference is that `extra-space` measures the padding from the right of the first object to the left of the second object while `minimum-space` counts from the left of the first object. Thus, a way to move the bar line farther from the clef is:

```
\relative {
  \key g \minor
  g'1
  \override Staff.Clef.space-alist.staff-bar = #'(extra-space . 4)
  \breathe
  \clef alto
  \time 6/8
  \key a \major
  aes4.
}
```



`space-alist` settings, not limited to the two spacing styles described above, are also possible to override the spacing between different columns. However, this kind of spacing is flexible, and does not merely depend on the types of object involved but also their shapes. Methods specific to it are documented in the next section.

Selected snippets

Separating key cancellations from key signature changes

By default, the accidentals used for key cancellations are placed adjacent to those for key signature changes. This behavior can be changed by overriding the `break-align-orders` property of the `BreakAlignment` grob.

If you look up the definition of the `break-alignment-interface` in LilyPond's Internals Reference (`https://lilypond.org/doc/v2.24/Documentation/internals/break_002dalignment_002dinterface`), you get the following list of available break-align symbols:

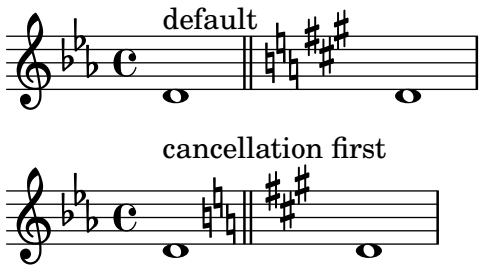
```
ambitus
breathing-sign
clef
cue-clef
cue-end-clef
custos
key-cancellation
key-signature
left-edge
signum-repetitionis
staff-bar
staff-ellipsis
time-signature
```

From this list, we find that we want to move `key-cancellation` so that it comes before `staff-bar`. This is accomplished with the `\breakAlignInsert` function.

```
music = { \key es \major d'1 \bar "||"
          \key a \major d'1 }

{ <>^\markup "default"
  \music }

{ <>^\markup "cancellation first"
  \breakAlignInsert key-cancellation before staff-bar
  \music }
```



See also

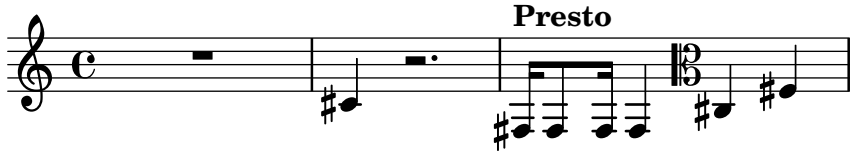
Notation Reference: Section 36.9.4 [Using the `break-alignable-interface`], page 797.

Extending LilyPond: Section "Association lists (alists)" in *Extending*.

Internals Reference: Section "Break_align_engraver" in *Internals Reference*, Section "BreakAlignGroup" in *Internals Reference*, Section "BreakAlignment" in *Internals Reference*, Section "break-alignable-interface" in *Internals Reference*, Section "break-aligned-interface" in *Internals Reference*, Section "break-alignment-interface" in *Internals Reference*.

30.4.3 Spacing between adjacent columns

Contrary to spacing within one column, spacing between adjacent columns is flexible and stretches or compresses according to the density of music on the line. By default, columns may even overlap in some situations. The following example shows three cases. The second accidental slides behind the bar line, while the third one overlaps with the clef. Also, the tempo marking *Presto* spans several columns. Observe how the first accidental, which remains within the vertical extent of the bar line on its left, is placed further apart.

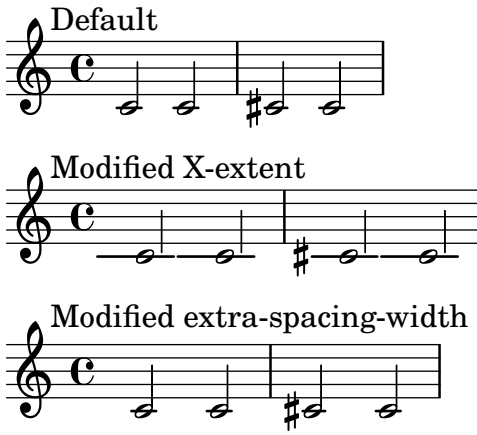


These spacing rules can be overridden. This is done by modifying the width and height that an object takes in horizontal spacing. The relevant properties are `extra-spacing-width` and `extra-spacing-height`. When unset, an object takes as much space in horizontal spacing as its `X-extent` and `Y-extent` properties allow. These are accurate values of its dimensions. The `extra-spacing-width` and `extra-spacing-height` properties make an object larger or smaller for computation of horizontal spacing between columns only, but preserve its dimensions for other spacing types.

```
{
  \textMark "Default"
  c'2 2 cis'2 2
}

{
  \textMark "Modified X-extent"
  \override NoteHead.X-extent = #'(-2 . 2)
  c'2 2 cis'2 2
}

{
  \textMark "Modified extra-spacing-width"
  \override NoteHead.extra-spacing-width = #'(-2 . 2)
  c'2 2 cis'2 2
}
```
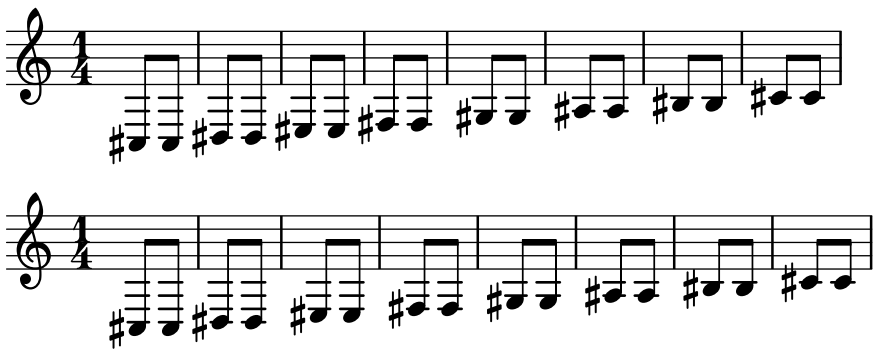


`extra-spacing-width` and `extra-spacing-height` are pairs of numbers, which are added to the dimensions on each axis. For instance, setting `extra-spacing-height` to `'(-2 . 3)` makes

the object three units larger on the top, and two units larger on the bottom (limit lowered by 2). The following example shows how to use `extra-spacing-height` to change the limit after which accidentals no longer overlap with bar lines.

```
music = \relative {
  \time 1/4
  cis8 8 | dis8 8 | eis8 8 | fis8 8 |
  gis8 8 | ais8 8 | bis8 8 | cis8 8 |
}

{
  \music
}
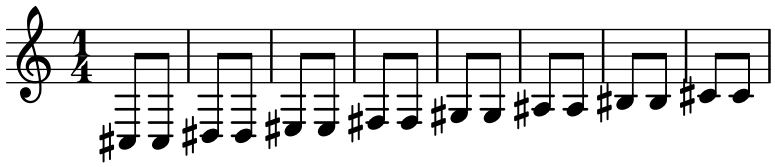
{
  \override Accidental.extra-spacing-height = #'(0 . 1.0)
  \music
}
```



The value `'(+inf.0 . -inf.0)` for `extra-spacing-width` or `extra-spacing-height` removes the object's presence.
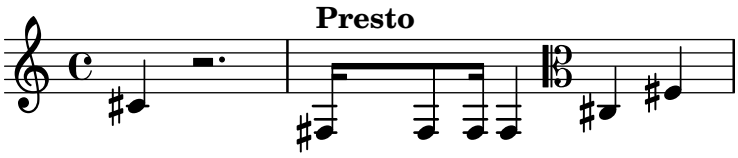
```
music = \relative {
  \time 1/4
  cis8 8 | dis8 8 | eis8 8 | fis8 8 |
  gis8 8 | ais8 8 | bis8 8 | cis8 8 |
}

{
  \override Accidental.extra-spacing-height = #'(+inf.0 . -inf.0)
  \music
}
```



Conversely, an `extra-spacing-height` of `'(-inf.0 . +inf.0)` makes the object infinitely high, preventing overlap with another column completely. The below example demonstrates this technique on `Accidental` and `MetronomeMark`. In the case of `MetronomeMark`, it is necessary to set `extra-spacing-width` to `'(0 . 0)` because the default is `'(+inf.0 . -inf.0)`, and even an infinitely high object does not take space if it has no width.

```
{
  \override Score.MetronomeMark.extra-spacing-width =
    #'(0 . 0)
  \override Score.MetronomeMark.extra-spacing-height =
    #'(-inf.0 . +inf.0)
  \override Accidental.extra-spacing-height =
    #'(-inf.0 . +inf.0)
  cis'4 r2.
  \tempo Presto
  fis16 8 16 4 \clef alto cis4 fis4
}
```



See also

Internals Reference: Section “item-interface” in *Internals Reference*, Section “separation-item-interface” in *Internals Reference*.

30.5 Line width

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 `#t` 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*.

30.6 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.

The following settings for proportional notation are provided, which may be used together or alone:

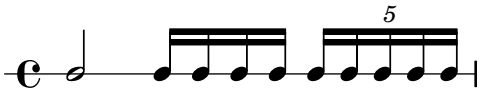
- `proportionalNotationDuration`

- uniform-stretching
- strict-note-spacing
- \remove Separating_line_group_engraver

In the examples that follow, we explore these different proportional notation settings and examine how they interact.

We start with the following one-measure example, which uses classical spacing with ragged-right turned on.

```
\new RhythmicStaff {
  c2 16 16 16 16 \tuplet 5/4 { 16 16 16 16 16 }
}
```

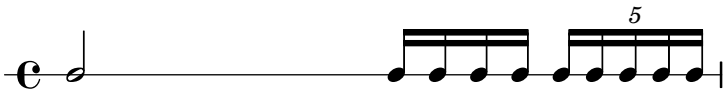


Notice that the half note starting the measure takes up far less than half of the horizontal space of the measure. Likewise, the sixteenth notes and sixteenth-note quintuplets (or twentieth notes) ending the measure together take up far more than half the horizontal space of the measure.

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 some other graphic above or below our score, we need proportional notation. We turn proportional notation on with the proportionalNotationDuration setting.

```
\new RhythmicStaff {
  c2 16 16 16 16 \tuplet 5/4 { 16 16 16 16 16 }
}
\layout {
  \context {
    \Score
    proportionalNotationDuration = #1/20
  }
}
```



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. Remember that context settings can appear in one of three locations within our input file – in a \with block, in a \context block, or directly in the music entry preceded by the \set command. As with all context settings, users can pick in which of the three different locations they would like to set proportionalNotationDuration.

The proportionalNotationDuration setting takes a single argument, which is the reference length to space all music. The value #1/20 specifies one twentieth of a whole note.

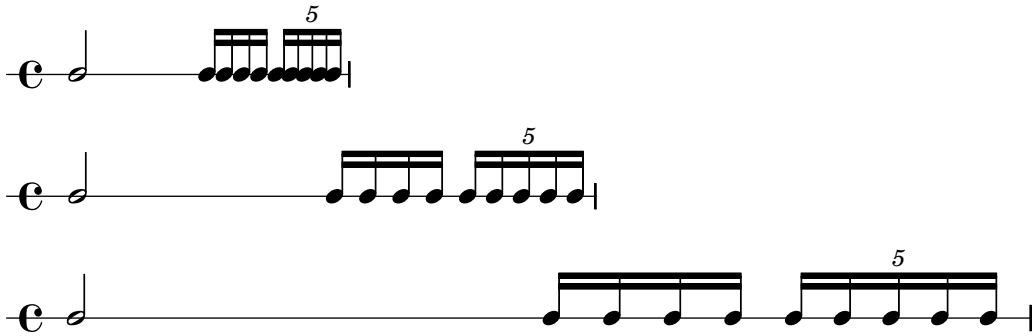
How do we select the right reference duration to pass to proportionalNotationDuration? Usually by a process of trial and error, beginning with a duration close to the fastest (or smallest) duration in the piece. Smaller reference durations space music loosely; larger reference durations space music tightly.

```
rhythm = { c2 16 16 16 16 \tuplet 5/4 { 16 16 16 16 16 } }

\new RhythmicStaff {
  \set Score.proportionalNotationDuration = #1/8
  % Allow overlapping of note heads.
  \override NoteHead.extra-spacing-width = #'(+inf.0 . -inf.0)
  \rhythm
}

\new RhythmicStaff {
  \set Score.proportionalNotationDuration = #1/16
  \rhythm
}

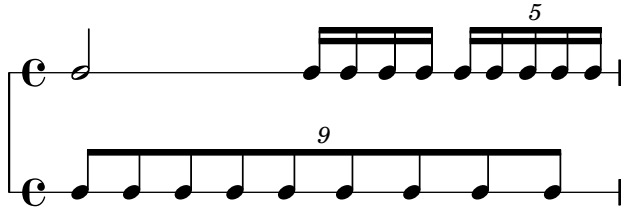
\new RhythmicStaff {
  \set Score.proportionalNotationDuration = #1/32
  \rhythm
}
```



Note that too large a reference duration – such as the eighth note, above – spaces music too tightly and can cause note head collisions. In general, proportional notation takes up more horizontal space than classical spacing. Proportional spacing provides rhythmic clarity at the expense of horizontal space.

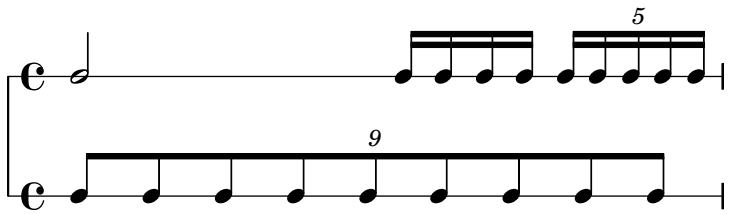
Next we examine how to optimally space overlapping tuplets. We start by examining what happens to our original example, with classical spacing, when we add a second staff with a different type of tuplet.

```
<<
  \new RhythmicStaff {
    c2 16 16 16 16 \tuplet 5/4 { 16 16 16 16 16 }
  }
  \new RhythmicStaff {
    \tuplet 9/8 { c8 8 8 8 8 8 8 8 8 }
  }
>>
```



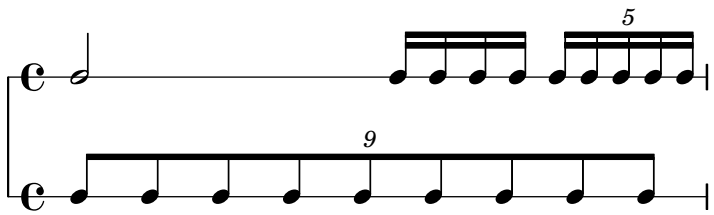
The spacing is bad because the evenly spaced notes of the bottom staff do not stretch uniformly. Classical engravings include very few complex triplets and so classical engraving rules can generate this type of result. Setting `proportionalNotationDuration` fixes this.

```
<<
  \new RhythmicStaff {
    c2 16 16 16 16 \tuplet 5/4 { 16 16 16 16 16 }
  }
  \new RhythmicStaff {
    \tuplet 9/8 { c8 8 8 8 8 8 8 8 8 }
  }
>>
\layout {
  \context {
    \Score
    proportionalNotationDuration = #1/20
  }
}
```



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 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 RhythmicStaff {
    c2 16 16 16 16 \tuplet 5/4 { 16 16 16 16 16 }
  }
  \new RhythmicStaff {
    \tuplet 9/8 { c8 8 8 8 8 8 8 8 8 }
  }
>>
\layout {
  \context {
    \Score
    proportionalNotationDuration = #1/20
    \override SpacingSpanner.uniform-stretching = ##t
  }
}
```



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.

It is recommended to always set the SpacingSpanner's uniform-stretching property to #t if you use proportionalNotationDuration. Omitting it, for example, causes skips to consume an incorrect amount of horizontal space.

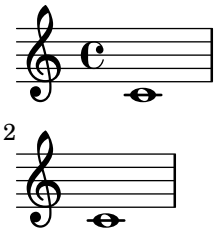
The SpacingSpanner is an abstract grob that lives in the Score context. As with our settings of proportionalNotationDuration, overrides to the SpacingSpanner can occur in any of three different places in our input file – in the Score's \with block, in a Score's \context block, or directly in the note entry.

By default, there is only one SpacingSpanner per Score. This means that 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 by using the command \newSpacingSection. See Section 30.2 [New spacing section], page 714, 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 "prefatory" space just before the first note in each system.

```
\paper {
  indent = 0
}

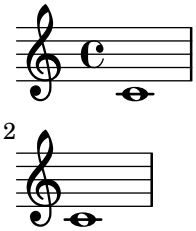
\new Staff {
  c'1 \break
  c'1
}
```



The amount of this prefatory space stays the same regardless whether a time signature, a key signature, or a clef follows. Separating_line_group_engraver is responsible for this space, and removing this engraver reduces the prefatory space to zero.

```
\paper {
  indent = 0
}

\new Staff \with {
  \remove Separating_line_group_engraver
} {
  c'1 \break
  c'1
}
```



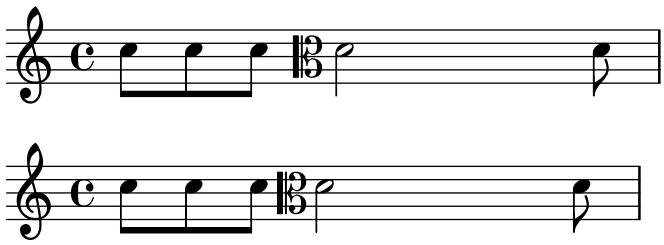
Non-musical elements like time signatures, key signatures, clefs, and accidentals are problematic in proportional notation. None of these elements has rhythmic duration, but all of them consume horizontal space. Different proportional scores approach these problems differently.

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. However, such scores are exceptional, and most proportional scores do include at least some time signatures. Clefs and accidentals are even more essential.

So what strategies exist for spacing non-musical elements in a proportional context? One good option is the `strict-note-spacing` property of `SpacingSpanner`. Compare the two scores below:

```
{
  \set Score.proportionalNotationDuration = #1/16
  c''8 8 8 \clef alto d'2 d'8
}

{
  \set Score.proportionalNotationDuration = #1/16
  \override Score.SpacingSpanner.strict-note-spacing = ##t
  c''8 8 8 \clef alto d'2 d'8
}
```



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 this property 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.

`\override SpacingSpanner.strict-grace-spacing = ##t`
: space grace notes strictly (see [Positioning grace notes with floating space], page 152)

`\set tupletFullLength = ##t`
: extend tuplet brackets to mark both rhythmic start and stop points

`\override Beam.breakable = ##t`
: permit broken beams (see [Beams across line breaks], page 103)

`\override Glissando.breakable = ##t`
: permit broken glissandi (see [Making glissandi breakable], page 180)

`\remove Forbid_line_break_engraver`
: allow line breaks even if a musical element is still active (see Section 28.1 [Line breaking], page 689)

See also

Notation Reference: Section 30.2 [New spacing section], page 714.

Snippets: Section “Spacing” in *Snippets*.

31 Fitting music onto fewer pages

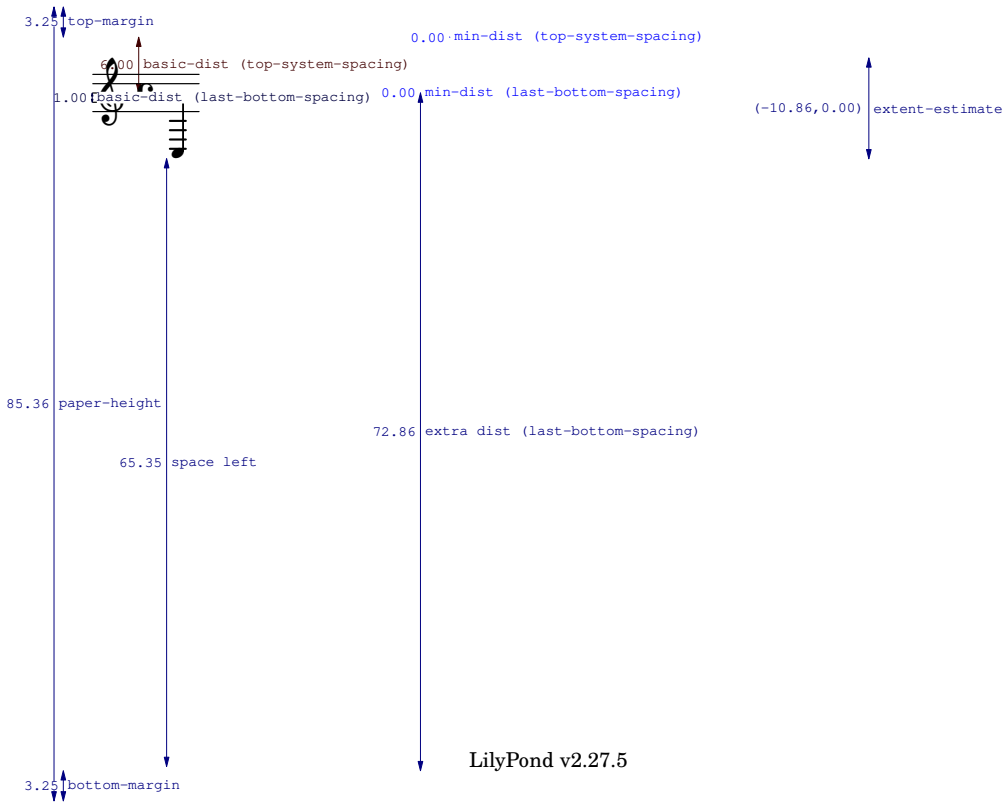
Sometimes you can end up with one or two staves on a second (or third, or fourth...) page. This is annoying, especially if you look at previous pages and it looks like there is plenty of room left on those.

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 31.1 [Displaying spacing], page 729.

31.1 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:

```
\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 point = (25.4/72.27) mm

1 staff space = (`staff-size`)/4 pts
= (`staff-size`)/4 * (25.4/72.27) mm

In this case, one staff space is approximately equal to 1.757 mm. 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 27.2 [Setting the staff size], page 685.

Snippets: Section "Spacing" in *Snippets*.

31.2 Changing spacing

The output of `annotate-spacing` reveals vertical dimensions in great detail. For details about modifying margins and other layout variables, see Chapter 26 [Page layout], page 671.

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 {
  system-system-spacing = #'((basic-distance . 0.1) (padding . 0))
  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.
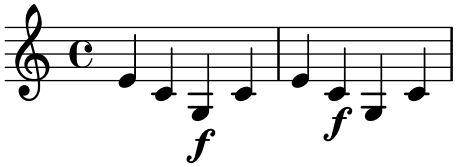
```
\paper {
  system-count = 10
}
```

- Force the number of pages. For example, the following assignment will force a layout with 2 pages.

```
\paper {
  page-count = 2
}
```
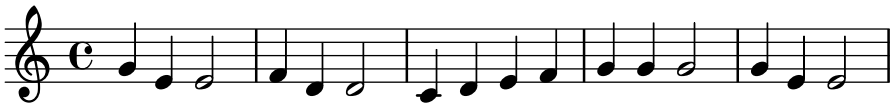
- Avoid (or reduce) objects that increase the vertical size of a system. For example, volta brackets for alternative repeat endings require extra space. If these endings are spread over two systems, they take up more space than if they were on the same system. As another example, dynamics that 'stick out' of a system can be moved closer to the staff:

```
\relative e' {
  e4 c g\f c
  e4 c g-\tweak X-offset -2.7 \f c
}
```
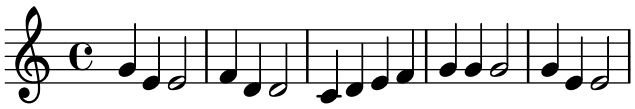


- Alter the horizontal spacing via SpacingSpanner. For more details, see Section 30.3 [Changing horizontal spacing globally], page 715. The following example illustrates the default spacing:

```
\score {
  \relative {
    g'4 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 {
    g'4 e e2 |
    f4 d d2 |
    c4 d e f |
    g4 g g2 |
    g4 e e2 |
  }
  \layout {
    \context {
      \Score
      \override SpacingSpanner.common-shortest-duration =
        \musicLength 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: Chapter 26 [Page layout], page 671, Section 30.3 [Changing horizontal spacing globally], page 715.

Snippets: Section "Spacing" in *Snippets*.

Changing defaults

32 Tuning output

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: the *Internals Reference*. That manual lists all the variables, functions, and options available in LilyPond.

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]

[1] Section "Scheme tutorial" in *Extending*, contains a short tutorial on entering numbers, lists, strings, and symbols in Scheme.

33 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*.

33.1 Contexts explained

Contexts are arranged hierarchically.

33.1.1 Output definitions – blueprints for contexts

This section explains the relevance of output definitions when working with *contexts*. Examples for actual output definitions are given later (see Section 33.5.1 [Changing all contexts of the same type], page 746).

While music written in a file may refer to context types and names, contexts are created only when the music is actually being interpreted. LilyPond interprets music under control of an *output definition* and may do so for several different output definitions, resulting in different output. The output definition relevant for printing music is specified using `\layout`.

A much simpler output definition used for producing MIDI output is specified using `\midi`. Several other output definitions are used by LilyPond internally, like when using the part combiner (Section 5.2.5 [Automatic part combining], page 233) or creating music quotes (Section 6.3.2 [Quoting other voices], page 266).

Output definitions define the relation between contexts as well as their respective default settings. While most changes are usually made inside of a `\layout` block, MIDI-related settings only have an effect when made within a `\midi` block.

Some settings affect several outputs: for example, if `autoBeaming` is turned off in some context, beams count as melismata for the purpose of matching music to lyrics, see Section 9.1.4 [Automatic syllable durations], page 349. This matching is done both for printed output as well as for MIDI – if changes made to `autoBeaming` within a context definition of a `\layout` block are not repeated in the corresponding `\midi` block, lyrics and music will get out of sync in MIDI output.

See also

Installed Files: `ly/engraver-init.ly`. `ly/performer-init.ly`.

33.1.2 `Score` – the master of all contexts

This is the top-level notation context.[1] No other context can contain a `Score` context. This context handles the administration of time signatures. It also makes sure that items such as clefs, time signatures, and key signatures are aligned across staves.

You cannot explicitly instantiate a `Score` context (since it is not contained in any other context). It is instantiated automatically when an output definition (a `\score` or `\layout` block) is processed.

[1] There exists a context called `Global` that is even one level higher than `Score`, and which is the hard-coded entry point for LilyPond. However, this is not meant to be modified; you will need this context only if you are going to implement an engraver in Scheme and can be safely ignored otherwise.

An alias called Timing is established by the Timing_translator in whatever context it is initialized, and the timing variables are then copied from wherever Timing had been previously established. The alias at Score level provides a target for initializing Timing variables in layout definitions before any Timing_translator has been run.

33.1.3 Top-level contexts – staff containers

StaffGroup
: Connect staves vertically by adding a bracket on the left side. The bar lines of the contained staves are connected vertically, too.

ChoirStaff
: Identical to StaffGroup except that the bar lines of the contained staves are not connected vertically.

GrandStaff
: Connect staves vertically by adding a brace on the left side. The bar lines of the contained staves are connected vertically, too.

PianoStaff
: Just like GrandStaff, but the staves are only removed together, never separately.

OneStaff
: Provides a common axis for the contained staves, making all of them appear in the same vertical space. This can be useful for typesetting staves of different types in immediate succession or for temporarily changing the character of one staff or overlaying it with a different one. Often used with \stopStaff and \startStaff for best results.

VaticanaScore
: Top-level context replacing Score for Gregorian chant notated in Vaticana style. Compared to Score, it changes the staff line color to red, uses packed spacing, and removes bar numbers.

ChordGridScore
: Top-level context replacing Score in chord grid notation. Compared to Score, it uses proportional notation, and has a few other settings like removing bar numbers.

StandaloneRhythmScore
: A Score-level context for use by \markup \rhythm.

33.1.4 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 when engraving; the notes are printed on one line. The MIDI rendition retains pitches unchanged.

TabStaff
: Context for generating tablature. It accepts only TabVoice contexts and handles the line spacing, the tablature clef, etc., properly.

DrumStaff
: Handles typesetting for percussion. Can contain DrumVoice.

VaticanaStaff
: A kind of Staff for typesetting Gregorian chant in a notational style approximating *Editio Vaticana*.

MensuralStaff
: Same as Staff context, except that it is accommodated for typesetting a piece in mensural style.

`PetrucciStaff`
: A kind of `Staff` approximating the mensural typesetting of Ottaviano Petrucci's *Harmonices Musices Odhecaton* (Venice, 1501).

`KievanStaff`
: Same as `Staff` context, except that it is accommodated for typesetting a piece in Kievan style.

`GregorianTranscriptionStaff`
: A staff for notating Gregorian chant in modern style.

`ChordGrid`
: Creates chord grid notation. This context is always part of a `ChordGridScore` context.

`StandaloneRhythmStaff`
: A `Staff`-level context for use by `\markup \rhythm`.

`FretBoards`
: A context for displaying fret diagrams.

`Devnull`
: Silently discard all musical information given to this context.

33.1.5 Bottom-level contexts – voices

`Voice`-level contexts initialize certain properties and start appropriate engravers. A bottom-level context is one without `\defaultchild`. While it is possible to let it accept/contain subcontexts, they can only be created and entered explicitly.

`Voice`
: Corresponds to a voice on a staff. This context handles the conversion of dynamic signs, stems, beams, super- and subscripts, slurs, ties, and rests.

  You have to instantiate this explicitly if you require multiple voices on the same staff.

`VaticanaVoice`
: A kind of `Voice` for typesetting Gregorian chant in a notational style approximating *Editio Vaticana*.

`MensuralVoice`
: Same as `Voice` context, except that it is accommodated for typesetting a piece in mensural style.

`PetrucciVoice`
: A kind of `Voice` approximating the mensural typesetting of Ottaviano Petrucci's *Harmonices Musices Odhecaton* (Venice, 1501).

`KievanVoice`
: Same as `Voice` context, except that it is accommodated for typesetting a piece in Kievan style.

`GregorianTranscriptionVoice`
: A voice for notating Gregorian chant in modern style.

`Lyrics`
: Corresponds to a voice with lyrics. Handles the printing of a single line of lyrics.

`VaticanaLyrics`
: Same as `Lyrics` context, except that it provides a hyphenation style (a single, flush-left hyphen between two syllables) as used in the notational style of *Editio Vaticana*.

`GregorianTranscriptionLyrics`
: A lyrics context for notating Gregorian chant in modern style.

`DrumVoice`
: A voice on a percussion staff.

`FiguredBass`
: The context in which `BassFigure` grobs 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 Section 6.3.3 [Formatting cue notes], page 269. Usually left to be created implicitly.

`ChordNames`
: Typesets chord names.

`NoteNames`
: Typesets note names.

`NullVoice`
: For aligning lyrics without printing notes.

`Devnull`
: Silently discard all musical information given to this context.

`Dynamics`
: Holds a single line of dynamics centered between the staves surrounding this context.

`StandaloneRhythmVoice`
: A `Voice`-level context for use by `\markup \rhythm`.

`Bottom`
: This is a generic bottom-level context, accepted by all intermediate-level contexts. It can be used for situations where the same music should appear, say, in a `Staff` and a `TabStaff` context.

33.2 Creating and referencing contexts
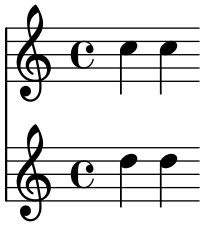
LilyPond creates lower-level contexts automatically if a music expression is encountered before a suitable context exists, but this is usually successful only for simple scores or music fragments like the ones in the documentation. For more complex scores it is advisable to specify all contexts explicitly with either the `\new` or `\context` command. The syntax of these two commands is very similar:

`[\new | \context]` *`Context`* `[=` *`name`*`]` [*`music-expression`*]

where either `\new` or `\context` may be specified. *Context* is the type of context which is to be created, *name* is an optional name to be given to the particular context being created, and *music-expression* is a single music expression that is to be interpreted by the engravers and performers in this context.
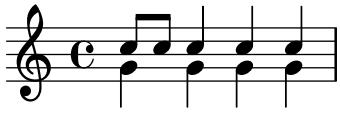
The `\new` prefix without a name is commonly used to create scores with many staves:

```
<<
  \new Staff \relative {
    % leave the Voice context to be created implicitly
    c''4 c
  }
  \new Staff \relative {
    d''4 d
  }
>>
```



and to place several voices into one staff:

```
\new Staff <<
  \new Voice \relative {
    \voiceOne c''8 c c4 c c
  }
  \new Voice \relative {
    \voiceTwo g'4 g g g
  }
>>
```
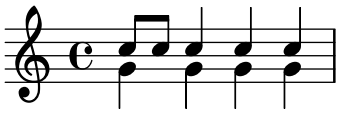


\new should always be used to specify unnamed contexts.

The difference between \new and \context is in the action taken:

- \new with or without a name always creates a fresh, distinct context, even if one with the same name already exists:

```
\new Staff <<
  \new Voice = "A" \relative {
    \voiceOne c''8 c c4 c c
  }
  \new Voice = "A" \relative {
    \voiceTwo g'4 g g g
  }
>>
```
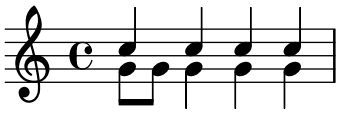


- \context with a name specified creates a distinct context only if a context of the same type with the same name in the same context hierarchy does not already exist. Otherwise it is taken as a reference to that previously created context, and its music expression is passed to that context for interpretation.

  Named contexts may be useful in special cases such as lyrics or figured bass, see Section 9.2.1 [Working with lyrics and variables], page 358, and Section “Vocal ensembles templates” in *Learning Manual* for the former, and Section 15.3.3 [Displaying figured bass], page 531, for the latter. More generally, one application of named contexts is in separating the score layout from the musical content. Either of these two forms is valid:

```
\score {
  <<
    % score layout
    \new Staff <<
      \new Voice = "one" {
        \voiceOne
      }
      \new Voice = "two" {
        \voiceTwo
      }
    >>
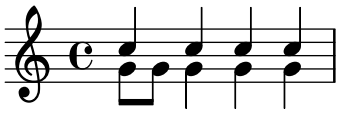
    % musical content
    \context Voice = "one" {
      \relative {
        c''4 c c c
      }
    }
    \context Voice = "two" {
      \relative {
        g'8 g g4 g g
      }
    }
  >>
}
```



```
\score {
  <<
    % score layout
    \new Staff <<
      \context Voice = "one" {
        \voiceOne
      }
      \context Voice = "two" {
        \voiceTwo
      }
    >>

    % musical content
    \context Voice = "one" {
      \relative {
        c''4 c c c
      }
    }
    \context Voice = "two" {
      \relative {
        g'8 g g4 g g
      }
```

```
    }
  >>
}
```



Alternatively, variables may be employed to similar effect. See Section "Organizing pieces with variables" in *Learning Manual*.

- `\context` with no name matches the first of any previously created contexts of the same type in the same context hierarchy, even one that has been given a name, and its music expression is passed to that context for interpretation. This form is rarely useful. However, `\context` with no name and no music expression is used to set the context in which a Scheme procedure specified with `\applyContext` is executed:

```
\new Staff \relative {
  c'1
  \context Timing
    \applyContext #(lambda (ctx)
                     (newline)
                     (display (ly:context-current-moment ctx)))
  c1
}
```

A context must be named if it is to be referenced later, for example when lyrics are associated with music:

```
\new Voice = "tenor" music
...
\new Lyrics \lyricsto "tenor" lyrics
```

For details of associating lyrics with music, see Section 9.1.4 [Automatic syllable durations], page 349.

The properties of all contexts of a particular type can be modified in a `\layout` block (with a different syntax), see Section 33.5.1 [Changing all contexts of the same type], page 746. This construct also provides a means of keeping layout instructions separate from the musical content. If a single context is to be modified, a `\with` block must be used, see Section 33.5.2 [Changing just one specific context], page 749.

See also

Learning Manual: Section "Organizing pieces with variables" in *Learning Manual*.

Notation Reference: Section 33.5.2 [Changing just one specific context], page 749, Section 9.1.4 [Automatic syllable durations], page 349.

33.3 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: inside of a `{...}` construct (sequential music), the construct's notion of the "current context" will descend whenever an element of the sequence

ends in a subcontext of the previous current context. This avoids spurious creation of implicit contexts in a number of situations but means that the first context descended into will be kept alive until the end of the expression.

In contrast, the contexts of a `<<...>>` construct's (simultaneous music) expression are not carried forth, so enclosing a context-creating command in an extra pair of `<<...>>` keeps the context from persisting through all of the enclosing `{...}` sequence.

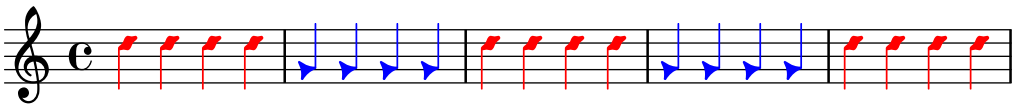
Any context can be kept alive by ensuring it has something to do at every musical moment. `Staff` contexts are kept alive by ensuring one of their voices is kept alive. One way of doing this is to add spacer rests to a voice in parallel with the real music. These need to be added to every `Voice` context 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 { d''4 d d d }
musicB = \relative { g'4 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 }
}

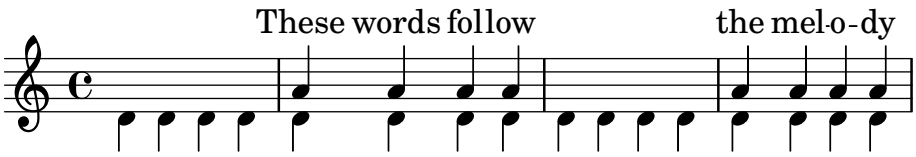
\score {
  \new Staff <<
    \keepVoicesAlive
    \music
  >>
}
```



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 { a'4 a a a }
accompaniment = \relative { d'4 d d d }
words = \lyricmode { These words fol -- low the mel -- o -- dy }

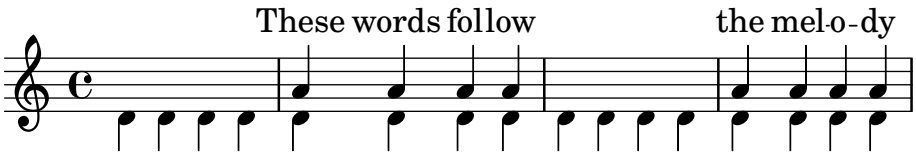
\score {
  <<
    \new Staff = "music" {
      <<
        \new Voice = "melody" {
          \voiceOne s1*4  % keep voice 'melody' alive for 4 bars
        }
        {
          \new Voice = "accompaniment" {
            \voiceTwo \accompaniment
          }
          <<
            \context Voice = "melody" { \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 {
  s1  % skip a bar
  a'4 a a a
  s1  % skip a bar
  a4 a a a
}
accompaniment = \relative {
  d'4 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 }
  >>
}
```



33.4 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_heads_engraver` and the `Staff` context contains a `Key_engraver`.

For a full a description of each plug-in, see Section "Engravers and Performers" in *Internals Reference*. Every context described in Section "Contexts" in *Internals Reference* lists the engravers used for that context.
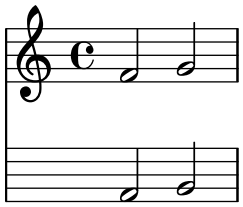
It can be useful to shuffle around these plug-ins. This is done by starting a new context with `\new` or `\context`, and modifying it.

```
\new context \with {
  \consists ...
  \consists ...
  \remove ...
  \remove ...
  etc.
}
{
  ...music...
}
```

where each '`...`' should be the name of an engraver. Here is a simple example that removes `Time_signature_engraver` and `Clef_engraver` from a `Staff` context.

```
<<
  \new Staff \relative {
    f'2 g
  }
  \new Staff \with {
     \remove Time_signature_engraver
```

```
    \remove Clef_engraver
  } \relative {
    f'2 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 affects 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*.

Known issues and warnings

The order in which the engravers are specified is the order in which they are called to carry out their processing. 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 following orderings are important:

- the `Bar_engraver` must normally be first,
- the `New_fingering_engraver` must come before the `Script_column_engraver`,
- the `Timing_translator` must come before the `Bar_number_engraver`.

See also

Installed Files: `ly/engraver-init.ly`.

33.5 Changing context default settings

Context and grob properties can be changed with `\set` and `\override` commands, see Chapter 35 [Modifying properties], page 760. These commands create music events, making the changes take effect at the point in time the music is being processed.

In contrast, this section explains how to change the *default* values of context and grob properties at the time the context is created. There are two ways of doing this. One modifies the default values in all contexts of a particular type, the other modifies the default values in just one particular instance of a context.

33.5.1 Changing all contexts of the same type

The default context settings which are to be used for typesetting in `Score`, `Staff`, `Voice`, and other contexts may be specified in a `\context` block within any `\layout` block.

Settings for MIDI output as opposed to typesetting has to be separately specified in `\midi` blocks (see Section 33.1.1 [Output definitions – blueprints for contexts], page 736).

The `\layout` block should be placed within the `\score` block to which it is to apply, after the music.

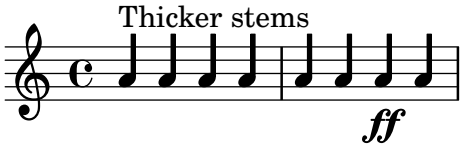
```
\layout {
  \context {
    \Voice
```

```
    [context settings for all Voice contexts]
  }
  \context {
    \Staff
    [context settings for all Staff contexts]
  }
}
```

The following types of settings may be specified:

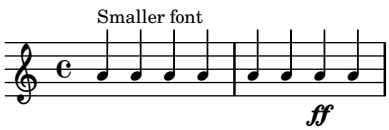
- An `\override` command, but with the context name omitted.

```
\score {
  \relative {
    a'4^"Thicker stems" a a a
    a4 a a\ff a
  }
  \layout {
    \context {
      \Staff
      \override Stem.thickness = 4.0
    }
  }
}
```



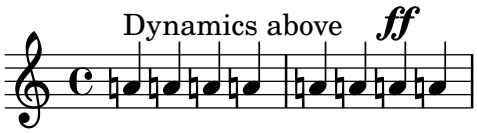
- Directly setting a context property.

```
\score {
  \relative {
    a'4^"Smaller font" a a a
    a4 a a\ff a
  }
  \layout {
    \context {
      \Staff
      fontSize = -4
    }
  }
}
```



- A predefined command such as `\dynamicUp` or a music expression like `\accidentalStyle dodecaphonic`.

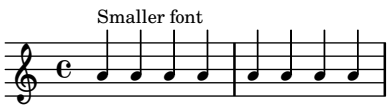
```
\score {
  \relative {
    a'4^"Dynamics above" a a a
    a4 a a\ff a
  }
  \layout {
    \context {
      \Voice
      \dynamicUp
    }
    \context {
      \Staff
      \accidentalStyle dodecaphonic
    }
  }
}
```



- A user-defined variable containing a `\with` block; for details of the `\with` block, see Section 33.5.2 [Changing just one specific context], page 749.
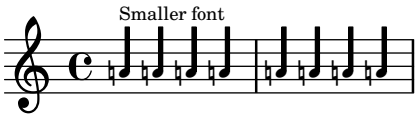
```
StaffDefaults = \with {
  fontSize = -4
}

\score {
  \new Staff {
    \relative {
      a'4^"Smaller font" a a a
      a4 a a a
    }
  }
  \layout {
    \context {
      \Staff
      \StaffDefaults
    }
  }
}
```



Property-setting commands can be placed in a `\layout` block without being enclosed in a `\context` block. Such settings are equivalent to including the same property-setting commands at the start of every context of the type specified. If no context is specified *every* bottom-level context is affected (see Section 33.1.5 [Bottom-level contexts – voices], page 738). The syntax of a property-setting command in a `\layout` block is the same as the same command written in the music stream.

```
\score {
  \new Staff {
    \relative {
      a'4^"Smaller font" a a a
      a4 a a a
    }
  }
  \layout {
    \accidentalStyle dodecaphonic
    \set fontSize = -4
    \override Voice.Stem.thickness = 4.0
  }
}
```



33.5.2 Changing just one specific context

The context properties of just one specific context instance can be changed in a `\with` block. All other context instances of the same type retain the default settings built into LilyPond and modified by any `\layout` block within scope. The `\with` block must be placed immediately after the `\new` *context-type* command:

```
\new Staff \with {
  [context settings for this context instance only]
} {
  ...
}
```

Alternatively, if the music is being entered using the short form of the input mode-specifying commands, e.g., `\chords` rather than `\chordmode`, the `\with` command must be placed immediately after the mode-specifying command:

```
\chords \with {
  [context settings for this (implicit) context instance only]
} {
  ...
}
```

as it is the implicit context created by these short forms which should be modified. The same consideration applies to the other input mode-specifying short forms (`\drums`, `\figures`), see Chapter 19 [Input modes], page 589.

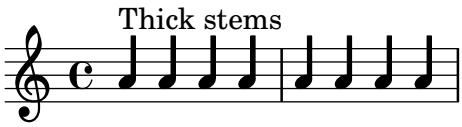
Since context modifications specified in `\with` blocks are inside music, they affect *all* outputs (typesetting *and* MIDI) as opposed to changes within an output definition.

The following types of settings may be specified:

- An `\override` command, but with the context name omitted.

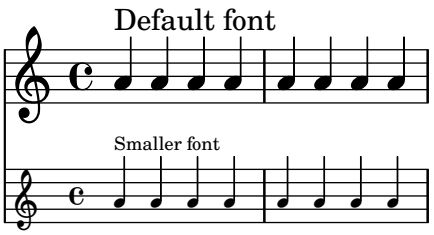
```
\score {
  \new Staff {
    \new Voice \with { \override Stem.thickness = 4.0 } {
      \relative {
        a'4^"Thick stems" a a a
        a4 a a a
      }
```

```
    }
  }
}
```



- Directly setting a context property.

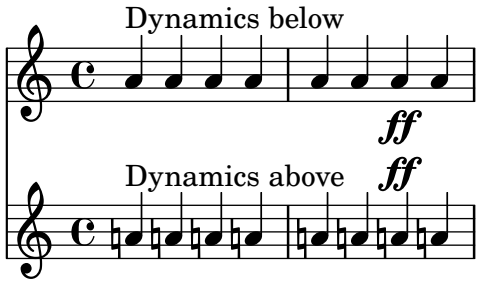
```
\score {
  <<
    \new Staff {
      \relative {
        a'4^"Default font" a a a
        a4 a a a
      }
    }
    \new Staff \with { fontSize = -4 } {
      \relative {
        a'4^"Smaller font" a a a
        a4 a a a
      }
    }
  >>
}
```



- A predefined command such as \dynamicUp.

```
\score {
  <<
    \new Staff {
      \new Voice {
        \relative {
          a'4^"Dynamics below" a a a
          a4 a a\ff a
        }
      }
    }
    \new Staff \with { \accidentalStyle dodecaphonic } {
      \new Voice \with { \dynamicUp } {
        \relative {
          a'4^"Dynamics above" a a a
          a4 a a\ff a
        }
      }
    }
```

```
  >>
}
```



See also

Notation Reference: Chapter 19 [Input modes], page 589.

33.5.3 Order of precedence

The value of a property which applies at a particular time is determined as follows:

- if an `\override` or `\set` command in the input stream is in effect that value is used,
- otherwise the default value taken from a `\with` statement on the context initiation statement is used,
- otherwise the default value taken from the most recent appropriate `\context` block in the `\layout` or `\midi` blocks is used,
- otherwise LilyPond's built-in default is used.

See also

Learning Manual: Section "Modifying context properties" in *Learning Manual*.

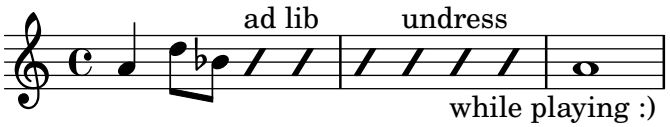
Notation Reference: Section 33.1 [Contexts explained], page 736, Section 33.1.5 [Bottom-level contexts – voices], page 738, Section 35.2 [`\set` and `\unset`], page 760, Section 35.3 [`\override` and `\revert`], page 762, Section 27.1 [The `\layout` block], page 683.

33.6 Defining new contexts

Context definition overview

Specific contexts, like `Staff` and `Voice`, are made from 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` context, we want commands that work in (existing) `Voice` contexts to continue working. This is achieved by giving the new context an alias of `Voice`,

```
\alias Voice
```

The context will print notes and instructive texts, so we need to add the engravers that provide this functionality, plus the engraver that groups notes, stems, and rests occurring at the same musical moment into columns.

```
\consists Note_heads_engraver
\consists Text_engraver
\consists Rhythmic_column_engraver
```

The note heads should all be placed on the center line.

```
\consists Pitch_squash_engraver
squashedPosition = 0
```

The `Pitch_squash_engraver` modifies note heads (created by the `Note_heads_engraver`) and sets their vertical position to the value of `squashedPosition`, in this case `0`, the center line.

The notes should look like a slash without a stem.

```
\override NoteHead.style = #'slash
\hide Stem
```

All these plug-ins have to communicate under the control of the context. The mechanisms with which contexts communicate are established by declaring the context `\type`. Within a `\layout` block, most contexts are of type `Engraver_group`. Some special contexts use other context types. Copying and modifying an existing context definition will also fill in the type. Since this example creates a definition from scratch, it needs to be specified explicitly.

```
\type Engraver_group
```

Put together, we get

```
\context {
  \name ImproVoice
  \type Engraver_group
  \consists Note_heads_engraver
  \consists Text_engraver
  \consists Rhythmic_column_engraver
  \consists Pitch_squash_engraver
  squashedPosition = 0
  \override NoteHead.style = #'slash
  \hide Stem
  \alias Voice
}
```

Contexts form hierarchies. We want to place the `ImproVoice` context within the `Staff` context, just like normal `Voice` contexts. Therefore, we modify the `Staff` definition with the `\accepts` command.

```
\context {
  \Staff
  \accepts ImproVoice
}
```

Often when reusing an existing context definition, the resulting context can be used anywhere where the original context would have been useful. Doing

```
\layout {
  ...
  \inherit-acceptability to from
}
```

arranges to have contexts of type *to* accepted by all contexts also accepting *from*. For example, using

```
\layout {
  ...
  \inherit-acceptability ImproVoice Voice
}
```

adds an `\accepts` for `ImproVoice` to both `Staff` and `RhythmicStaff` definitions.

The opposite of `\accepts` is `\denies`, which is sometimes needed when reusing existing context definitions.

Arranging the required pieces into a `\layout` block leaves us with

```
\layout {
  \context {
    \name ImproVoice
    ...
  }
  \inherit-acceptability ImproVoice Voice
}
```

Then the output at the start of this subsection can be entered as

```
\relative {
  a'4 d8 bes8
  \new ImproVoice {
    c4^"ad lib" c
    c4 c^"undress"
    c c_"while playing :)"
  }
  a1
}
```

See also

Internals Reference: Section “Contexts” in *Internals Reference*, Section “Engravers and Performers” in *Internals Reference*.

New contexts in MIDI

In MIDI output, the syntax for defining new context types is the same, except that the `\context` block should be placed inside a `\midi` block, and the `\type` should normally be `Performer_group` rather than `Engraver_group`. The term *engraver* refers to a context plug-in that creates visual output. A *performer*, on the other hand, is relevant in MIDI output only. When plug-ins have “translator” in their name rather than “engraver” or “performer”, they are relevant for both graphical and audio output. Thus, when adapting a context definition for the `\midi` block, you need to

- copy it in a `\midi` block,
- change `Engraver_group` to `Performer_group`,

- remove \consists for engravers (they are not relevant), and possibly add \consists for performers.

Please note that, in order to maintain consistent interpretation between graphical and MIDI output, it is recommended to copy any custom context definition in a \midi block. It should at the minimum include those commands that specify the context hierarchy, such as \accepts, \defaultchild, and \inherit-acceptability. Copying aliases is advised as well.

Thus, to complete the example above, the following can be added:

```
\midi {
  \context {
    \name ImproVoice
    \type Performer_group
    \alias Voice
    \consists Note_performer
    \consists Beam_performer
    \consists Dynamic_performer
    \consists Tie_performer
    \consists Slur_performer
  }
  \context {
    \Staff
    \accepts ImproVoice
  }
}
```

This makes the ImproVoice context also work in MIDI output.

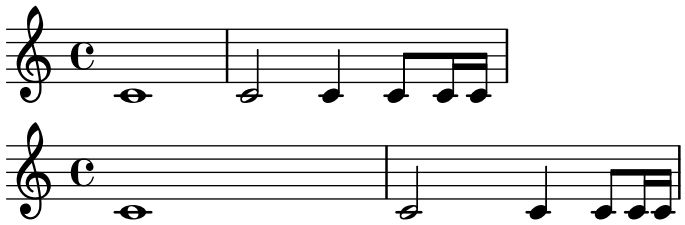
Replacing the Score context

In order to write a context MyScore that acts as the topmost context, as the Score context usually does, use \inherit-acceptability MyScore Score. The following example defines a ProportionalScore context where proportional notation is enabled (see Section 30.6 [Proportional notation], page 722).

```
\layout {
  \context {
    \Score
    \name ProportionalScore
    \alias Score
    proportionalNotationDuration = #1/8
  }
  \inherit-acceptability ProportionalScore Score
}

music = { c'1 2 4 8 16 16 }

\new Score \music
\new ProportionalScore \music
```



Since the topmost context needs to contain a number of fundamental engravers, inheriting settings with \Score is easiest in most cases. If you nevertheless define a score-level context from scratch without inheriting the Score definition, the argument to \type should be Score_engraver (or Score_performer in \midi) rather than Engraver_group. Furthermore, giving the topmost context the Score alias is strongly recommended given that a number of engravers need to access the topmost context using its alias.

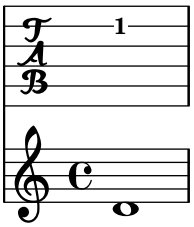
33.7 Context layout order

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 includes 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 are repositioned below the outer context rather than nested within it.

The "accepts" list of a context can be changed with the \accepts or \denies commands. \accepts adds a context to the "accepts" list and \denies removes a context from the list.

For example, a TabStaff by default \accepts TabVoice contexts and \denies Voice contexts. If a Voice context is written within the TabStaff, it would be set on a separate staff.

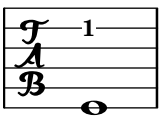
```
\score {
  \new TabStaff <<
    \new TabVoice { c'1 }
    \new Voice { d'1 }
  >>
}
```



However, by using the \accepts command, Voice can be forced onto the TabStaff context.

```
\score {
  \new TabStaff <<
    \new TabVoice { c'1 }
    \new Voice { d'1 }
  >>

\layout {
    \context {
      \TabStaff
      \accepts Voice
    }
  }
}
```



`\denies` is mainly used when a new context is being based on another, but the required nesting differs. For example, the `VaticanaStaff` context is based on the `Staff` context, but with the `VaticanaVoice` context substituted for the `Voice` context in the "accepts" list.

Note that a context is silently created implicitly if a command is encountered when there is no suitable context available to contain it.

Within a context definition, the type of subcontext to be implicitly created is specified using `\defaultchild`. A number of music events require a bottom-level context: when such an event is encountered, subcontexts are created recursively until reaching a context with no `\defaultchild` setting.

Implicit context creation can at times give rise to unexpected new staves or scores. Using `\new` to create contexts explicitly avoids those problems.

Sometimes a context is required to exist for just a brief period, a good example being the staff context for an ossia. This is usually achieved by introducing the context definition at the appropriate place in parallel with corresponding section of the main music. By default, the temporary context is placed below all the existing contexts. To reposition it above the context called "main", it should be defined like this:

```
\new Staff \with { alignAboveContext = "main" }
```

A similar situation arises when positioning a temporary lyrics context within a multi-staff layout such as a `ChoirStaff`, for example, when adding a second verse to a repeated section. By default the temporary lyrics context is placed beneath the lower staves. By defining the temporary lyrics context with `alignBelowContext` it can be positioned correctly beneath the (named) lyrics context containing the first verse.

Examples showing this repositioning of temporary contexts can be found elsewhere – see Section "Nesting music expressions" in *Learning Manual*, Section 6.2 [Modifying single staves], page 250, and Section 9.2 [Techniques specific to lyrics], page 358.

See also

Learning Manual: Section "Nesting music expressions" in *Learning Manual*.

Notation Reference: Section 6.2 [Modifying single staves], page 250, Section 9.2 [Techniques specific to lyrics], page 358.

Application Usage: Section "An extra staff appears" in *Application Usage*.

Installed Files: `ly/engraver-init.ly`.

34 Explaining the Internals Reference
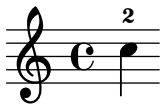
See also

Notation Reference: Section B.24 [Naming conventions], page 973.

34.1 Navigating the program reference

Suppose we want to move the fingering indication in the fragment below:

```
c''-2
```



If you visit the documentation on fingering instructions (in Section 7.1.2 [Fingering instructions], page 282), 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 the following engraver(s): 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 the following engraver(s): 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.

34.2 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 *interface*s, 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',
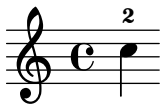
```
(Fingering
  . ((padding . 0.5)
     (avoid-slur . around)
     (slur-padding . 0.2)
     (staff-padding . 0.5)
     (self-alignment-X . 0)
     (self-alignment-Y . 0)
     (script-priority . 100)
     (stencil . ,ly:text-interface::print)
     (direction . ,ly:script-interface::calc-direction)
     (font-encoding . fetaText)
     (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 web page in the Internals Reference is directly generated from this definition.

34.3 Determining the grob property

Recall that we wanted to change the position of the **2** in

```
c''-2
```



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
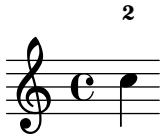
> `padding` (dimension, in staff space)
>
> Add this much extra space between objects that are next to each other.

By increasing the value of `padding`, we can move the fingering away from the note head. The following command will insert "three staff spaces" worth of distance between the note and a fingering mark:

```
\once \override Voice.Fingering.padding = 3
```

Inserting the padding before the fingering object is created results in the following:

```
\once \override Voice.Fingering.padding = 3
c''-2
```



In this case, the context for this tweak is `Voice`. See Section "Fingering_engraver" in *Internals Reference* plug-in, which says:

> Fingering_engraver is part of contexts: . . . Section "Voice" in *Internals Reference*

35 Modifying properties

35.1 Overview of modifying properties

Within each context, there are two different kinds of properties: *context properties* and *grob properties*. Context properties apply to a context as a whole, whereas grob properties are used for initializing grobs (graphical objects) engraved from within a context.

Context properties control the translation from music to notation. For example, `localAlterations` is used to determine whether to print accidentals; or `currentBarNumber` for determining which bar number to print. They can also change value over time while interpreting a piece of music; `currentBarNumber` is an obvious example of this.

The `\set` command (and its counterpart `\unset`) is used to alter values for context properties, whereas the `\override` command (and its counterpart `\revert`) is used to change values for grob properties.

35.2 \set and \unset

The `\set` command syntax is

```
\set context.property = value
```

where *value* must be preceded by the '`#`' character if it is a Scheme object.
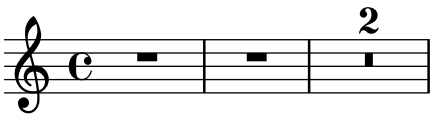
The counterpart command `\unset`'s syntax is

```
\unset context.property
```

This removes a previously set definition of the *property* from the *context*. Any properties that have been set in an enclosing *context* are not altered by an `\unset` in the same enclosed context.
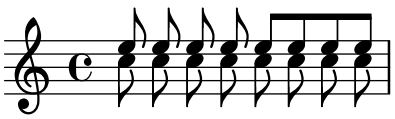
For example, multi-measure rests are combined into a single bar (as explained in Section 6.3.4 [Compressing empty measures], page 274) if the context property `skipBars` is set to `#t`:

```
R1*2
\set Score.skipBars = ##t
R1*2
```
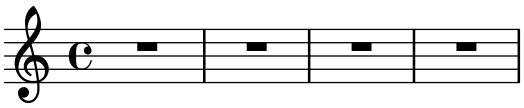


If the *context* argument is left out, then the property is set in the current 'bottom' context (typically `ChordNames`, `Voice`, `TabVoice`, or `Lyrics`).

```
<<
  \set Score.autoBeaming = ##f
  \relative {
    e''8 e e e
    \set autoBeaming = ##t
    e8 e e e
  } \\
  \relative {
    c''8 c c c c8 c c c
  }
>>
```



Note that the bottom context may not always contain an *engraver* that uses the *property* that you wish to change. For example, attempting to set the `skipBars` property of the default, bottom context has no effect because it is a property of the `Score`, not `Voice` context.

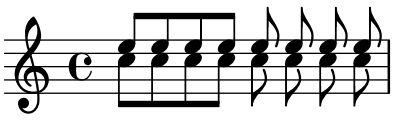
```
R1*2
\set skipBars = ##t
R1*2
```



Contexts are hierarchical; any change specified for an enclosing context (e.g., `Staff`) would also apply to all `Voices` in that current `Staff` context (assuming, of course, that the `Voice` context didn't have an override of its own).

All contexts inherit settings established in the top-most `Global` context (via `\grobdescriptions`), although a few of those defaults get overridden in their own context definitions.

```
<<
  \set Score.autoBeaming = ##t
  \relative {
    \unset autoBeaming
    e''8 e e e
    \unset Score.autoBeaming
    e8 e e e
  } \\
  \relative {
    c''8 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
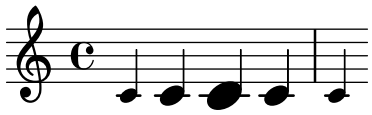
```
\set Voice.autoBeaming = ##t
\set autoBeaming = ##t
```

are equivalent if the current bottom context is `Voice`.

As described above, `\unset` restores the default value of a context property. However, it is sometimes useful to change a value for some time, then return to the previously used value. For this purpose, there are the two commands `\pushContextProperty` and `\popContextProperty`: the first one pushes the current value to a stack, while the second one pops a value from the stack and restores the property to it.

```
{
  c'
  \pushContextProperty Staff.fontSize
  \set Staff.fontSize = 3
  c'
  \pushContextProperty Staff.fontSize
  \set Staff.fontSize = 6
```

```
  c'
  \popContextProperty Staff.fontSize
  c'
  \popContextProperty Staff.fontSize
  c'
}
```



35.3 \override and \revert
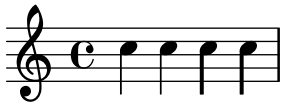
LilyPond does its typesetting by creating 'grobs', graphical objects. The various properties with which grobs are initialized can be set in each context. The names of grobs start with a capital letter; their default properties can be looked up in `scm/define-grobs.scm`.

The syntax for the `\override` command is

```
\override [context.]GrobName.property = value
```
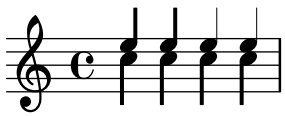
For example, we can increase the thickness of a note stem by overriding the `thickness` property of the `Stem` object:

```
c''4 c''
\override Voice.Stem.thickness = 3.0
c''4 c''
```



If no context is specified in an `\override` command, the bottom context is used:

```
\override Staff.Stem.thickness = 3.0
<<
  \relative {
    e''4 e
    \override Stem.thickness = 0.5
    e4 e
  } \\
  \relative {
    c''4 c c c
  }
>>
```



Some tweakable options are called 'subproperties' and reside inside properties. To tweak those, use commands in the form

```
\override Stem.details.beamed-lengths = #'(4 4 3)
```

or to modify the ends of spanners, use a form like these
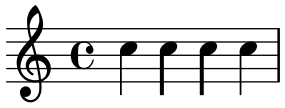
```
\override TextSpanner.bound-details.left.text = "left text"
\override TextSpanner.bound-details.right.text = "right text"
```

The effects of \override can be undone by \revert; its syntax is

\revert [*context*.]*GrobName*.*property*
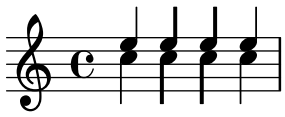
For example,

```
\relative {
  c''4
  \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:
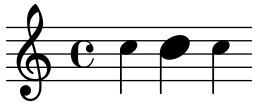
```
<<
  \relative {
    e''4
    \override Staff.Stem.thickness = 3.0
    e4 e e
  } \\
  \relative {
    c''4 c c
    \revert Staff.Stem.thickness
    c4
  }
>>
```



35.4 The \once command

\once is used in conjunction with the \set or \override command to affect only the current musical moment.

```
c''4
\once \set fontSize = 4.7
c''4
c''4
```



See also

Internals Reference: Section “Backend” in *Internals Reference*.

35.5 \temporary and \undo

\temporary modifies an override, like \once does. If you apply an \override, the old value is usually lost and can no longer be restored. \once works differently in that it reestablishes the previous value. If the effect of an override should last for more than a single timestep but the previous behavior should be restored afterwards, you can pair \temporary \override with \revert on the same property, with music requiring the override to be in effect in the middle.

\undo is a command that is most practical in combination with overrides packaged in definitions. If you apply \undo to such a property-changing definition, this will create the kind of command reverting the original command. So you can, for example, bracket a passage without visible Stem grobs as

```
\temporary \omit Stem
...
\undo \omit Stem
```

See Section 36.7.1 [Removing the stencil], page 785, for an explanation of the \omit command.

35.6 \set versus \override

Both \set and \override commands manipulate properties associated with contexts. The properties exist in a hierarchy of contexts where each context contains zero or more others below it. A property that is not set in a specific context shows the value from the nearest enclosing context where it is set.

The lifetime and value of a context property is dynamic and only available when music is being interpreted (i.e., 'iterated'). At the time of a context's creation, properties are initialized from its corresponding definitions (along with any other modification). Any subsequent changes are achieved with 'property-setting' commands within the music itself.

While graphical object (or "grob") properties also follow the hierarchy of contexts, they are changed with \override and \revert and have names starting with capital letters, while plain context properties are changed with \set and \unset and have names starting with lowercase letters.

Grobs are usually created by 'engravers' at the time of interpreting a music expression, receiving their initial properties from the current engraver's context. The engraver (or other 'backend' parts of LilyPond) can then change (or add to) the grob's initial properties. However, this does not affect the context's own grob definitions.

The commands \tweak and \overrideProperty change grob properties by bypassing all context properties completely and, instead, catch grobs as they are being created, setting properties on them for a music event (\tweak) or, in the case of \overrideProperty for a specific override.

35.7 \tweak and \single

When multiple grobs occur at the same musical moment the \override command cannot be used to modify just one of them. In this case the \tweak command is used.

The \tweak command has the following syntax

```
\tweak [layout-object.]grob-property value
```

and applies to the music expression that immediately follows *value* in the music stream. Specifying *layout-object* is necessary for disambiguation if the music expression causes the indirect creation of grobs with different types (for example, NoteHead causes Stem).
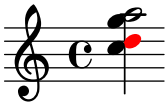
For an introduction to the syntax and uses of the tweak command see Section "Tweaking methods" in *Learning Manual*.

Items that may appear more than once at the same musical moment include, but are not limited to, 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
```
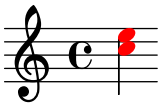


\tweak can also be used to modify slurs:

```
\relative { c'-\tweak thickness 5 ( d e f) }
```



Tweaking a whole chord tweaks all the contained notes:
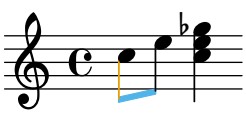
```
{ \tweak color #red <c'' e''>4 }
```



As mentioned above, the simple \tweak command syntax form cannot be used to modify any object that is not directly created from the input. In particular, it will not affect stems, automatic beams, or accidentals, since these are generated later by NoteHead layout objects rather than by music elements in the input stream.

Such indirectly created layout objects can be tweaked using the form of the \tweak command in which the grob name is specified explicitly:
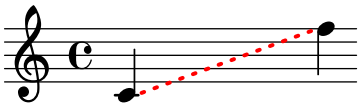
```
\tweak Stem.color #(universal-color 'orange)
\tweak Beam.color #(universal-color 'skyblue) c''8 e''
<c'' e'' \tweak Accidental.font-size -3 ges''>4
```



\tweak cannot 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.

Multiple \tweak commands placed before a music expression all affect the grob(s) created at this musical moment.

```
c'
  -\tweak springs-and-rods #ly:spanner::set-spacing-rods
  -\tweak minimum-length 15
  -\tweak style #'dashed-line
  -\tweak dash-fraction 0.2
  -\tweak thickness 3
  -\tweak color #red
  \glissando
f''
```
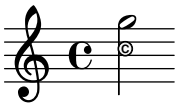


The music stream which is generated from a section of an input file, including any automatically inserted elements, may be examined, see Section "Displaying music expressions" in *Extending*. This may be helpful in determining what may be modified by a `\tweak` command, or in determining how to adjust the input to make a `\tweak` apply.

The `\single` command takes one or more `\override` commands (which are intended to apply at a given musical moment or beyond) and converts them effectively into a *single* 'tweak' that now applies to the specific grobs created.

The file `ly/property-init.ly` contains many definitions of multiple `\override` commands and so can be used in conjunction with the `\single` command. For example, the function `\easyHeadsOn` can be used with `\single` to affect just one note head in a chord;

```
\relative c' {
  <\single \easyHeadsOn c' g'>2
}
```



See also

Learning Manual: Section "Tweaking methods" in *Learning Manual*.

Extending LilyPond: Section "Displaying music expressions" in *Extending*.

Known issues and warnings

The `\tweak` command cannot be used to modify the control points of just one of several ties in a chord, other than the first one encountered in the input file.

35.8 The `\offset` command

While it is possible to set grob properties to new values with the `\override`, `\tweak`, and `\overrideProperty` commands, it is often more convenient to modify such properties relative to a default value. The `\offset` command is available for this purpose.

The syntax for `\offset` is

`[-]\offset` *`property offsets item`*

The command works by adding the contents of *offsets* to the default setting of the property *property* of the grob indicated by *item*.

Depending on the formulation of the command, `\offset` may act as either a `\tweak` or `\override`. The variations in usage are discussed after consideration is given to grob properties that may be used with `\offset`.

Properties which may be offset

Many, but not all, grob properties may be offset. If *property* cannot be offset, the object will remain unchanged and a warning will be issued. In such cases, `\override` or `\tweak` should be used to modify the object instead.

One can work by trial and error and let the warnings be the guide to what may or may not be offset. A more systematic approach is possible, however.

The following criteria determine whether a property can be modified with `\offset`:

- The property has a 'default setting' in the grob's description. Such properties are listed for each grob in Section "All layout objects" in *Internals Reference*. (They are also found in `scm/define-grobs.scm`.)
- The property takes a numerical value. Numerical values include `number`, list of `numbers`, `number-pair`, and `number-pair-list`. The pages at Section "All layout objects" in *Internals Reference* list the type of data characteristic to each property. It is immaterial whether the default setting is a function.
- The property cannot be a 'subproperty' – a property residing within another property.
- Properties set to infinite values cannot be offset. There is no sensible way to offset positive and negative infinity.

The following examples consider several grob properties against the criteria outlined above.

- Properties that may be offset

  `Hairpin.height`
  : This property is not a subproperty, and it is listed at Section "Hairpin" in *Internals Reference*. For a value, it takes 'dimension, in staff space' set to `0.6666` – clearly a non-infinite `number`.

  `Arpeggio.positions`
  : The page Section "Arpeggio" in *Internals Reference* lists a `positions` property which accepts a 'pair of numbers'. It defaults to `ly:arpeggio::positions` – a callback which will be evaluated during the typesetting phase to yield a pair of numbers for any given `Arpeggio` object.

- Properties that may not be offset

  `Hairpin.color`
  : There is no listing for `color` at Section "Hairpin" in *Internals Reference*.

  `Hairpin.circled-tip`
  : The listing for `Hairpin.circled-tip` at Section "Hairpin" in *Internals Reference* shows that it takes a `boolean` value. Booleans are non-numerical.

  `Stem.details.lengths`
  : Though listed at Section "Stem" in *Internals Reference* and defaulting to a list of `numbers`, this is a 'subproperty'. There is currently no support for 'nested properties'.

`\offset` as an override

If *item* is a grob name like `Arpeggio` or `Staff.OttavaBracket`, the result is an `\override` of the specified grob type.

```
\offset property offsets [context.]GrobName
```

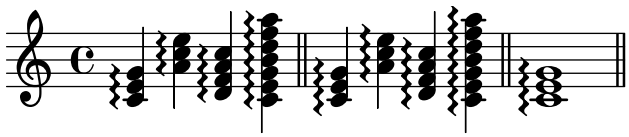
Note that the leading hyphen is *never* used with the 'override' form, just as it is never used with the `\override` command itself.

The following example uses the 'override' form to lengthen the default arpeggios shown in the first measure to cover the extent of the chords more fully. The arpeggios are stretched by a half

staff space to top and bottom. Also shown is the same operation done on the first chord with an ordinary override of the `positions` property. This method is not at all expressive of the task of 'stretching by a half staff space', as the endpoints must be specified with absolute rather than relative coordinates. Furthermore, individual overrides would be needed for the other chords, as they vary in size and position.

```
arpeggioMusic = {
  <c' e' g'>\arpeggio <a' c'' e''>\arpeggio
  <d' f' a' c''>\arpeggio <c' e' g' b' d'' f'' a''>\arpeggio
}

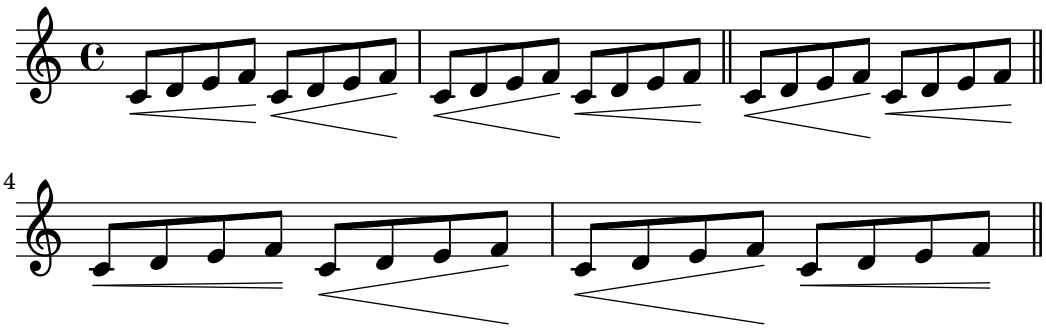
{
  \arpeggioMusic
  \bar "||"
  \offset positions #'(-0.5 . 0.5) Arpeggio
  \arpeggioMusic
  \bar "||"
  \once \override Arpeggio.positions = #'(-3.5 . -0.5)
  <c' e' g'>1\arpeggio
  \bar "||"
}
```



In its 'override' usage, `\offset` may be prefaced with `\once` or `\temporary` and reverted using `\revert` with *property* (see Section "Intermediate substitution functions" in *Extending*). This follows from the fact that `\offset` actually creates an `\override` of *property*.

```
music = { c'8\< d' e' f'\! }

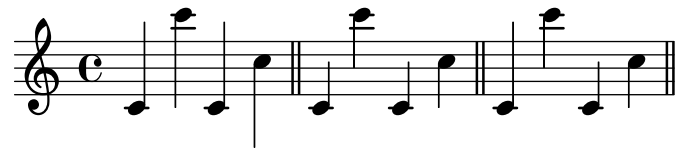
{
  \music
  \offset height 1 Hairpin
  \music
  \music
  \revert Hairpin.height
  \music
  \bar "||"
  \once \offset height 1 Hairpin
  \music \music
  \bar "||"
  \override Hairpin.height = 0.2
  \music
  \temporary \offset height 2 Hairpin
  \music
  \music
  \revert Hairpin.height
  \music
  \bar "||"
}
```



Also like \override, the 'override' form of \offset may be used with \undo and \single.

```
longStem = \offset length 6 Stem

{
 \longStem c'4 c''' c' c''
 \bar "||"
 \undo \longStem c'4 c''' c' c''
 \bar "||"
 \single \longStem c'4 c''' c' c''
 \bar "||"
}
```



\offset as a tweak
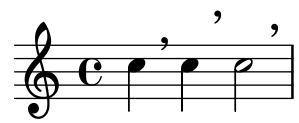
If *item* is a music expression such as (or \arpeggio, the result is the same music expression with a tweak applied.

[-]\offset [*GrobName*.]*property offsets music-expression*

The syntax of \offset in its 'tweak' form is analogous to the \tweak command itself, both in ordering and in the presence or absence of the leading hyphen.

The following example uses the 'tweak' form to adjust the vertical position of the BreathingSign object. Compare this with the ordinary \tweak command also demonstrated. The syntax is equivalent; however, the output of \tweak is less intuitive, since BreathingSign.Y-offset is calculated from the middle staff line. It is not necessary to know how Y-offset is calculated when using \offset.

```
{
  c''4
  \breathe
  c''4
  \offset Y-offset 2 \breathe
  c''2
  \tweak Y-offset 3 \breathe
}
```
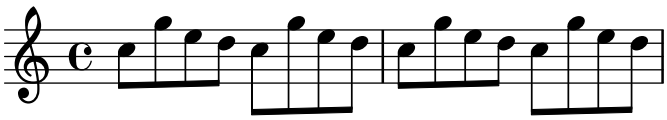


In the previous example, the tweaked objects were created directly from the user input: the `\breathe` command was an explicit instruction to return a `BreathingSign` object. Since the focus of the command was unambiguous, there was no need to specify the object's name. When an object is *indirectly* created, however, it is necessary to include the grob's name. This is the same as for the `\tweak` command.

In the following example, the `Beam` object is lowered two staff spaces by applying `\offset` to the `positions` property.

The first application of `\offset` requires that the grob's name be included, because nothing in the input explicitly creates the beam. In the second application, the beam is created manually with the music expression `[`; therefore, the grob's name is not needed. (Also illustrated is a shorthand: a single `number` will be applied to both members of a `number-pair`.)

```
{
  c''8 g'' e'' d''
  \offset Beam.positions #'(-2 . -2)
  c''8 g'' e'' d''
  c''8 g'' e'' d''
  c''8-\offset positions -2 [ g'' e'' d'']
}
```



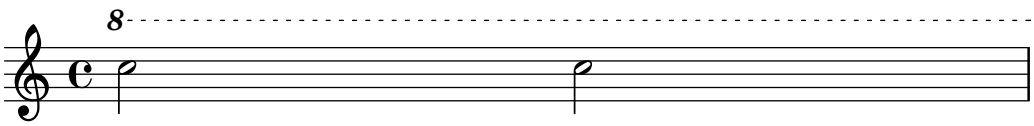
`\offset` with broken spanners

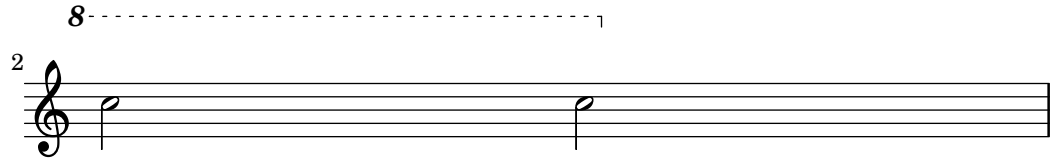
Independently modifying segments of a spanner extending over a line break or breaks is also possible. In this case, *offsets* takes a list of values of the property's required data type.

The `\offset` command used in this manner is similar to the `\alterBroken` command. (See Section 36.4.1 [Modifying broken spanners], page 776.) In contrast with `\alterBroken`, however, the values given to `\offset` are relative, not absolute.

The following example displaces the 'broken' `OttavaBracket` object through its `staff-padding` property. Since the property takes a `number`, *offsets* is provided with a list of `numbers` to account for the two segments created by the line break. The bracket piece on the first line is effectively untouched since `0` is added to its default value of `staff-padding`. The segment on the second line is raised three staff spaces from its default height. The default height happens to be `2`, though it is not necessary to know this to achieve the desired positioning.

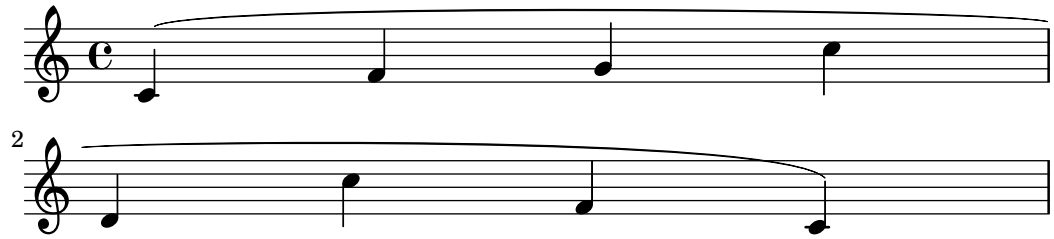
```
{
  \offset staff-padding #'(0 3) Staff.OttavaBracket
  \ottava 1
  c'''2 c'''
  \break
  c'''2 c'''
}
```





The following example mimics the effect of the \shape command by offsetting the control-points property of the Slur object. Here, *offsets* is a list of number-pair-lists, one for each slur segment. This example achieves a result identical to the corresponding illustration at Section 36.11 [Modifying shapes], page 801.

```
{
  c'4-\offset control-points #'(
              ((0 . 0) (0 . 0) (0 . 0) (0 . 1))
              ((0.5 . 1.5) (1 . 0) (0 . 0) (0 . -1.5))
             ) ( f'4 g' c''
  \break
  d'4 c'' f' c')
}
```



See Section "Adjusting slur positions vertically" in *Snippets* for an \offsetPositions Scheme function that eases the vertical shifting of slurs.

35.9 Modifying alists

Some user-configurable properties are internally represented as *alists* (association lists), which store pairs of *keys* and *values*. The structure of an alist is:

```
'((key1 . value1)
  (key2 . value2)
  (key3 . value3)
  ...)
```

If an alist is a grob property or \paper variable, its keys can be modified individually without affecting other keys.

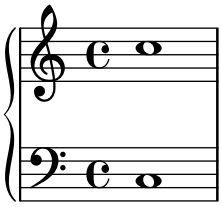
For example, to reduce the space between adjacent staves in a staff group, use the staff-staff-spacing property of the StaffGrouper grob. The property is an alist with four keys: basic-distance, minimum-distance, padding, and stretchability. The standard settings for this property are listed in the "Backend" section of the Internals Reference (see Section "StaffGrouper" in *Internals Reference*):
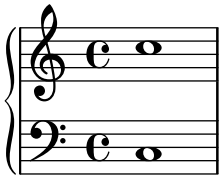
```
'((basic-distance . 9)
  (minimum-distance . 7)
  (padding . 1)
  (stretchability . 5))
```

One way to bring the staves closer together is by reducing the value of the basic-distance key (9) to match the value of minimum-distance (7). To modify a single key individually, use a *nested declaration*:

```
% default space between staves
\new PianoStaff <<
  \new Staff { \clef treble c''1 }
  \new Staff { \clef bass   c1   }
>>

% reduced space between staves
\new PianoStaff \with {
  % this is the nested declaration
  \override StaffGrouper.staff-staff-spacing.basic-distance = 7
} <<
  \new Staff { \clef treble c''1 }
  \new Staff { \clef bass   c1   }
>>
```

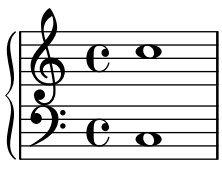




Using a nested declaration will update the specified key (such as `basic-distance` in the above example) without altering any other keys already set for the same property.

Now suppose we want the staves to be as close as possible without overlapping. The simplest way to do this is to set all four alist keys to zero. However, it is not necessary to enter four nested declarations, one for each key. Instead, the property can be completely redefined with one declaration, as an alist:

```
\new PianoStaff \with {
  \override StaffGrouper.staff-staff-spacing =
    #'((basic-distance . 0)
       (minimum-distance . 0)
       (padding . 0)
       (stretchability . 0))
} <<
  \new Staff { \clef treble c''1 }
  \new Staff { \clef bass   c1   }
>>
```



Note that any keys not explicitly listed in the alist definition will be reset to their *default-when-unset* values. In the case of `staff-staff-spacing`, any unset key values would be reset

to zero (except `stretchability`, which takes the value of `basic-distance` when unset). Thus the following two declarations are equivalent:

```
\override StaffGrouper.staff-staff-spacing =
  #'((basic-distance . 7))

\override StaffGrouper.staff-staff-spacing =
  #'((basic-distance . 7)
     (minimum-distance . 0)
     (padding . 0)
     (stretchability . 7))
```

One (possibly unintended) consequence of this is the removal of any standard settings that are set in an initialization file and loaded each time an input file is compiled. In the above example, the standard settings for `padding` and `minimum-distance` (defined in `scm/define-grobs.scm`) are reset to their default-when-unset values (zero for both keys). Defining a property or variable as an alist (of any size) will always reset all unset key values to their default-when-unset values. Unless this is the intended result, it is safer to update key values individually with a nested declaration.

> **Note:** Nested declarations will not work for context property alists (such as `beamExceptions`, `keyAlterations`, `timeSignatureSettings`, etc.). These properties can only be modified by completely redefining them as alists.

36 Useful concepts and properties

This section discusses various common layout issues and the tweaking methods related to them.

See also

Learning Manual: Section “Tweaking output” in *Learning Manual*, Section “Other sources of information” in *Learning Manual*.

Notation Reference: Chapter 34 [Explaining the Internals Reference], page 757, Chapter 35 [Modifying properties], page 760.

Extending LilyPond: Section “Interfaces for programmers” in *Extending*.

Installed Files: `scm/define-grobs.scm`.

Snippets: Section “Tweaks and overrides” in *Snippets*.

Internals Reference: Section “All layout objects” in *Internals Reference*.

36.1 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.

36.1.1 Articulation direction indicators
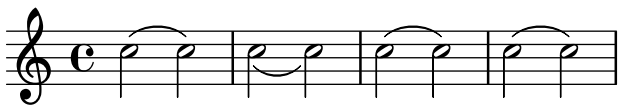
By default some directions are always up or always down (e.g., dynamics or fermata), while other things can alternate between up or down based on the stem direction (like slurs or accents).

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:

```
\relative {
  c''2( c)
  c2_( c)
  c2( c)
  c2^( c)
}
```



36.1.2 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 of the form

`\`*xxx*`Up, \`*xxx*`Down or \`*xxx*`Neutral`

where `\`*xxx*`Neutral` means "use the default" direction. See Section "Within-staff objects" in *Learning Manual*.

In a few cases, arpeggio for example, the value of the `direction` property can specify whether the object is to be placed to the right or left of the parent. 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.

These indications affect all notes until they are canceled.

```
\relative {
  c''2( c)
  \slurDown
  c2( c)
  c2( c)
  \slurNeutral
  c2( c)
}
```



In polyphonic music, it is generally better to specify an explicit `voice` than change an object's direction. For more information, see Section 5.2 [Multiple voices], page 222.

See also

Learning Manual: Section "Within-staff objects" in *Learning Manual*.

Notation Reference: Section 5.2 [Multiple voices], page 222.

36.2 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. In general, distance units may be specified by appending `\mm` (millimeter), `\cm` (centimeter), `\in` (inches), `\pt` (points), or `\bp` (big points).

	mm	**cm**	**in**	**pt**	**bp**
mm	1	0.1	0.0394	2.8453	2.8346
cm	10	1	0.3937	28.4528	28.3465
in	25.4	2.54	1	72.27	72
pt	0.3515	0.0351	0.0138	1	0.996
bp	0.3538	0.0354	0.0139	1.0038	1

Page layout distances can also be specified in scalable units (see the following paragraph) by appending `\staff-space` to the quantity. See Chapter 26 [Page layout], page 671, for a detailed description of LilyPond's page layout.

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 27.2 [Setting the staff size], page 685.

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: Chapter 26 [Page layout], page 671, Section 27.2 [Setting the staff size], page 685.

36.3 Dimensions

The dimensions of a graphical object specify the positions of the left and right edges and the bottom and top edges of the objects' bounding box as distances from the objects' reference point in units of staff spaces. These positions are usually coded as two Scheme pairs. For example, the text markup command `\with-dimensions` takes three arguments, the first two of which are a Scheme pair giving the left and right edge positions and a Scheme pair giving the bottom and top edge positions:

```
\with-dimensions #'(-5 . 10) #'(-3 . 15) arg
```

This specifies a bounding box for *arg* with its left edge at -5, its right edge at 10, its bottom edge at -3 and its top edge at 15, all measured from the objects' reference point in units of staff spaces.

For more information on how such boxes are defined, including the formal definition of horizontal and vertical space as set up by the `\hspace` and `\vspace` markup commands, see Section "LilyPond's box model" in *Extending*.

See also

Notation Reference: Section A.1.8 [Other markup commands], page 874, (documentation for commands similar to `\with-dimensions`, such as `\with-dimension` or `\with-dimension-from`), Section 36.2 [Distances and measurements], page 775.

36.4 Spanners
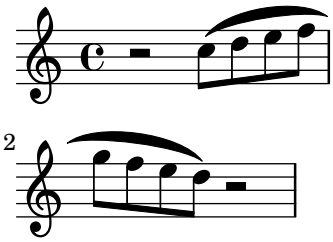
Many objects of musical notation extend over several notes or even several bars. Examples are slurs, beams, tuplet brackets, volta repeat brackets, crescendi, trills, and glissandi. Such objects are collectively called "spanners", and have special properties to control their appearance and behavior, as well as special tweaking methods related to the fact that they can be broken across systems.

36.4.1 Modifying broken spanners

When a spanner crosses a line break or breaks, each piece inherits the attributes of the original spanner. Thus, ordinary tweaking of a broken spanner applies the same modifications to each

of its segments. In the example below, overriding `thickness` affects the slur on either side of the line break.

```
\relative c'' {
  r2
  \once\override Slur.thickness = 10
  c8( d e f
  \break
  g8 f e d) r2
}
```



Independently modifying the appearance of individual pieces of a broken spanner is possible with the `\alterBroken` command. This command can produce either an `\override` or a `\tweak` of a spanner property.

The syntax for `\alterBroken` is

```
[-]\alterBroken property values target
```
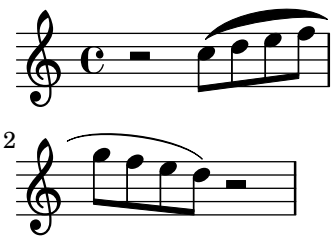
The argument *values* is a list of values, one for each broken piece. If *target* is a grob name like `Slur` or `Staff.PianoPedalBracket`, the result is an `\override` of the specified grob type. If *target* is a music expression such as '`(`' or '`[`' the result is the same music expression with an appropriate tweak applied.

The leading hyphen must be used with the `\tweak` form. Do not add it when `\alterBroken` is used as an `\override`.

In its `\override` usage, `\alterBroken` may be prefaced by `\once` or `\temporary` and reverted by using `\revert` with *property* (see Section "Intermediate substitution functions" in *Extending*).

The following code applies an independent `\override` to each of the slur segments in the previous example:

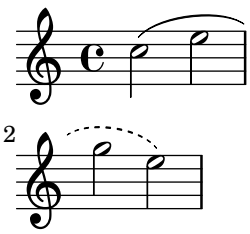
```
\relative c'' {
  r2
  \alterBroken thickness #'(10 1) Slur
  c8( d e f
  \break
  g8 f e d) r2
}
```



The `\alterBroken` command may be used with any spanner object, including `Tie`, `PhrasingSlur`, `Beam` and `TextSpanner`. For example, an editor preparing a scholarly edition

may wish to indicate the absence of part of a phrasing slur in a source by dashing only the segment which has been added. The following example illustrates how this can be done, in this case using the \tweak form of the command:

```
% The empty list is conveniently used below, because it is the
% default setting of dash-definition, resulting in a solid curve.
\relative {
  c''2-\alterBroken dash-definition #'(() ((0 1.0 0.4 0.75))) \(e
  \break
  g2 e\)
}
```



It is important to understand that \alterBroken sets each piece of a broken spanner to the corresponding value in *values*. When there are fewer values than pieces, any additional piece will be assigned the empty list. This may lead to undesired results if the layout property is not set to the empty list by default. In such cases, each segment should be assigned an appropriate value.

Known issues and warnings

Line breaks may occur in different places following changes in layout. Settings chosen for \alterBroken may be unsuitable for a spanner that is no longer broken or is split into more segments than before. Explicit use of \break can guard against this situation.

The \alterBroken command is ineffective for spanner properties accessed before line breaking such as direction.

See also

Extending LilyPond: Section “Difficult tweaks” in *Extending*.

36.4.2 Setting minimum lengths for spanners

The spanner-interface interface provides three properties that apply to several spanners.

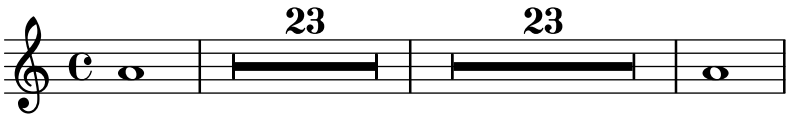
The minimum-length property

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.
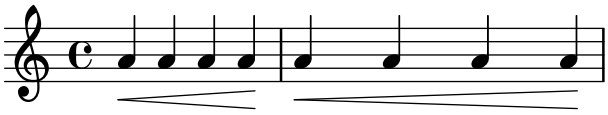
```
a'~ a'
a'
% increase the length of the tie
-\tweak minimum-length 5
~ a'
```



```
\relative \compressMMRests {
  a'1
  R1*23
  % increase the length of the rest bar
  \once \override MultiMeasureRest.minimum-length = 20
  R1*23
  a1
}
```




```
\relative {
  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:

```
\relative {
  a'( g)
  a
  -\tweak minimum-length 5
  ( g)

  a\( g\)
  a
  -\tweak minimum-length 5
  \( g\)
}
```
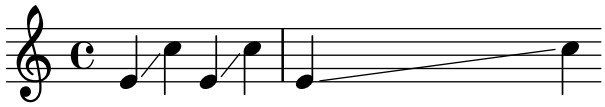


For some layout objects, the `minimum-length` property becomes effective only if the `set-spacing-rods` procedure is called explicitly. To do this, the `springs-and-rods` property should be set to `ly:spanner::set-spacing-rods`. For example, the minimum length of a glissando has no effect unless the `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
```
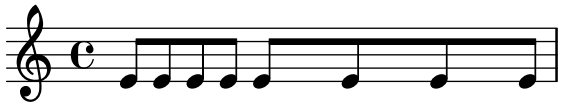
```
\once \override Glissando.minimum-length = 20
\once \override Glissando.springs-and-rods =
                  #ly:spanner::set-spacing-rods
e' \glissando c''
```



The same is true of the Beam object:

```
% not effective alone
\once \override Beam.minimum-length = 20
e'8 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
e'8 e' e' e'
```



The minimum-length-after-break property

The property minimum-length-after-break can be used to stretch broken spanners starting after a line break. As for the minimum-length property, it is often needed to set the springs-and-rods property to ly:spanner::set-spacing-rods.

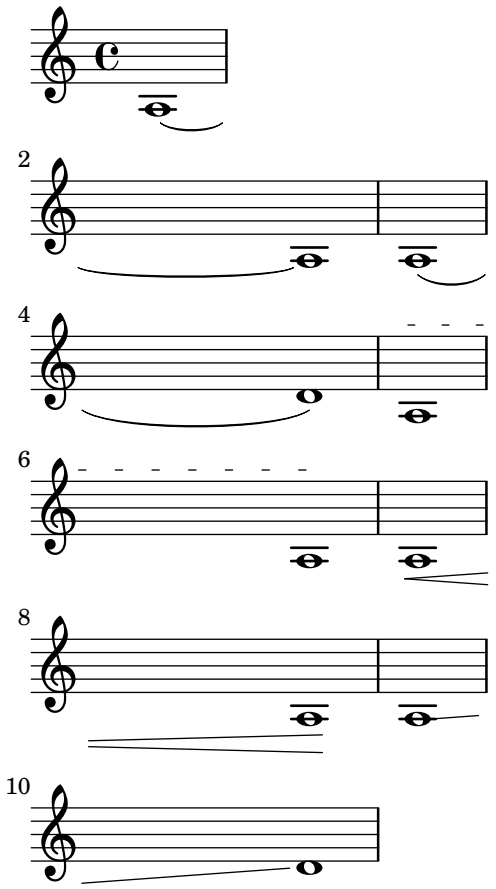
```
{
  \once \override Tie.minimum-length-after-break = 20
  a1~
  \break
  a1

  \once \override Slur.minimum-length-after-break = 20
  a1(
  \break
  d'1)

  \once \override TextSpanner.springs-and-rods =
    #ly:spanner::set-spacing-rods
  \once \override TextSpanner.minimum-length-after-break = 20
  a1\startTextSpan
  \break
  a1\stopTextSpan

  \once \override Hairpin.after-line-breaking = ##t
  \once \override Hairpin.to-barline = ##f
  \once \override Hairpin.minimum-length-after-break = 20
  a1\<
  \break
  a1\!
```
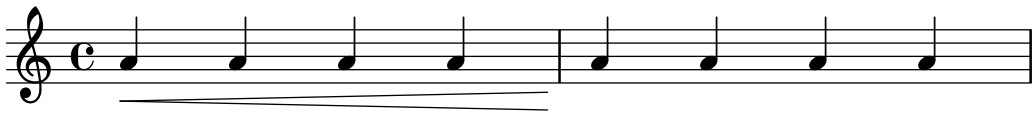
```
  \once \override Glissando.springs-and-rods =
    #ly:spanner::set-spacing-rods
  % for completeness; not necessary for manual breaks
  \once \override Glissando.breakable = ##t
  \once \override Glissando.after-line-breaking = ##t
  \once \override Glissando.minimum-length-after-break = 20
  a1\glissando
  \break
  d'1
}
```

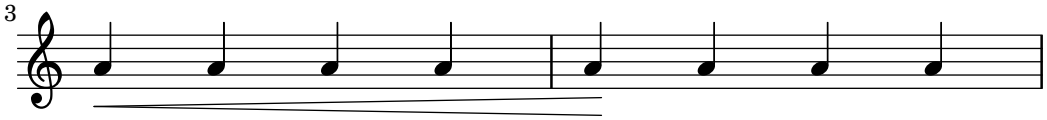


36.4.3 Controlling spanner end points

The `to-barline` property of the `spanner-interface`, usually defaulting to `#t`, causes hairpins and other spanners that are terminated on the first note of a measure to end instead on the immediately preceding bar line. If set to `#f`, the spanner extends beyond the bar line and end on the note itself:

```
\relative {
  a' \< a a a a \! a a a \break
  \override Hairpin.to-barline = ##f
  a \< a a a a \! a a a
}
```



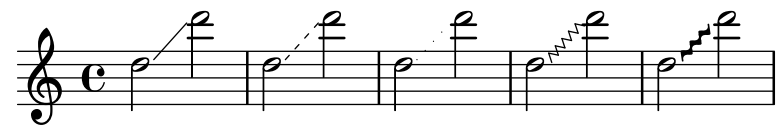


This property is not effective for all spanners. For example, setting it to `#t` has no effect on slurs or phrasing slurs or on other spanners for which terminating on the bar line would not be meaningful.

36.5 Line styles

The Section "line-interface" in *Internals Reference* groups all objects printing lines. All objects supporting these interfaces can be printed using different line styles. Here is an example showing the available values.

```
\relative {
  d''2 \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
}
```
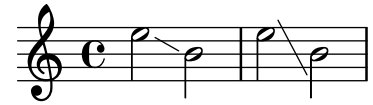


Some objects may support specific additional styles.

36.6 Line spanners

Some performance indications, e.g., *rallentando* and *accelerando* and trills are written as text and are extended over many measures with lines, sometimes dotted or wavy.

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 that need to be specified are nested two levels down within the property hierarchy, but the syntax of the `\override` command is quite simple:

```
e''2 \glissando b'
\once \override Glissando.bound-details.left.Y = 3
\once \override Glissando.bound-details.right.Y = -2
e''2 \glissando b'
```

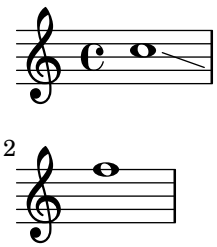


The units for the `Y` property are *staff spaces*, with the center line of the staff being the zero point. For the glissando, this is the value for `Y` at the X-coordinate corresponding to the center point of each note head, if the line is imagined to be extended to there.

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` sublists of `bound-details`. For example:

```
\override Glissando.bound-details.right-broken.Y = -3
c''1 \glissando \break
f''1
```



A number of further properties of the `left` and `right` sublists of the `bound-details` property may be modified in the same way as `Y`:

`Y` This sets the Y-coordinate of the end point, measured in staff spaces 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 hard-coded to 0.
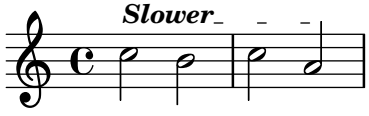
`attach-dir`
This determines where the line starts and ends in the X-direction, relative to the bound object. So, a value of `-1` (or `LEFT`) makes the line start/end at the left side of the note head it is attached to.

`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 subproperty. 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 }
\relative { c''2\startTextSpan b c a\stopTextSpan }
```
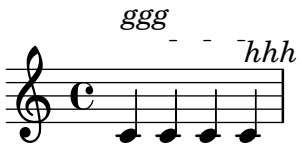


`stencil-align-dir-y`
`stencil-offset`
Without setting one of these, the stencil is simply put at the end point, centered on the line, as defined by the `X` and `Y` subproperties. Setting either `stencil-align-dir-y` or `stencil-offset` will move the symbol at the edge vertically relative to the end 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"

\relative { c'4^\startTextSpan c c c \stopTextSpan }
```
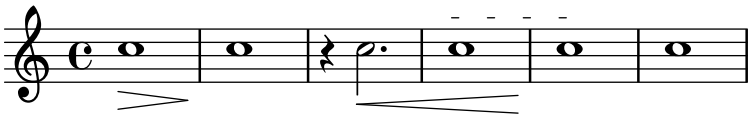


Note that negative values move the text *up*, contrary to the effect that might be expected, as a value of `-1` or `DOWN` means align the *bottom* edge of the text with the spanner line. A value of `1` or `UP` aligns the top edge of the text with the spanner line.

`arrow` Setting this subproperty to `#t` produces an arrowhead at the end points of the line.

`padding` This subproperty 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` prematurely terminates all spanners in its argument, obeying the `to-barline` property if set.

```
\relative c'' {
  \endSpanners c1 \> c
  \endSpanners { r4 c2.\< c1\startTextSpan } c1 c
}
```



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*.

36.7 Visibility of objects

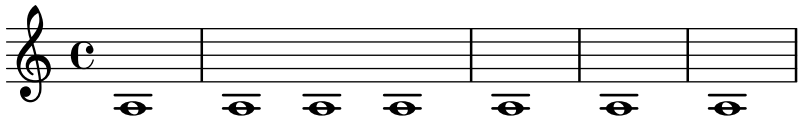
There are four main ways in which the visibility of layout objects can be controlled: their stencil can be removed, they can be made transparent, they can be colored white, or their `break-visibility` property can be overridden. The first three apply to all layout objects; the last to just a few – the *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.

36.7.1 Removing the stencil

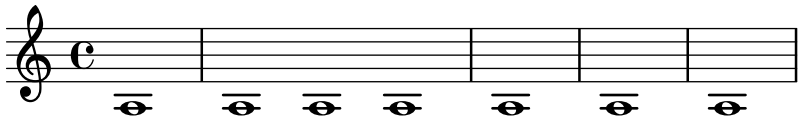
Every layout object has a stencil property. By default this is set to the specific function which draws that object. If this property is overridden to `#f` no function will be called and the object will not be drawn. The default action can be recovered with `\revert`.

```
a1 a
\override Score.BarLine.stencil = ##f
a a
\revert Score.BarLine.stencil
a a a
```



This rather common operation has a shortcut `\omit`:
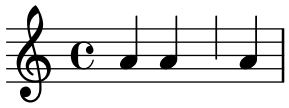
```
a1 a
\omit Score.BarLine
a a
\undo \omit Score.BarLine
a a a
```



36.7.2 Making objects transparent
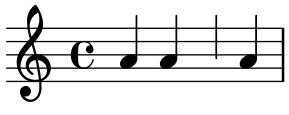
Every layout object has a transparent property which by default is set to `#f`. If set to `#t` the object still occupies space but is made invisible.

```
a'4 a'
\once \override NoteHead.transparent = ##t
a' a'
```



This rather common operation has a shortcut `\hide`:

```
a'4 a'
\once \hide NoteHead
a' a'
```



36.7.3 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
a'1
```



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
a'1
```
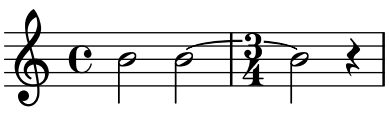


Selected snippets

Using the whiteout property

Any graphical object can be printed over a white background to mask parts of objects that lie beneath. This can be useful to improve the appearance of collisions in complex situations when repositioning objects is impractical. It is necessary to explicitly set the `layer` property to control which objects are masked by the white background.

In this example the collision of the tie with the time signature is improved by masking out the part of the tie that crosses the time signature, setting the `whiteout` property of `TimeSignature`. To do this, `TimeSignature` is moved to a layer above `Tie`, which is left in the default layer 1, and `StaffSymbol` is moved to a layer above `TimeSignature` so it is not masked.

```
{
  \override Score.StaffSymbol.layer = 4
  \override Staff.TimeSignature.layer = 3
  b'2 b'~
  \once \override Staff.TimeSignature.whiteout = ##t
  \time 3/4
  b' r4
}
```



36.7.4 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 behavior 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 predefined 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:

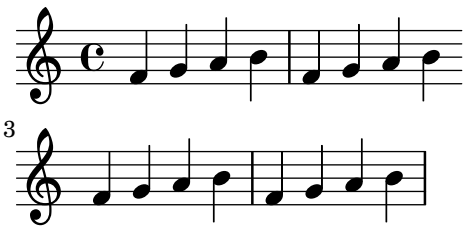
Function form	**Vector form**	**Before break**	**At no break**	**After break**
`all-visible`	`#(#t #t #t)`	yes	yes	yes
`begin-of-line-visible`	`#(#f #f #t)`	no	no	yes
`center-visible`	`#(#f #t #f)`	no	yes	no
`end-of-line-visible`	`#(#t #f #f)`	yes	no	no
`begin-of-line-invisible`	`#(#t #t #f)`	yes	yes	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`
`Divisio`	`Staff`	`begin-of-line-invisible`
`DoublePercentRepeat`	`Voice`	`begin-of-line-invisible`
`KeyCancellation`	`Staff`	`begin-of-line-invisible`
`KeySignature`	`Staff`	`begin-of-line-visible`
`ClefModifier`	`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 bar lines:

```
\relative {
  f'4 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:
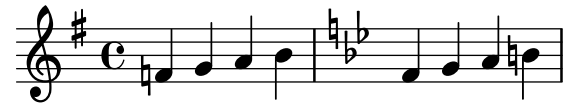
- Bar lines cannot be printed at the start of line.
- A bar number cannot be printed at the start of the *first* line unless it is set to be different from `1`.
- Clef – see the next section.
- Double percent repeats are either *all printed* or *all suppressed*. Use `begin-of-line-invisible` to print them and `all-invisible` to suppress them.
- Key signature – see the next section.
- ClefModifier – see the next section.
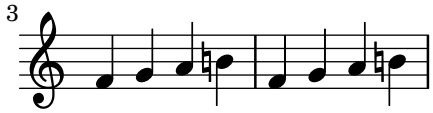
36.7.5 Special considerations

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.
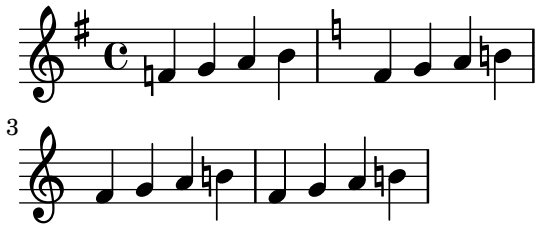
```
\relative {
  \key g \major
  f'4 g a b
  % Try to remove all key signatures
  \override Staff.KeySignature.break-visibility = #all-invisible
  \key bes \major
  f4 g a b
  \break
  f4 g a b
  f4 g a b
}
```





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.
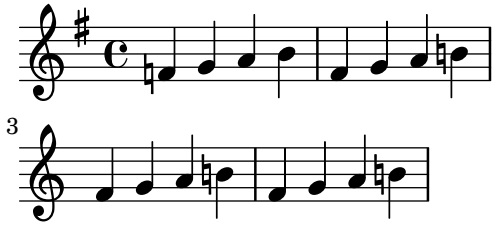
```
\relative {
  \key g \major
  f'4 g a b
  \set Staff.explicitKeySignatureVisibility = #all-invisible
  \override Staff.KeySignature.break-visibility = #all-invisible
  \key bes \major
  f4 g a b \break
  f4 g a b
  f4 g a b
}
```



Visibility of cancelling accidentals

To remove the cancelling accidentals printed at an explicit key change, set the Staff context property `printKeyCancellation` to `#f`:
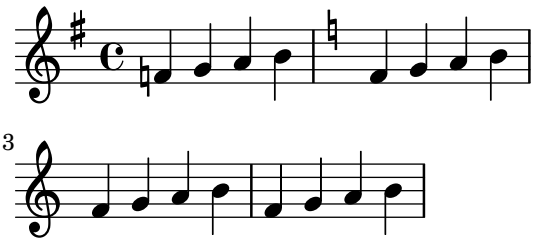
```
\relative {
  \key g \major
  f'4 g a b
  \set Staff.explicitKeySignatureVisibility = #all-invisible
  \set Staff.printKeyCancellation = ##f
  \override Staff.KeySignature.break-visibility = #all-invisible
  \key bes \major
  f4 g a b \break
  f4 g a b
  f4 g a b
}
```



With these overrides only the accidentals before the notes remain to indicate the change of key.
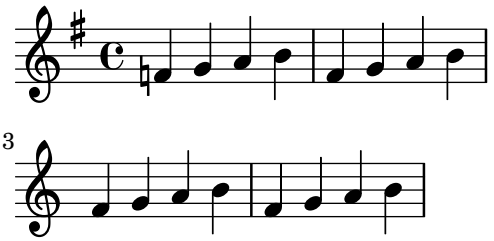
Note that when changing the key to C major or A minor the cancelling accidentals would be the *only* indication of the key change. In this case setting `printKeyCancellation` to `#f` has no effect:

```
\relative {
  \key g \major
  f'4 g a b
  \set Staff.explicitKeySignatureVisibility = #all-invisible
  \set Staff.printKeyCancellation = ##f
  \key c \major
  f4 g a b \break
  f4 g a b
  f4 g a b
}
```



To suppress the cancelling accidentals even when the key is changed to C major or A minor, override the visibility of the `KeyCancellation` grob instead:

```
\relative {
  \key g \major
  f'4 g a b
  \set Staff.explicitKeySignatureVisibility = #all-invisible
  \override Staff.KeyCancellation.break-visibility = #all-invisible
  \key c \major
  f4 g a b \break
  f4 g a b
  f4 g a b
}
```



Transposed clefs

The small transposition symbol on transposed clefs is produced by the `ClefModifier` layout object. Its visibility is automatically inherited from the `Clef` object, so it is not necessary to apply any required `break-visibility` overrides to the `ClefModifier` layout objects to suppress transposition symbols for invisible clefs.

For explicit clef changes, the `explicitClefVisibility` property controls both the clef symbol and any transposition symbol associated with it.

See also

Learning Manual: Section “Visibility and color of objects” in *Learning Manual*.

36.8 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.
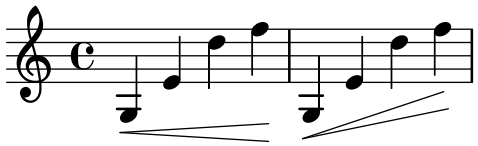
36.8.1 Rotating layout objects

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:
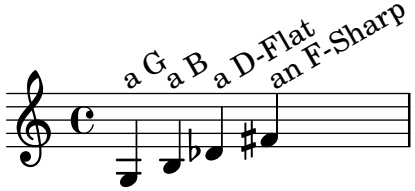
```
g4\< e' d'' f''\!
\override Hairpin.rotation = #'(15 -1 0)
g4\< e' d'' f''\!
```



36.8.2 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.

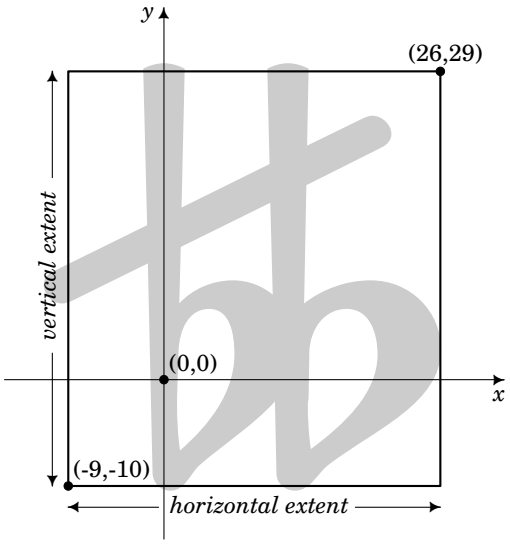
```
\override TextScript.outside-staff-priority = ##f
g4^\markup { \rotate #30 "a G" }
b^\markup { \rotate #30 "a B" }
des'^\markup { \rotate #30 "a D-Flat" }
fis'^\markup { \rotate #30 "an F-Sharp" }
```



36.9 Aligning objects

All graphical objects have a *reference point*; coordinate-wise, this reference point is defined to have position (0,0).

All grobs also have 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. In the image below, the point with zero coordinates is the reference point; the horizontal extent is `#'(-9 . 26)`, the vertical extent is `#'(-10 . 29)`.



See Section "LilyPond's box model" in *Extending* for details on extents of music glyphs.

Graphical objects also have *alignment points*, which are not necessarily the same as their reference point. An alignment point is a point relative to an object that LilyPond uses when aligning that object with other objects. Self-alignment points are relative to the object itself; parent-alignment points are relative to the object's parent, if one exists.

Most grobs have parents that influence their positioning. For these objects, LilyPond moves the grob so that its alignment point is at the same position as the parent's alignment point. For example, `TextScript.parent-alignment-X` is the property of a `TextScript` grob that specifies the alignment point along the X axis, relative to that `TextScript` grob's parent (its associated `NoteColumn` grob), to which LilyPond aligns that `TextScript` grob. When placing a `TextScript` grob horizontally, LilyPond aligns the grob's self-alignment point with a parent-alignment point relative to the `NoteColumn` grob that is the `TextScript`'s parent. If the `NoteColumn` grob has two or more children, those child grobs can have different parent-alignment points. See Section 36.9.3 [Using the `self-alignment-interface`], page 794, for more information.

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.[1] 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.

[1] For objects not directly associated with a staff other conventions might apply. To name an example, `Y-offset` values for `InstrumentName` grobs are measured from the top of the current system's topmost staff.

For example, an accidental can be repositioned vertically by setting `Y-offset` but any changes to `X-offset` have no effect.

Many graphical objects can be aligned to a previously placed object using the `self-alignment-interface` and/or the `side-position-interface`. For a list of these objects, see Section "self-alignment-interface" in *Internals Reference* and Section "side-position-interface" in *Internals Reference*.

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
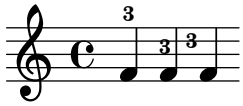
Notation Reference: Section 36.9.4 [Using the `break-alignable-interface`], page 797.

Extending LilyPond: Section "Callback functions" in *Extending*.

36.9.1 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.

```
f'-3
f'-\tweak X-offset 0
  -\tweak Y-offset 0
  -3
f'-\tweak X-offset -1
  -\tweak Y-offset 0.5
  -3
```



36.9.2 Using the `side-position-interface`

An object that 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 returns 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 returns the correct value of `Y-offset` to place the object to the top or bottom of the parent according to value of `direction`.

Instead of values 0 and 1 you might use the Scheme symbols `X` and `Y` for specifying a side axis. Similarly, you might use the symbols `DOWN` (`LEFT`) and `UP` (`RIGHT`) instead of values -1 and 1 for specifying a direction.

36.9.3 Using the `self-alignment-interface`

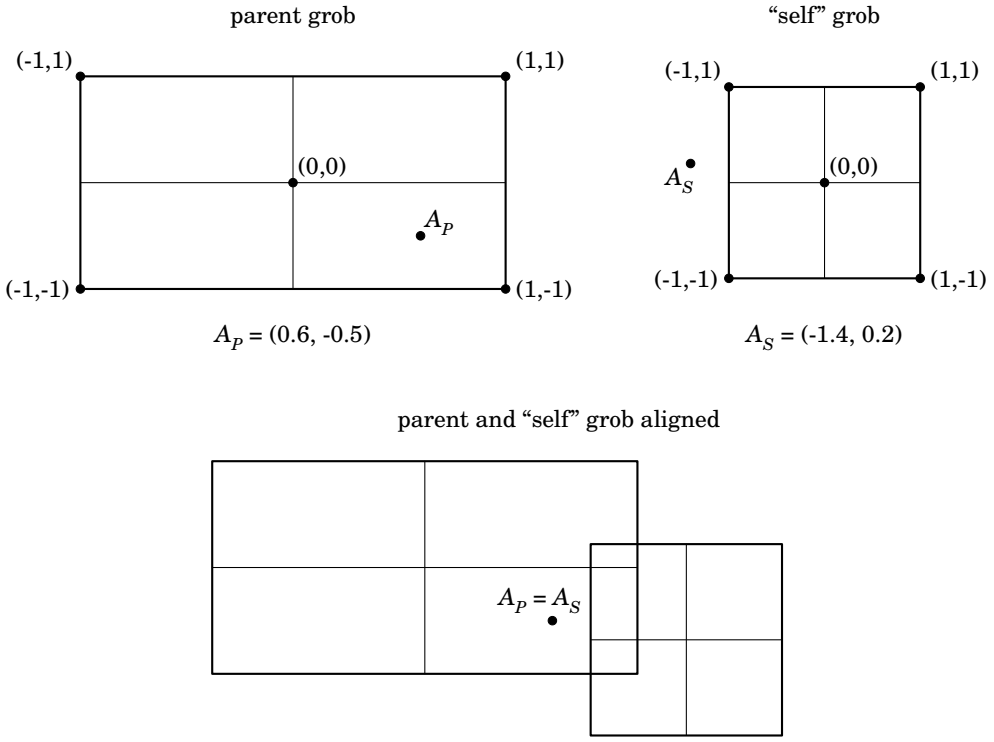
The `self-alignment-interface` defines objects' alignment points. This interface applies to many graphical objects in LilyPond (see Section "self-alignment-interface" in *Internals Reference* for a list).

The alignment interface is based on the dimension of the grob's bounding box. The X (Y) coordinate of a grob's alignment point is a multiple of half the width (height) of this bounding box. In other words, one unit is half the total X (Y) extent of the object.

An X (Y) coordinate of -1 represents the left (lower) edge of the bounding box. An X (Y) coordinate of 0 represents the center. An X (Y) coordinate of 1 represents the right (upper) edge.

A coordinate can be any real number, including values less than -1 or greater than 1.

The image below shows the general case, where both the parent and the 'self' grob have non-zero horizontal and vertical alignment parameters to position the alignment points A_P and A_S (these two points are chosen arbitrarily here for the sake of demonstrating the concept).
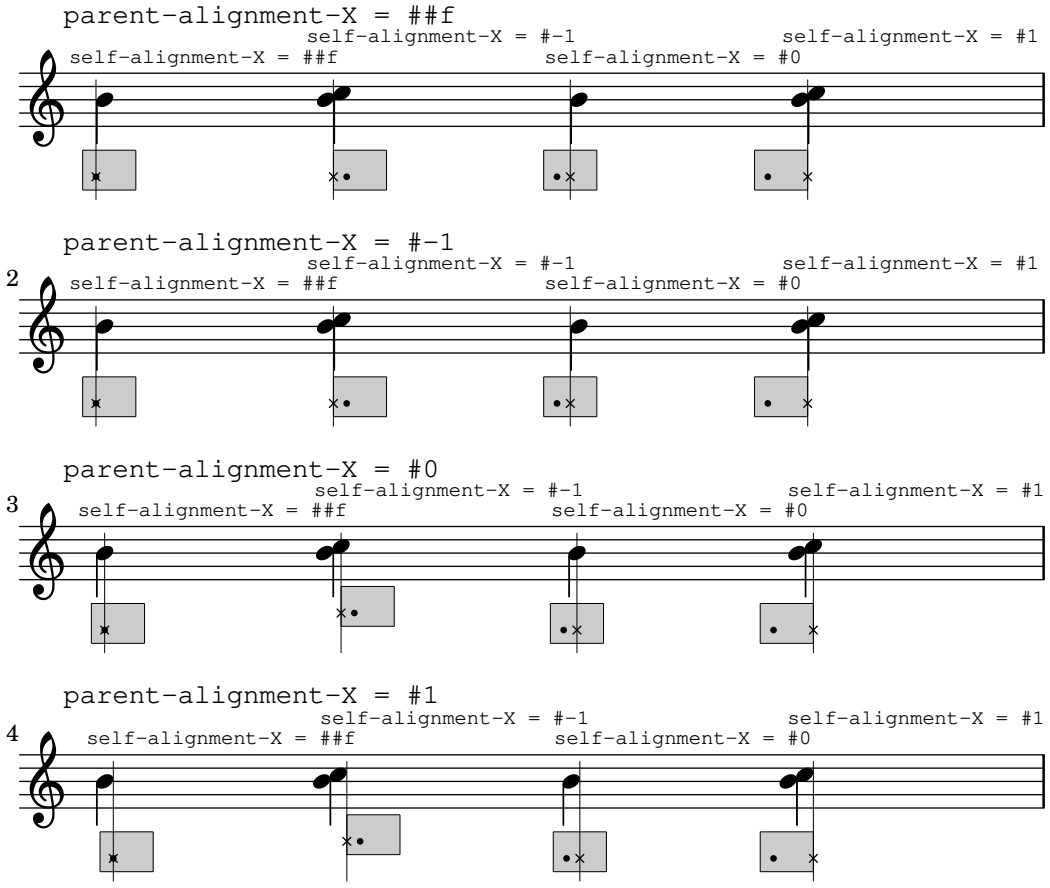


Self-aligning and parent-aligning objects horizontally

The horizontal alignment of an object that 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`. Negative values move the object to the right, positive to the left. A value of 0 centers the object (more specifically, its bounding box) on the alignment point of its parent, a value of -1 aligns the left edge of the object to the alignment point of its parent, and a value of 1 aligns the right edge of the object to the alignment point of its parent. The symbols `LEFT`, `CENTER`, and `RIGHT` may be used instead of the values -1, 0, and 1, respectively.

Value `#f` horizontally aligns the grob using its reference point. This value only works, however, if the grob has a parent; LilyPond returns an error if it has none.

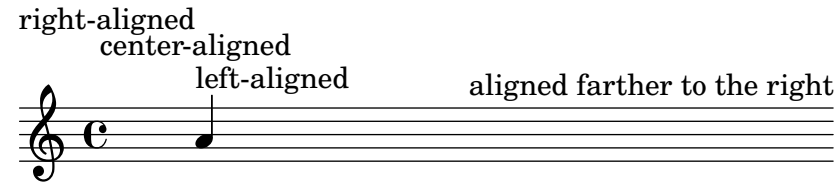
If an object has a parent, its horizontal alignment is also similarly affected by the `parent-alignment-X` property.

The following image shows various `self-alignment-X` and `parent-alignment-X` value combinations for `TextScript` objects whose parents are their associated `NoteColumn` grobs. Here, the `TextScript` grobs are gray boxes having the horizontal extent `'(-1 . 3)` and the vertical extent `'(-1 . 2)`; the reference point of the boxes is marked with a dot. The reference point of a `NoteColumn` grob is always (0,0); suspended `NoteHead` grobs (i.e., note heads in chords on the other, non-default side of the stem) stick out but are not taken into account. The horizontal position of the parent-alignment point is represented by a vertical line, and the alignment point of `TextScript` is marked with a cross.



Either `\override` or `\tweak` can be used to modify the value of `self-alignment-X`. To separately align several annotations on a single note, however, you must use `\tweak`.

```
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 farther to the right"
```



Self-aligning objects vertically

Objects may be aligned vertically (using `self-alignment-Y`) 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 very large numbers may be required. A value of -1 aligns the lower edge of the object with the alignment point of the parent object, a value of 0 aligns the center of the object with the alignment point of the parent, and a value of 1 aligns the top edge of the object with the alignment point of the parent. The symbols `DOWN`, `CENTER`, and `UP` may be substituted for -1, 0, and 1, respectively.

Value `#f` vertically aligns the grob using its reference point. This value only works, however, if the grob has a parent; LilyPond returns an error if it has none.

Note that using `parent-alignment-Y` almost never works; there are too many other constraints that influence how an object is vertically positioned relative to its parent grob. Take the previous image as an example: the vertical value of the parent's reference point (the `NoteColumn` grob) depends on the attached object (the `TextScript` grob) together with various padding property values – as can be seen, LilyPond tries to move the `TextScript` grobs as near as possible to the staff. Due to this interdependency between parent and object itself, accessing or modifying `parent-alignment-Y` usually fails and often produces warnings about 'cyclic dependencies'.

The standard way to adjust vertical distances is to modify one of the various padding properties LilyPond offers, for example, `outside-staff-padding`.

Self-aligning objects in both directions

By setting both `self-alignment-X` and `self-alignment-Y`, an object may be aligned in both directions simultaneously.

The following example shows how to adjust the position of instrument names. The `InstrumentName` grob is an example of a grob that has no parent: `self-alignment-X` gives an alignment relative to the left margin and the start of the staff (i.e., the system's indentation), and `self-alignment-Y` aligns the `InstrumentName` grob relative to the related staff's middle line.

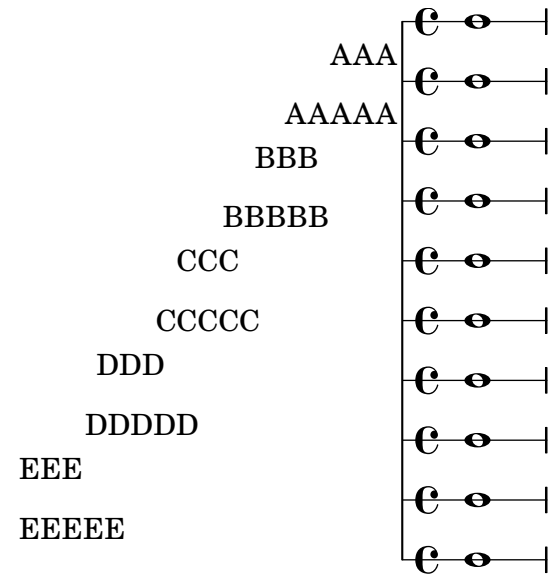
```
sn = #(define-music-function (name x y) (string? number? number?)
       #{
         \new RhythmicStaff \with {
           instrumentName = #name
           \override InstrumentName.self-alignment-X = #x
           \override InstrumentName.self-alignment-Y = #y
         } { f'1 }
       #})

<<
  \sn "AAA" #1 #3
  \sn "AAAAA" #1 #3
  \sn "BBB" #0.5 #1.5
  \sn "BBBBB" #0.5 #1.5
  \sn "CCC" #0 #0
  \sn "CCCCC" #0 #0
  \sn "DDD" #-0.5 #-1.5
  \sn "DDDDD" #-0.5 #-1.5
  \sn "EEE" #-1 #-3
```

```
  \sn "EEEEE" #-1 #-3
>>

\layout {
  indent = 5\cm

  \context {
    \RhythmicStaff
    \override VerticalAxisGroup.staff-staff-spacing =
      #'((minimum-distance . 0)
         (basic-distance . 0)
         (padding . 2.5))
  }
}
```



36.9.4 Using the break-alignable-interface

Rehearsal marks, text marks, bar numbers, and more generally all objects that support the break-alignable-interface, collectively referred to as "break-alignable objects", may be aligned with notation objects such as bar lines, key signatures, time signatures, and generally any object that supports the break-aligned-interface. To be more precise, break-aligned items have a break-align-symbol property, providing symbols that can be used as 'anchor points' for other objects via the break-align-symbols property. See Section "break-alignment-interface" in *Internals Reference*, for a complete list of available symbols.
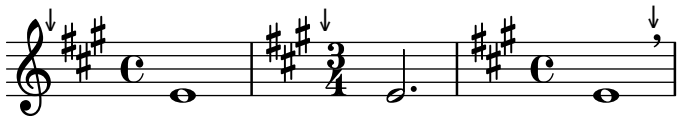
Each type of object has its own default reference point, to which break-alignable items are aligned:

```
\override Score.TextMark.self-alignment-X = #CENTER
% Align text mark to the right edge of the clef.
\override Score.TextMark.break-align-symbols = #'(clef)
\clef treble
\key a \major
\textMark "↓"
e'1

% Align text mark to the left edge of the time signature.
```

```
\override Score.TextMark.break-align-symbols = #'(time-signature)
\key a \major
\time 3/4
\textMark "↓"
e'2.

% Center text mark above the breathing mark.
\override Score.TextMark.break-align-symbols = #'(breathing-sign)
\key a \major
\time 4/4
e'1
\breathe
\textEndMark "↓"
```

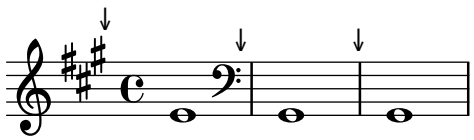


A list of possible target alignment objects may be specified. If some of the objects are invisible at that point due to the setting of `break-visibility` or the explicit visibility settings for keys and clefs, the rehearsal mark or bar number is aligned to the first object in the list which is visible. 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.

```
\override Score.TextMark.self-alignment-X = #CENTER
% Align text mark to the right edge of the key signature.
\override Score.TextMark.break-align-symbols = #'(key-signature clef)
\key a \major
\clef treble
\textMark "↓"
e'1

% Align text mark to the right edge of the clef.
\set Staff.explicitKeySignatureVisibility = #all-invisible
\override Score.TextMark.break-align-symbols = #'(key-signature clef)
\key a \major
\clef bass
\textMark "↓"
gis,1

% Center text mark above the bar line.
\set Staff.explicitKeySignatureVisibility = #all-invisible
\set Staff.explicitClefVisibility = #all-invisible
\override Score.TextMark.break-align-symbols = #'(key-signature clef)
\key a \major
\clef treble
\textMark "↓"
e'1
```
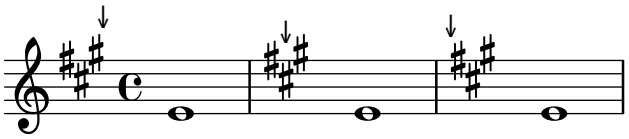


The alignment of the break-alignable item relative to the notation object can be changed, as shown in the following example. In a score with multiple staves, this setting should be done for all the staves.

```
\override Score.TextMark.self-alignment-X = #CENTER
% Align text mark with the right edge of the key signature.
\override Score.TextMark.break-align-symbols = #'(key-signature)
\key a \major
\clef treble
\time 4/4
\textMark "↓"
e'1

% Center text mark above the key signature
\once \override
  Score.KeySignature.break-align-anchor-alignment = #CENTER
\textMark "↓"
\key a \major
e'1

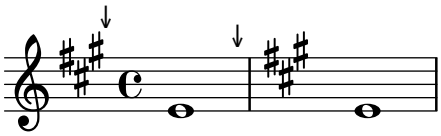
% Align text mark with the left edge of the key signature.
\once \override
  Score.KeySignature.break-align-anchor-alignment = #LEFT
\key a \major
\textMark "↓"
e'1
```



The break-alignable item can also be offset to the right or left of the left edge by an arbitrary amount. The units are staff spaces:

```
\override Score.TextMark.self-alignment-X = #CENTER
% Align text mark with the left edge of the key signature,
% then shifted right by 3.5 staff spaces.
\override Score.TextMark.break-align-symbols = #'(key-signature)
\once \override Score.KeySignature.break-align-anchor = 3.5
\key a \major
\textMark "↓"
e'1

% Align text mark with the left edge of the key signature,
% then shifted left by 2 staff spaces.
\once \override Score.KeySignature.break-align-anchor = -2
\key a \major
\textMark "↓"
e'1
```



See also

Internals Reference: Section "break-alignable-interface" in *Internals Reference*, Section "break-alignment-interface" in *Internals Reference*.

36.10 Modifying stencils
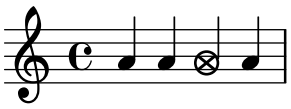
All layout objects have a `stencil` property which is part of the `grob-interface`. By default, this property is usually set to a function specific to the object that is tailor-made to render the symbol which represents it in the output. For example, the standard setting for the `stencil` property of 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 Section 8.2.4 [Graphic notation inside markup], page 332.

The following example demonstrates this by changing the note head symbol to a cross within a circle.

```
XinO = {
  \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 {
  a' a \XinO a a
}
```



Any of the *Feta* glyphs used in the Emmentaler font can be supplied to the `\musicglyph` markup command – see Section B.8 [The Emmentaler font], page 907.

`EPS` files and Postscript commands can both be inserted inline using the `\epsfile` and `\postscript` markup commands respectively – see Section A.1.3 [Graphical markup], page 840.

See also

Notation Reference: Section 8.2.4 [Graphic notation inside markup], page 332, Section 8.2 [Formatting text], page 320, Section A.1 [Text markup commands], page 809, Section B.8 [The Emmentaler font], page 907, Section A.1.3 [Graphical markup], page 840.

36.11 Modifying shapes

36.11.1 Modifying ties and slurs

`Ties`, `Slurs`, `PhrasingSlurs`, `LaissezVibrerTies` and `RepeatTies` are all 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 in two ways:

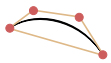
a. by specifying the displacements to be made to the control points of the automatically calculated Bézier curve, or

b. by explicitly specifying the positions of the four control points required to define the wanted curve.

Both methods are explained below. The first method is more suitable if only slight adjustments to the curve are required; the second may be better for creating curves which are related to just a single note.

Cubic Bézier curves

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 and image 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 traveling directly from the third control point. The curve is entirely contained in the quadrilateral defined by the four control points.
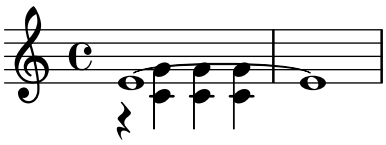


Translations, rotations and scaling of the control points all result in exactly the same operations on the curve.

Specifying displacements from current control points

In this example the automatic placement of the tie is not optimum, and `\tieDown` would not help.

```
<<
  { e'1~ 1 }
\\
  \relative { r4 <g' c,> <g c,> <g c,> }
>>
```



Adjusting the control points of the tie with `\shape` allows the collisions to be avoided.

The syntax of `\shape` is

`[-]\shape` *`displacements item`*

This will reposition the control points of *item* by the amounts given by *displacements*. The *displacements* argument is a list of number pairs or a list of such lists. Each element of a pair represents the displacement of one of the coordinates of a control point. If *item* is a string, the
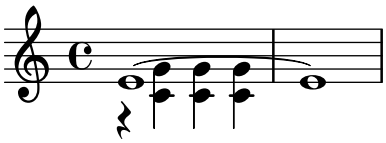
result is \once\override for the specified grob type. If *item* is a music expression, the result is the same music expression with an appropriate tweak applied.

In other words, the \shape function can act as either a \once\override command or a \tweak command depending on whether the *item* argument is a grob name, like "Slur", or a music expression, like "(". The *displacements* argument specifies the displacements of the four control points as a list of four pairs of (dx . dy) values in units of staff spaces (or a list of such lists if the curve has more than one segment).

The leading hyphen is required if and only if the \tweak form is being used.
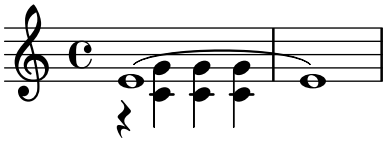
So, using the same example as above and the \once\override form of \shape, this will raise the tie by half a staff space:

```
<<
  {
    \shape #'((0 . 0.5) (0 . 0.5) (0 . 0.5) (0 . 0.5)) Tie
    e'1~ 1
  }
\\
  \relative { r4 <g' c,> <g c,> <g c,> }
>>
```
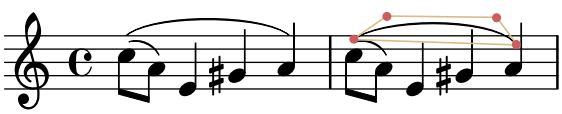


This positioning of the tie is better, but maybe it should be raised more in the center. The following example does this, this time using the alternative \tweak form:

```
<<
  {
    e'1-\shape #'((0 . 0.5) (0 . 1) (0 . 1) (0 . 0.5)) ~ e'
  }
\\
  \relative { r4 <g' c,> <g c,> <g c,> }
>>
```



To aid the tweaking process, the \vshape function is provided. Its name means *visual shape*: it acts exactly like \shape, except that the control points and polygon are additionally displayed.
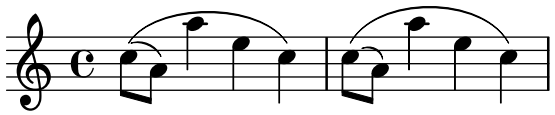
```
\relative {
  c''8(\( a) e4 gis a\)
  \vshape #'((0 . -0.3) (0.5 . -0.2)
             (0.5 . -0.3) (0 . -0.7)) PhrasingSlur
  c8(\( a) e4 gis a\)
}
```



It is advisable to start with \vshape and adjust until a satisfactory curve is obtained, then simply remove the "v" letter in \vshape.
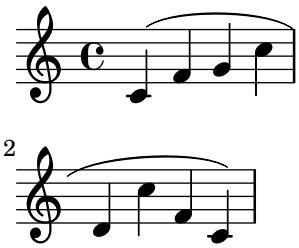
Two different curves starting at the same musical moment may also be shaped:

```
\relative {
  c''8(\( a) a'4 e c\)
  \shape #'((0.7 . -0.4) (0.5 . -0.4)
            (0.3 . -0.3) (0 . -0.2)) Slur
  \shape #'((0 . 0) (0 . 0.5)
            (0 . 0.5) (0 . 0)) PhrasingSlur
  c8(\( a) a'4 e c\)
}
```
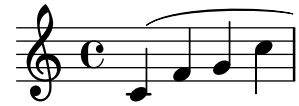


The \shape function can also displace the control points of curves which stretch across line breaks. Each piece of the broken curve can be given its own list of offsets. If changes to a particular segment are not needed, the empty list can serve as a placeholder. In this example the line break makes the single slur look like two:
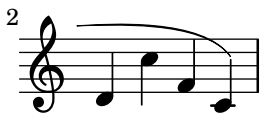
```
\relative {
  c'4( f g c
  \break
  d,4 c' f, c)
}
```



Changing the shapes of the two halves of the slur makes it clearer that the slur continues over the line break:
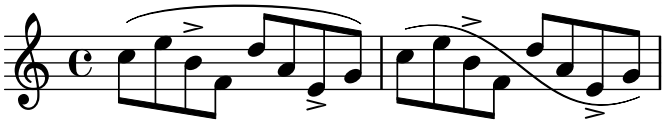
```
% () may be used as a shorthand for ((0 . 0) (0 . 0) (0 . 0) (0 . 0))
% if any of the segments does not need to be changed
\relative c' {
  \shape #'(
             (( 0 . 0) (0 . 0) (0 . 0) (0 . 1))
             ((0.5 . 1.5) (1 . 0) (0 . 0) (0 . -1.5))
           ) Slur
  c4( f g c
  \break
  d,4 c' f, c)
}
```





If an S-shaped curve is required the control points must always be adjusted manually – LilyPond will never select such shapes automatically.

```
\relative c'' {
  c8( e b-> f d' a e-> g)
  \shape #'((0 . -1) (5.5 . -0.5) (-5.5 . -10.5) (0 . -5.5))
         PhrasingSlur
  c8\( e b-> f d' a e-> g\)
}
```

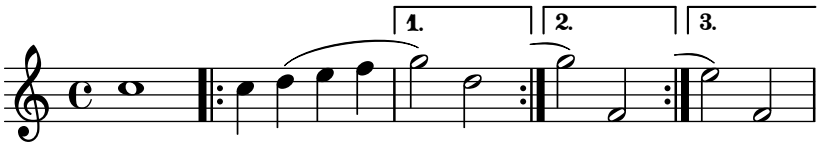


Specifying control points explicitly

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 specified 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. Be aware that these values may need to be manually adjusted if any further changes are made to the music or the layout.

One situation where specifying the control points explicitly is preferable to specifying displacements is when they need to be specified relative to a single note. Here is an example of this. It shows one way of indicating a slur extending into alternative sections of a volta repeat.

```
\relative {
  c''1
  \repeat volta 3 { c4 d( e f }
  \alternative {
    \volta 1 { g2) d }
    \volta 2 {
      g2
        % create a slur and move it to a new position
        % the <> is just an empty chord to carry the slur termination
        -\tweak control-points
                  #'((-2 . 3.8) (-1 . 3.9) (0 . 4) (1 . 3.4)) ( <> )
      f,
    }
    \volta 3 {
      e'2
        % create a slur and move it to a new position
        -\tweak control-points
                  #'((-2 . 3) (-1 . 3.1) (0 . 3.2) (1 . 2.4)) ( <> )
      f,
    }
  }
}
```



Known issues and warnings

It is not possible to modify shapes of ties or slurs by changing the `control-points` property if there are multiple ties or slurs at the same musical moment – the `\tweak` command will also not work in this case. However, the `tie-configuration` property of `TieColumn` can be overridden to set start line and direction as required.

See also

Internals Reference: Section "TieColumn" in *Internals Reference*.

Appendices

Appendix A Markup commands

A.1 Text markup commands

The following commands can all be used inside `\markup { }`.

A.1.1 Font markup

`\abs-fontsize` *size* (number) *arg* (markup)

Use *size* as the absolute font size (in points) to display *arg*.

This function adjusts the `baseline-skip` and `word-space` properties 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

Used properties:

- `baseline-skip` (3)
- `word-space` (0.6)

`\bold` *arg* (markup)

Print *arg* with a bold face.

```
\markup {
  default
  \hspace #2
  \bold bold
}
```

default **bold**

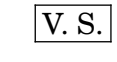
The code `\markup \bold ...` is a shorthand for `\markup \override #'(font-series . bold) ...` – using the more verbose form, it is possible to obtain nuances such as semi-bold, if the text font has such variants. Refer to the documentation for the `font-series` properties (Section "User backend properties" in *Internals Reference*).

`\box` *arg* (markup)

Draw a box around *arg*.

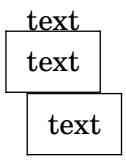
This function looks at the `thickness`, `box-padding`, and `font-size` properties to determine the line thickness and padding around the markup.

```
\markup {
  \override #'(box-padding . 0.5)
  \box \line { V. S. }
}
```



Note that the box does not horizontally displace its argument. Use markup commands like `\left-align` or `\table` to make LilyPond realign it.

```
\markup {
  \override #'(box-padding . 1.5)
  \column {
    "text"
    \box "text"
    \left-align \box "text"
  }
}
```



Used properties:

- `box-padding` (`0.2`)
- `font-size` (`0`)
- `thickness` (`1`)

`\caps` *arg* (markup)

Print *arg* in (fake) small caps.

This function is a copy of the `\smallCaps` command.

```
\markup {
  default
  \hspace #2
  \caps {
    Text in small caps
  }
}
```

default TEXT IN SMALL CAPS

Use `\fontCaps` for real small caps (if the font provides it).

`\dynamic` *arg* (markup)

Print *arg* using the (music) font for dynamics.

This font only contains letters **f**, **m**, **n**, **p**, **r**, **s**, and **z**. 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

`\figured-bass` *arg* (markup)

Set *arg* as small numbers for figured bass.

Specially slashed digits can be achieved with a trailing backslash (for numbers 6, 7, and 9) or a trailing plus (for numbers 2, 4, and 5).[1]

[1] Internally, this works by activating the 'dlig' OpenType feature of the Emmentaler font.

The use of a backslash is in analogy to `\figuremode` (see Section 15.3.2 [Entering figured bass], page 527). Note that to get a backslash character in markup it must be escaped by doubling it. Additionally, it must be put into double quotes.

```
\markup {
  \figured-bass {
    2 3 4+ 7 "9\\"
  }
}
```

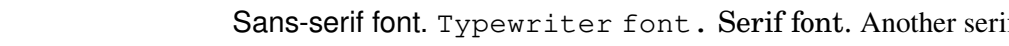

`\finger` *arg* (markup)
: Set *arg* as small numbers for fingering instructions.

```
\markup {
  \finger {
    1 2 3 4 5
  }
}
```


`\font-select` *defs* (pair) *arg* (markup)
: Output the markup *arg* using fonts specified in *defs*.

defs is either a single pair (*family* . *font-name*) or a non-empty list of such pairs. Here, *family* is usually `serif`, `sans` or `typewriter`; *font-name* is a string. Font families not mentioned in the call to `\font-select` remain unchanged.

```
\markup {
  \font-select #'((sans . "Liberation Sans")
                  (typewriter . "Liberation Mono"))
  {
    \sans { Sans-serif font. }
    \typewriter { Typewriter font. }
    Serif font.
    \font-select #'(serif . "Liberation Serif") {
      Another serif font.
    }
  }
}
```



`\fontCaps` *arg* (markup)
: Print *arg* in small caps.

This command sets the `font-variant` property to `small-caps`.

Unlike `\smallCaps`, which merely uses capital letters at a smaller font size, this uses the actual variant of the font for small caps. (As a consequence, if you configure a custom text font, this command has no effect if that font does not have a small caps variant.)

```
\markup \fontCaps "Small caps"
```


`\fontsize` *increment* (number) *arg* (markup)
: Increase current font size by *increment* to print *arg*.

This function adjusts the `baseline-skip` and `word-space` properties 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 value 2 to print *arg*.

```
\markup {
  default
  \hspace #2
  \huge huge
}
```

default huge

`\italic` *arg* (markup)
Print *arg* in italics.

This command sets the `font-shape` property to `italic`.

```
\markup {
  default
  \hspace #2
  \italic italic
}
```

default *italic*

`\large` *arg* (markup)
Set font size to value 1 to print *arg*.

```
\markup {
  default
  \hspace #2
  \large large
}
```

default large

`\larger` *arg* (markup)
Increase current font size by 1 to print *arg*.

This function adjusts the `baseline-skip` and `word-space` properties accordingly.

```
\markup {
  default
  \hspace #2
  \larger larger
}
```

default larger

\magnify *sz* (number) *arg* (markup)
: Magnify current font by factor *sz* to print *arg*.

```
\markup {
  default
  \hspace #2
  \magnify #1.5 {
    50% larger
  }
}
```

default 50% larger

Note that magnification only works if a font name is explicitly selected. Use \fontsize otherwise.

\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:

- font-size (0)

\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

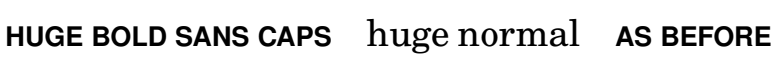
Used properties:

- font-size (0)

\normal-text *arg* (markup)
: Print *arg* with default text font.

  This resets all font-related properties (except the size), no matter what font was used earlier.

```
\markup {
  \huge \bold \sans \fontCaps {
    huge bold sans caps
    \hspace #2
    \normal-text {
      huge normal
    }
    \hspace #2
    as before
  }
}
```



\normal-weight *arg* (markup)

Switch to normal weight (in contrast to bold) to print *arg*.

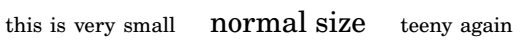
This command sets the `font-series` property to `normal`.

```
\markup {
  \bold {
    some bold text
    \hspace #2
    \normal-weight {
      normal font series
    }
    \hspace #2
    bold again
  }
}
```


\normalsize *arg* (markup)

Set font size to default (i.e., to value 0) to print *arg*.

```
\markup {
  \teeny {
    this is very small
    \hspace #2
    \normalsize {
      normal size
    }
    \hspace #2
    teeny again
  }
}
```



\number *arg* (markup)

Print *arg* using the (music) font for numbers.

This font also contains symbols for figured bass, some punctuation, spaces of various widths, some letters and text variants of accidentals, the common time, and the cut time symbol. Use `\dynamic` to access the (very small number of) letters. For

accidentals you might use \number in combination with Unicode characters to access the text representation forms of accidental glyphs, as the following table shows.

Unicode value	Unicode character
U+266D	♭
U+266E	♮
U+266F	♯
U+1D12A	𝄪
U+1D12B	𝄫
U+1D134	𝄴
U+1D135	𝄵

Examples:

```
\number ♭ → ♭
\number { \char ##x266F } → ♯
```

To get accidentals protected against overrides of font-name it is preferable to use \text-doubleflat, \text-flat, \text-natural, \text-sharp, \text-doublesharp, or the general \text-accidental for the text variants of accidentals. For the time signature symbols, use \text-common-time and \text-cut-time.

The appearance of digits in the Emmentaler font can be controlled with four OpenType features: 'tnum', 'cv47', 'ss01', and 'kern', which can be arbitrarily combined.

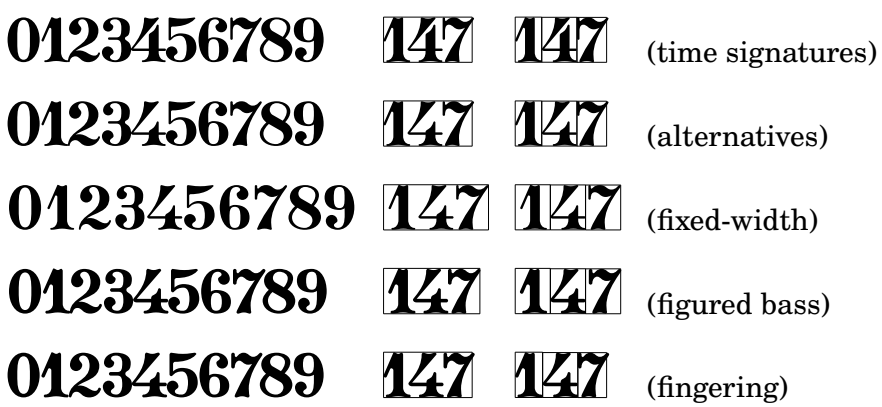
tnum — If off (which is the default), glyphs 'zero' to 'nine' have no left and right side bearings. If on, the glyphs all have the same advance width by making the bearings non-zero.

cv47 — If on, glyphs 'four' and 'seven' have shorter vertical strokes. Default is off.

ss01 — If on, glyphs 'zero' to 'nine' have a fatter design, making them more readable at small sizes. Default is off.

kern — If on (which is the default), provide pairwise kerning between (most) glyphs.

```
\markuplist
  \number
  \fontsize #4.5
  \override #'((padding . 2)
               (baseline-skip . 4)
               (box-padding . 0)
               (thickness . 0.1))
  \table #'(-1 -1 -1 -1) {
      0123456789 \box 147 \concat { \box 1 \box 4 \box 7 }
    \normal-text \normalsize "(time signatures)"
    \override #'(font-features .("cv47")) {
      0123456789 \box 147 \concat { \box 1 \box 4 \box 7 } }
    \normal-text \normalsize "(alternatives)"
    \override #'(font-features .("tnum" "cv47" "-kern")) {
      0123456789 \box 147 \concat { \box 1 \box 4 \box 7 } }
    \normal-text \normalsize "(fixed-width)"
    \override #'(font-features . ("tnum" "cv47" "ss01")) {
      0123456789 \box 147 \concat { \box 1 \box 4 \box 7 } }
    \normal-text \normalsize "(figured bass)"
    \override #'(font-features . ("cv47" "ss01")) {
      0123456789 \box 147 \concat { \box 1 \box 4 \box 7 } }
    \normal-text \normalsize "(fingering)"
  }
```



See also the markup commands `\figured-bass` and `\finger`, which set the font features accordingly.

`\overtie` *arg* (markup)

Print a tie over *arg*.

```
\markup \line {
  \overtie "overtied"
  \override #'((offset . 5) (thickness . 1))
  \overtie "overtied"
  \override #'((offset . 1) (thickness . 5))
  \overtie "overtied"
}
```



Used properties:

- `shorten-pair` (`(0 . 0)`)

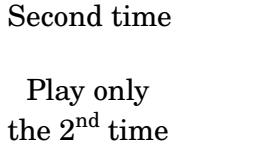
- `height-limit` (0.7)
- `direction` (1)
- `offset` (2)
- `thickness` (1)

`\replace` *replacements* (list) *arg* (markup)

Use *replacements* to replace strings in *arg*.

Each (*key* . *value*) pair of the *replacements* alist specifies what should be replaced; *key* gets replaced by *value*. Note the quasi-quoting syntax with a backquote in the second example.

```
\markup \replace #'(("2nd" . "Second"))
  "2nd time"
\markup \replace
  #`(("2nd" . ,#{ \markup \concat { 2 \super nd } #}))
  \center-column {
    \line { Play only }
    \line { the 2nd time }
  }
```



Used properties:

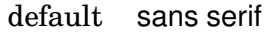
- `replacement-alist`

`\sans` *arg* (markup)

Print *arg* with a sans-serif font.

This command sets the `font-family` property to `sans`.

```
\markup {
  default
  \hspace #2
  \sans {
    sans serif
  }
}
```



`\serif` *arg* (markup)

Print *arg* with a serif font.

This command sets the `font-family` property to `serif`.

```
\markup {
  \sans \bold {
    sans serif, bold
    \hspace #2
    \serif {
      text in serif font
    }
    \hspace #2
    return to sans
```

```
  }
}
```

sans serif, bold text in serif font return to sans

\simple *str* (string)
: Print string *str*.

  \markup \simple "x" is equivalent to \markup "x". This command was previously used internally, but no longer is, and is being kept for backward compatibility only.

\small *arg* (markup)
: Set font size to value -1 to print *arg*.

```
\markup {
  default
  \hspace #2
  \small small
}
```

default small

\smallCaps *arg* (markup)
: Print *arg* in (fake) small caps.

  Unlike \fontCaps, which uses the actual small caps variant of the current font, this fakes small caps by using capital letters at a smaller font size. It can thus be used for fonts that don't have a small caps variant.

```
\markup {
  default
  \hspace #2
  \smallCaps {
    Text in small caps
  }
}
```

default TEXT IN SMALL CAPS

\smaller *arg* (markup)
: Decrease current font size by 1 to print *arg*.

  This function adjusts the baseline-skip and word-space properties accordingly.

```
\markup {
  \fontsize #3.5 {
    large text
    \hspace #2
    \smaller { smaller text }
    \hspace #2
    large text
  }
}
```

large text smaller text large text

\sub *arg* (markup)
: Set *arg* in subscript.

```
\markup { \concat { H \sub 2 O } }
```



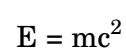
See also \super.

Used properties:

- font-size (0)

\super *arg* (markup)
Set *arg* in superscript.

```
\markup { E = \concat { mc \super 2 } }
```



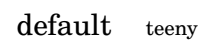
See also \sub.

Used properties:

- font-size (0)

\teeny *arg* (markup)
Set font size to value -3 to print *arg*.

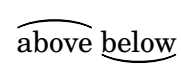
```
\markup {
  default
  \hspace #2
  \teeny teeny
}
```



\tie *arg* (markup)
Add a horizontal bow at the bottom or top of *arg*.

This function uses make-tie-stencil to create the bow; it looks at the thickness and offset properties to determine the line thickness and vertical offset, respectively. The added bow fits the extent of *arg*; use the shorten-pair property to modify this. The direction property may be set explicitly using override or direction modifiers, or implicitly by using voiceOne, etc.

```
\markup {
  \override #'(direction . 1)
  \tie "above"
  \override #'(direction . -1)
  \tie "below"
}
```



See also \undertie and \overtie, which are shorthands for this function.

Used properties:

- shorten-pair ((0 . 0))
- height-limit (0.7)
- direction (1)
- offset (2)
- thickness (1)

\tiny *arg* (markup)
Set font size to value -2 to print *arg*.

```
\markup {
  default
  \hspace #2
  \tiny tiny
}
```

default tiny

\typewriter *arg* (markup)
Print *arg* in typewriter.

This command sets the `font-family` property to `typewriter`, also switching off the 'liga' OpenType feature to disable ligatures like 'ff' or 'fi'.

```
\markup {
  "default fi ff"
  \hspace #2
  \typewriter "typewriter fi ff"
}
```

default fi ff typewriter fi ff

\underline *arg* (markup)
Underline *arg*.

This function looks at the property `thickness` to determine the line thickness, at `offset` to determine the line's vertical offset from *arg*, and at `underline-skip` to determine the distance of additional lines from the others.

The `underline-shift` property is used to make subsequent calls work correctly. Overriding it makes little sense since it would end up adding the provided value to the one of `offset`.

```
\markup \justify-line {
  \underline "underlined"
  \override #'(offset . 5)
  \override #'(thickness . 1)
  \underline "underlined"
  \override #'(offset . 1)
  \override #'(thickness . 5)
  \underline "underlined"
  \override #'(offset . 5)
  \override #'(underline-skip . 4)
  \underline \underline \underline "underlined thrice"
}
```

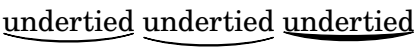
underlined underlined underlined underlined thrice

Used properties:

- `underline-skip` (2)
- `underline-shift` (0)
- `offset` (2)
- `thickness` (1)

\undertie *arg* (markup)
Print a tie under *arg*.

```
\markup \line {
  \undertie "undertied"
  \override #'((offset . 5) (thickness . 1))
  \undertie "undertied"
  \override #'((offset . 1) (thickness . 5))
  \undertie "undertied"
}
```



Used properties:

- `shorten-pair` (`(0 . 0)`)
- `height-limit` (`0.7`)
- `direction` (`1`)
- `offset` (`2`)
- `thickness` (`1`)

`\upright` *arg* (markup)

Print *arg* upright.

This command is the opposite of `\italic`; it sets the `font-shape` property to `upright`.

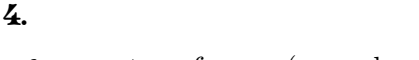
```
\markup {
  \italic {
    italic text
    \hspace #2
    \upright {
      upright text
    }
    \hspace #2
    italic again
  }
}
```



`\volta-number` *arg* (markup)

Set *arg* in a font appropriate for volta numbers.

```
\markup \volta-number "4."
```



`\with-string-transformer` *transformer* (procedure) *arg* (markup)

Apply string transformer function *transformer* to *arg*.

Whenever a string is interpreted inside *arg*, function *transformer* is called first, and its result is then interpreted. The arguments passed to *transformer* are the output definition, the property alist chain, and the markup *arg*. See Section “New markup command definition” in *Extending* about the two first arguments.

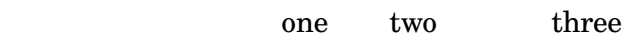
```
\markup \with-string-transformer
  #(lambda (layout props str)
     (string-upcase str))
  \concat { "abc" \larger "def" }
```



A.1.2 Markup for text alignment

`\abs-hspace` *amount* (number)
: Create an invisible object taking up absolute horizontal space of *amount* points.

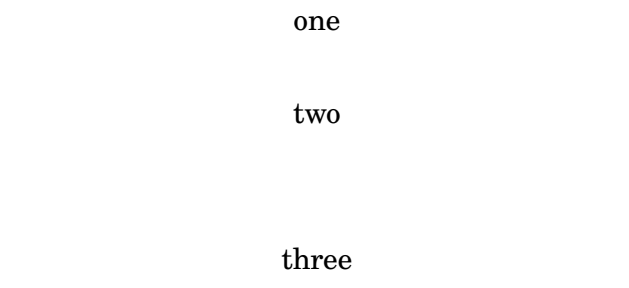
```
\markup {
  one
  \abs-hspace #20
  two
  \abs-hspace #40
  three
}
```



See also `\hspace`.

`\abs-vspace` *amount* (number)
: Create an invisible object taking up absolute vertical space of *amount* points.

```
\markup {
    \center-column {
    one
    \abs-vspace #20
    two
    \abs-vspace #40
    three
  }
}
```



See also `\vspace`.

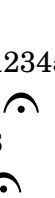
`\align-on-other` *axis* (non-negative, exact integer) *other-dir* (boolean-or-number) *other* (markup) *self-dir* (boolean-or-number) *self* (markup)
: Align markup *self* on markup *other* along *axis*.

  This function uses *self-dir* and *other-dir* for mutual alignment of *self* and *other*, respectively, translating *self* as requested relative to its surroundings. *other* is not printed.

  If *self-dir* or *other-dir* is `#f`, use the reference point of *self* or *other*, respectively.

```
\markup \column {
  12
  \align-on-other #X #RIGHT 12
                     #LEFT 12345
  \align-on-other #X #RIGHT 123
                     #LEFT \fermata
  123
  \align-on-other #X #RIGHT 123
```

```
                                ##f \fermata
}
```



`\center-align` *arg* (markup)
: Align *arg* to its X center.

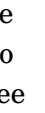
```
\markup {
  \column {
    one
    \center-align two
    three
  }
}
```



`\center-column` *args* (markup list)
: Put *args* into a centered column.

  See also `\column`.

```
\markup {
  \center-column {
    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* (to be more precise, the space between the baselines of the markups).

```
\markup {
  \column {
    one
    two
    three
  }
}
```

one
two
three

The baseline of the output of `\column` is the baseline of its first line.

Used properties:

- `baseline-skip`

`\combine` *arg1* (markup) *arg2* (markup)
: Print *arg1*, then print *arg2* on top of it.

  Note: `\combine` cannot take a list of markups enclosed in curly braces as an argument; for this purpose use `\overlay` instead.

```
\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 are concatenated on the input level, allowing ligatures. For example, `\concat { "f" "i" }` is equivalent to `"fi"`.

```
\markup {
  \concat {
    one two three
  }
}
```

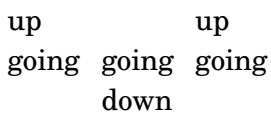
onetwothree

`\dir-column` *args* (markup list)
: Make a column of *args*.

  Depending on the setting of the `direction` layout property, the arguments are stacked upwards or downwards.

```
\markup {
  \override #`(direction . ,UP)
  \dir-column {
    going up
  }
  \hspace #1
  \dir-column {
    going down
  }
  \hspace #1
  \override #'(direction . 1)
  \dir-column {
    going up
  }
```

```
}
```



The baseline of the output of `\dir-column` is the baseline of its first line.

Used properties:

- `baseline-skip`
- `direction`

`\fill-line` *args* (markup list)

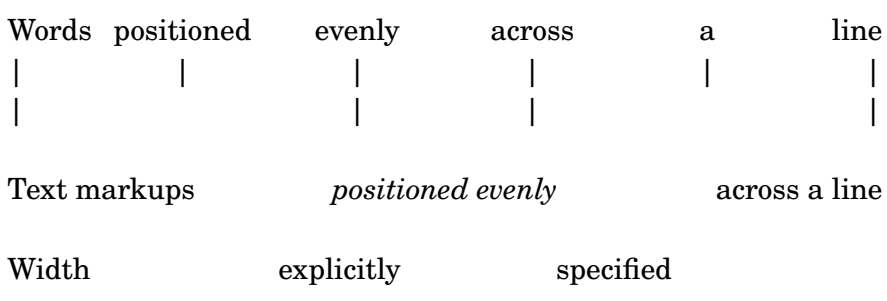
Put markups *args* into a horizontal line at fixed positions.

If there is a single argument, it gets centered. If there are two arguments, they get aligned to the left and right, respectively. Otherwise, if there are *n* arguments, the markups are aligned to *n* equally spaced positions, with the first markup left-aligned, the last markup right-aligned, and the remaining markups centered at the respective positions. If there are no arguments, return an empty stencil.

The width of the horizontal line can be modified by overriding the `line-width` property. The space between arguments is at least as large as the value of the `word-space` property, at the cost of lengthening the line if necessary.

Empty arguments (like `""`) are ignored. If you want to skip one position but keep the others, use `\null`.

```
\markup {
  \column {
    \fill-line { Words positioned evenly across a line }
    \fill-line { | | | | | | }
    \fill-line { | \null | | \null | }
    \null
    \fill-line {
      \line { Text markups }
      \line { \italic { positioned evenly } }
      \line { across a line }
    }
    \null
    \override #'(line-width . 50)
    \fill-line { Width explicitly specified }
  }
}
```



See also `\justify-line`.

Used properties:

- `line-width` (`#f`)

- word-space (0.6)
- text-direction (1)

\fill-with-pattern *space* (number) *dir* (direction) *pattern* (markup) *left* (markup) *right* (markup)

Put *left* and *right* at the start and end of a line, respectively, and fill the space in between with repeated *pattern* markups.

Patterns are spaced apart by *space* and aligned to direction *dir*. The width of the line is given by the line-width property.

```
\markup \column {
  "right-aligned:"
  \fill-with-pattern #1 #RIGHT . first right
  \fill-with-pattern #1 #RIGHT . second right
  \null
  "center-aligned:"
  \fill-with-pattern #1.5 #CENTER - left right
  \null
  "left-aligned:"
  \override #'(line-width . 50) {
    \fill-with-pattern #2 #LEFT : left first
    \fill-with-pattern #2 #LEFT : left second
  }
}
```

right-aligned:
first . . . . . right
second . . . . . right

center-aligned:
left - right

left-aligned:
left : : : : : : : : : : : : : : : : first
left : : : : : : : : : : : : : : : : second

Used properties:

- line-width
- word-space

\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 two three

one three
two

\halign *dir* (number) *arg* (markup)
: Print *arg* with horizontal alignment set to *dir*.

  If *dir* is -1, *arg* is left-aligned, while +1 makes it right-aligned. Values in between interpolate the 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

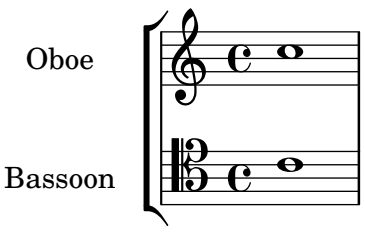
one
two
three

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
  }
>>
```



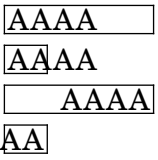
\hspace *amount* (number)
Create an invisible object taking up *amount* horizontal space.

```
\markup {
  one
  \hspace #2
  two
  \hspace #8
  three
}
```

one two three

amount can be also a negative value, which can be best visualized as if the current drawing point gets moved to the left.

```
\markup \concat {
  \hspace #4
  \column {
    \box \concat { AAAA \hspace #4 }
    \box \concat { AAAA \hspace #-4 }
    \box \concat { \hspace #4 AAAA }
    \box \concat { \hspace #-4 AAAA }
  }
}
```



See also `\abs-hspace`.

`\justify` *args* (markup list)

Print *args* as lines aligned both at the left and the right.

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.

The baseline of the output of `\justify` is the baseline of its first line.

Used properties:

- `text-direction` (`1`)
- `word-space`
- `line-width` (`#f`)
- `baseline-skip`

`\justify-field` *symbol* (symbol)

Justify the data that 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-line` *args* (markup list)

Put markups *args* into a horizontal line, equally spaced.

If there is a single argument, it gets centered. If there are two arguments, they get aligned to the left and right, respectively. Otherwise, the markups are spread to fill the entire line, separated by equally large spaces. If there are no arguments, return an empty stencil.

The width of the horizontal line can be modified by overriding the `line-width` property. The space between arguments is at least as large as the value of the `word-space` property, at the cost of lengthening the line if necessary.

```
\markup {
  \justify-line {
    Constant space between neighboring words
  }
}
```

Constant space between neighboring words

See also `\fill-line`.

Used properties:

- `line-width` (`#f`)

- word-space (0.6)
- text-direction (1)

\justify-string *arg* (string)

Print string *arg* as lines aligned both at the left and the right.

Paragraphs are indicated by double newlines. Use \override #'(line-width . *X*) to set the line width; *X* is the number of staff spaces.

```
\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

The baseline of the output of \justify-string is the baseline of its first line.

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* into a left-aligned column.

```
\markup {
  \left-column {
    one
    two
    three
  }
}
```

one
two
three

Used properties:

- `baseline-skip`

`\line` *args* (markup list)
: Put *args* into a horizontal line.

The property `word-space` determines the space between markups in *args*. For right-to-left scripts like Hebrew, `text-direction` should be set to -1.

```
\markup
  \override #'(word-space . 3)
  \column {
    \line { "A B" "C D" "E F" }
    \override #'(text-direction . -1)
    \line { "A B" "C D" "E F" }
  }
```

A B C D E F
E F C D A B

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`.

The argument to `\lower` is the vertical displacement amount, measured in (global) staff spaces, which is independent of the markup's current font size. If you need vertical movement that takes the font size into account, use `\translate-scaled` instead.

This function is normally used to move one element inside of a markup relative to the other elements. When using it on the whole markup, bear in mind that spacing mechanisms that place the markup itself on the page could cancel this shift. Consider using grob properties such as `padding`, `Y-offset`, or `extra-offset`, or spacing variables such as `markup-system-spacing`.

```
\markup {
  one
  \lower #3 two
  three
}
```

one three
two

\overlay *args* (markup list)
Take a list of markups *args* and combine them.

```
\markup {
  \fontsize #5
  \override #'(thickness . 2)
  \overlay {
    \draw-line #'(0 . 4)
    \arrow-head #Y #DOWN ##f
    \translate #'(0 . 4) \arrow-head #Y #UP ##f
  }
}
```



\pad-around *amount* (number) *arg* (markup)
Add padding *amount* all around *arg*.

Identical to function \pad-markup.

```
\markup {
  \box {
    default
  }
  \hspace #2
  \box {
    \pad-around #0.5 {
      padded
    }
  }
}
```

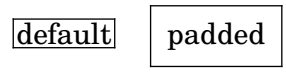
default padded

\pad-markup *amount* (number) *arg* (markup)
Add padding *amount* all around *arg*.

Identical to function \pad-around.

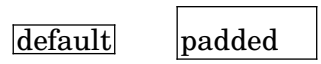
```
\markup {
  \box {
    default
  }
  \hspace #2
  \box {
    \pad-markup #1 {
      padded
```

```
    }
  }
}
```



\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
    }
  }
}
```



\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
    }
  }
}
```



\pad-x-left *amount* (number) *arg* (markup)
Add padding *amount* to the left of *arg* in the X direction.

```
\markup {
  \box {
    default
  }
  \hspace #4
  \box {
    \pad-x-left #2 {
      padded
    }
  }
}
```



\pad-x-right *amount* (number) *arg* (markup)
: Add padding *amount* to the right of *arg* in the X direction.

```
\markup {
  \box {
    default
  }
  \hspace #4
  \box {
    \pad-x-right #2 {
      padded
    }
  }
}
```



\put-adjacent *axis* (integer) *dir* (direction) *arg1* (markup) *arg2* (markup)
: Put *arg2* next to *arg1* along *axis* to the *dir* side, without moving *arg1*.

```
\markup \column {
  text
  \put-adjacent #X #LEFT text *
  text
}
```



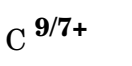
\raise *amount* (number) *arg* (markup)
: Raise *arg* by the distance *amount*.

  A negative *amount* indicates lowering, see also \lower.

  The argument to \raise is the vertical displacement amount, measured in (global) staff spaces, which is independent of the markup's current font size. If you need vertical movement that takes the font size into account, use \translate-scaled instead.

  This function is normally used to move one element inside of a markup relative to the other elements. When using it on the whole markup, bear in mind that spacing mechanisms that place the markup itself on the page could cancel this shift. Consider using grob properties such as padding, Y-offset, or extra-offset, or spacing variables such as markup-system-spacing.

```
\markup { C \small \bold \raise #1.0 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* into a right-aligned column.

```
\markup {
  \right-column {
    one
    two
    three
  }
}
```

one
two
three

Used properties:

- `baseline-skip`

\rotate *ang* (number) *arg* (markup)
: Rotate *arg* by *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 axes. See also `\translate-scaled`.

This function is normally used to move one element inside of a markup relative to the other elements. When using it on the whole markup, bear in mind that spacing mechanisms that place the markup itself on the page could cancel this shift. Consider using grob properties such as `padding`, `X-offset`, `Y-offset` or `extra-offset`, or spacing variables such as `markup-system-spacing`.

```
\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.

  This function is normally used to move one element inside of a markup relative to the other elements. When using it on the whole markup, bear in mind that spacing mechanisms that place the markup itself on the page could cancel this shift. Consider using grob properties such as `padding`, `X-offset`, `Y-offset` or `extra-offset`, or spacing variables such as `markup-system-spacing`.

  See also \translate.

```
\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 #1
    two
    \vspace #3
    three
  }
}
```

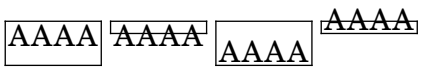
one

two

three

amount can be also a negative value, which can be best visualized as if the current drawing point gets moved up.

```
\markup {
  \vspace #1
  \box \column { AAAA \vspace #0.4 }
  \box \column { AAAA \vspace #-0.4 }
  \box \column { \vspace #0.4 AAAA }
  \box \column { \vspace #-0.4 AAAA }
}
```



See also `\abs-vspace`.

`\wordwrap` *args* (markup list)
Print *args* as left-aligned lines.

This function provides simple word-wrap. Use `\override #'(line-width . X)` to set the line width; *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.

The baseline of the output of `\wordwrap` is the baseline of its first line.

Used properties:

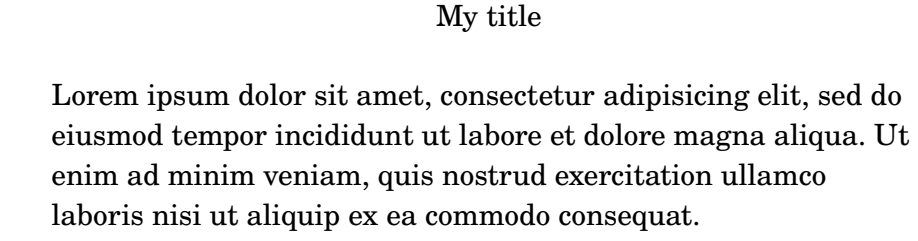
- `text-direction` (`1`)
- `word-space`
- `line-width` (`#f`)
- `baseline-skip`

`\wordwrap-field` *symbol* (symbol)
Word-wrap the data that 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
}
```



`\wordwrap-string` *arg* (string)

Print string *arg* as left-aligned lines.

Paragraphs are indicated by double newlines. Use `\override #'(line-width . X)` to set the line width; *X* is the number of staff spaces.

```
\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

The baseline of the output of `\wordwrap-string` is the baseline of its first line.

Used properties:

- `text-direction` (1)
- `word-space`
- `line-width`
- `baseline-skip`

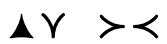
A.1.3 Graphical markup

`\arrow-head` *axis* (integer) *dir* (direction) *filled* (boolean)

Print an arrow head along *axis* in direction *dir*.

Fill the head if *filled* is set to `#t`.

```
\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
    }
  }
}
```



`\beam` *width* (number) *slope* (number) *thickness* (number)

Draw a beam with given *width*, *slope*, and *thickness*.

```
\markup {
  \beam #5 #1 #2
}
```



`\bracket` *arg* (markup)

Draw vertical brackets around *arg*.

```
\markup {
  \bracket {
    \note {2.} #UP
  }
}
```

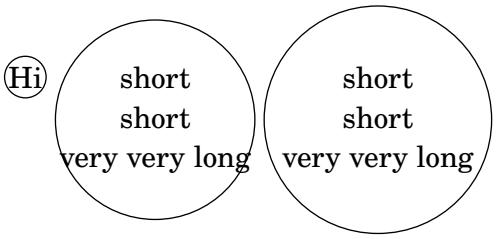


\circle *arg* (markup)

Draw a circle around *arg*.

Use properties `thickness`, `circle-padding`, and `font-size` to set the line thickness and padding around the markup. If `bbox` is set to `#t`, make the circle enclose the bounding box of *arg*, otherwise use either the width or the height of *arg* (whatever is larger) as the diameter.

```
\markup {
  \circle {
    Hi
  }
  \circle {
    \center-column { "short" "short" "very very long" }
  }
  \override #'(bbox . #t) \circle {
    \center-column { "short" "short" "very very long" }
  }
}
```



Note that the circle does not horizontally displace its argument. Use markup commands like `\left-align` or `\table` to make LilyPond realign it.

Used properties:

- `bbox` (`#f`)
- `circle-padding` (`0.2`)
- `font-size` (`0`)
- `thickness` (`1`)

\draw-circle *radius* (number) *thickness* (number) *filled* (boolean)

Draw a circle with given *radius* and *thickness*.

Fill the circle if *filled* is set to `#t`.

```
\markup {
  \draw-circle #2 #0.5 ##f
  \hspace #2
  \draw-circle #2 #0 ##t
}
```



`\draw-dashed-line` *dest* (pair of numbers)

Draw a dashed line along vector *dest*.

Properties `on` and `off` give the length of a dash and the space between two dashes, respectively; `phase` shortens the first dash by the given amount.

If the `full-length` property is set to `#t` (which is the default), the value of property `off` (and `on` under some boundary conditions) gets adjusted so that there is neither whitespace at the end of the line nor the last dash truncated.

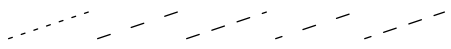
```
\markup {
  \override #'((on . 0.3) (off . 0.5))
  \draw-dashed-line #'(6 . 2)

  \draw-dashed-line #'(6 . 2)

  \override #'(full-length . #f)
  \draw-dashed-line #'(6 . 2)

  \override #'(phase . 0.5)
  \draw-dashed-line #'(6 . 2)

  \override #'((full-length . #f) (phase . 0.5))
  \draw-dashed-line #'(6 . 2)
}
```



Used properties:

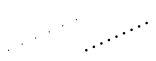
- `full-length` (`#t`)
- `phase` (`0`)
- `off` (`1`)
- `on` (`1`)
- `thickness` (`1`)

`\draw-dotted-line` *dest* (pair of numbers)

Draw a dotted line along vector *dest*.

Property `off` gives the space between two dots; its value gets adjusted so that the first and last dot exactly start and end the line, respectively. `phase` shifts all dots along the vector by the given amount.

```
\markup {
  \draw-dotted-line #'(5.1 . 2.3)
  \override #'((thickness . 2) (off . 0.2))
  \draw-dotted-line #'(5.1 . 2.3)
}
```



Used properties:

- `phase` (`0`)
- `off` (`1`)
- `thickness` (`1`)

`\draw-hline`

Draw a horizontal line.

The property `span-factor` sets the length of the line as a multiple of the `line-width` property.

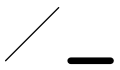
```
\markup {
  \column {
    \draw-hline
    \override #'(span-factor . 1/3)
    \draw-hline
  }
}
```

Used properties:

- `span-factor` (1)
- `line-width`
- `thickness` (1)

`\draw-line` *dest* (pair of numbers)
Draw a line along vector *dest*.

```
\markup {
  \draw-line #'(4 . 4)
  \override #'(thickness . 5)
  \draw-line #'(-3 . 0)
}
```



Used properties:

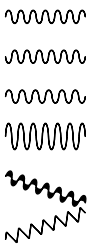
- `thickness` (1)

`\draw-squiggle-line` *sq-length* (number) *dest* (pair of numbers) *eq-end?* (boolean)
Draw a squiggled line along vector *dest*.

sq-length is the length of the first bow; this value gets always adjusted so that an integer number of squiggles is printed. If *eq-end?* is set to `#t`, the squiggled line ends with a bow in the same direction as the starting one.

The appearance of the squiggled line may be customized by overriding the `thickness`, `angularity`, `height`, and `orientation` properties.

```
\markup
  \column {
    \draw-squiggle-line #0.5 #'(6 . 0) ##t
    \override #'(orientation . -1)
    \draw-squiggle-line #0.5 #'(6 . 0) ##t
    \draw-squiggle-line #0.5 #'(6 . 0) ##f
    \override #'(height . 1)
    \draw-squiggle-line #0.5 #'(6 . 0) ##t
    \override #'(thickness . 5)
    \draw-squiggle-line #0.5 #'(6 . -2) ##t
    \override #'(angularity . 2)
    \draw-squiggle-line #0.5 #'(6 . 2) ##t
  }
```



Used properties:

- `orientation` (`1`)
- `height` (`0.5`)
- `angularity` (`0`)
- `thickness` (`0.5`)

`\ellipse` *arg* (markup)

Draw an ellipse around *arg*.

Use properties `thickness`, `x-padding`, `y-padding`, and `font-size` to set the line thickness and padding around the markup.

This is the same as function `\oval` but with different padding defaults.

```
\markup {
  \ellipse {
    Hi
  }
}
```



Note that the ellipse does not horizontally displace its argument. Use markup commands like `\left-align` or `\table` to make LilyPond realign it.

Used properties:

- `y-padding` (`0.2`)
- `x-padding` (`0.2`)
- `font-size` (`0`)
- `thickness` (`1`)

`\epsfile` *axis* (number) *size* (number) *file-name* (string)

Inline an image *file-name*, scaled along *axis* to *size*.

See `\image` for details on this command; calling

```
\markup \epsfile axis size file-name
```

is the same as

```
\markup
  \override #'(background-color . #f)
  \image axis size file-name
```

`\filled-box` *xext* (pair of numbers) *yext* (pair of numbers) *blot* (number)

Draw a box of dimensions *xext* and *yext*, with rounded corners given by *blot*.

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
  \combine
    \filled-box #'(1 . 8) #'(0 . 7) #0.2
    \with-color #white
      \filled-box #'(3.6 . 5.6) #'(3.5 . 5.5) #0.7
}
```



\hbracket *arg* (markup)
: Draw horizontal brackets around *arg*.

```
\markup {
  \hbracket {
    \line {
      one two three
    }
  }
}
```

one two three

\image *axis* (number) *size* (number) *file-name* (string)
: Inline an image *file-name*, scaled along *axis* to *size*.

  The image format is determined based on the extension of *file-name*, which should be `.png` for a PNG image, or `.eps` for an EPS image (`.PNG` and `.EPS` are allowed as well).

  EPS files are only supported in the PostScript backend – for all output formats –, and in the Cairo backend – when creating PostScript or EPS output.

  If the image has transparency, it will appear over a colored background with the color specified by the `background-color` property, which defaults to `"white"`.

  To disable the colored background, set `background-color` to `#f`. For EPS images, this always works (where EPS images work in the first place). On the other hand, for PNG images, it works in the Cairo backend (which can output all supported formats), as well as in the SVG backend, but *not* in the PostScript backend, which is the default. See Section "Advanced command-line options for LilyPond" in *Application Usage* for how to activate the Cairo backend.

  Use `\withRelativeDir` as a prefix to *file-name* if the file should be found relative to the input file.

  Used properties:

  - `background-color` (`"white"`)

\oval *arg* (markup)
: Draw an oval around *arg*.

  Use properties `thickness`, `x-padding`, `y-padding`, and `font-size` to set the line thickness and padding around the markup.

  This is the same as function `\ellipse` but with different padding defaults.

```
\markup {
  \oval {
    Hi
  }
}
```



Note that the oval does not horizontally displace its argument. Use markup commands like `\left-align` or `\table` to make LilyPond realign it.

Used properties:

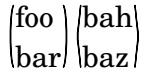
- `y-padding` (`0.75`)
- `x-padding` (`0.75`)
- `font-size` (`0`)
- `thickness` (`1`)

`\parenthesize` *arg* (markup)

Draw parentheses around *arg*.

This is useful for parenthesizing a column containing several lines of text.

```
\markup {
  \parenthesize
  \column {
    foo
    bar
  }
  \override #'(angularity . 2)
  \parenthesize
  \column {
    bah
    baz
  }
}
```



Used properties:

- `width` (`0.25`)
- `line-thickness` (`0.1`)
- `thickness` (`1`)
- `size` (`1`)
- `padding`
- `angularity` (`0`)

`\path` *thickness* (number) *commands* (list)

Draw a path with line *thickness* according to the directions given in *commands*.

commands is a list of lists where the `car` of each sublist is a drawing command and the `cdr` comprises the associated arguments for each command.

There are seven commands available to use in *commands*: `moveto`, `rmoveto`, `lineto`, `rlineto`, `curveto`, `rcurveto`, and `closepath`. Note that the commands that begin

with 'r' are the relative variants of the other three commands. You may also use the standard SVG single-letter equivalents: `moveto` = `M`, `lineto` = `L`, `curveto` = `C`, `closepath` = `Z`. The relative commands are written lowercase: `rmoveto` = `r`, `rlineto` = `l`, `rcurveto` = `c`.

The commands `moveto`, `rmoveto`, `lineto`, and `rlineto` take 2 arguments, namely the X and Y coordinates of the destination point.

The commands `curveto` and `rcurveto` create cubic Bézier curves, and take 6 arguments; the first two are the X and Y coordinates for the first control point, the second two are the X and Y coordinates for the second control point, and the last two are the X and Y coordinates for the destination point.

The `closepath` command takes zero arguments and closes the current subpath in the active path.

Line-cap styles and line-join styles may be customized by overriding the `line-cap-style` and `line-join-style` properties, respectively. Available line-cap styles are `butt`, `round`, and `square`. Available line-join styles are `miter`, `round`, and `bevel`.

The property `filled` specifies whether or not the path is filled with color.

```
samplePath =
  #'((lineto -1 1)
     (lineto 1 1)
     (lineto 1 -1)
     (curveto -5 -5 -5 5 -1 0)
     (closepath))

\markup \scale #'(2 . 2) {
  \path #0.25 #samplePath

  \override #'(line-join-style . miter)
  \path #0.25 #samplePath

  \override #'(filled . #t)
  \path #0.25 #samplePath
}
```



Used properties:

- `filled` (`#f`)
- `line-join-style` (`round`)
- `line-cap-style` (`round`)

`\polygon` *points* (list of number pairs)

A polygon delimited by the list of *points*.

Property `extroversion` defines how the shape of the polygon is adapted to its thickness: if it is 0, the polygon is traced as-is. If it is -1, the outer side of the line is just on the given points. If set to 1, the line has its inner side on the points. The `thickness` property controls the thickness of the line; for filled polygons, this means the diameter of the blot.

```
regularPentagon =
  #'((1 . 0) (0.31 . 0.95) (-0.81 . 0.59)
     (-0.81 . -0.59) (0.31 . -0.95))

\markup \scale #'(2 . 2) {
  \polygon #'((-1 . -1) (0 . -3) (2 . 2) (1 . 2))
  \override #'(filled . #f)
    \override #'(thickness . 2)
      \combine
        \with-color #(universal-color 'blue)
          \polygon #regularPentagon
        \with-color #(universal-color 'vermillion)
          \override #'(extroversion . 1)
            \polygon #regularPentagon
}
```



Used properties:

- `thickness` (1)
- `filled` (#t)
- `extroversion` (0)

`\postscript` *str* (string)

Insert *str* directly into the output as a PostScript command string.

This command is meant as a *last resort*. Almost all needs are better fulfilled by other markup commands (see, for example, `\path` and `\draw-line`). If you do use this command, keep the following points in mind:

- `\postscript` does not work in SVG output.
- Only a subset of the PostScript language is supported during the conversion from PostScript to PDF.
- There are no stability guarantees on the details of how LilyPond produces its own output (i.e., the context into which the PostScript code is inserted). They may change substantially across versions.
- LilyPond cannot understand the shape of the drawing, leading to suboptimal spacing. Usually, it is necessary to explicitly set up dimensions with a command like `\with-dimensions`.
- Depending on how you install LilyPond, the version of the PostScript interpreter (Ghostscript) can vary, and some of its features may be disabled.

str is processed with the following constraints.

- The string is embedded into the (intermediate) output file with the PostScript commands

  ```
  gsave currentpoint translate 0.1 setlinewidth
  ```

  before and

  ```
  grestore
  ```
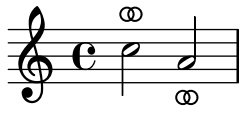
  after it.

- After these preceding commands (i.e., `currentpoint translate`) the origin of the current transformation is the reference point of `\postscript`. Scale and rotation of the current transformation reflect the global staff line distance and (if applied) other transformation markup commands (e.g., `\scale` and `\rotate`) encapsulating the `\postscript` command.
- The current point is set to the coordinate (0, 0).
- If an unwanted line appears at the beginning of your PostScript code, you are probably missing a call to `newpath`.

```
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
}
```
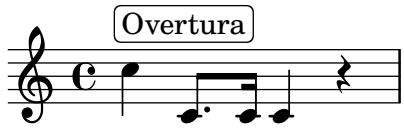


`\rounded-box` *arg* (markup)

Draw a box with rounded corners around *arg*.

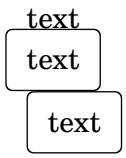
This function looks at properties `thickness`, `box-padding`, and `font-size` to determine the line thickness and padding around the *arg*. The `corner-radius` property defines the radius of the round corners (default value is 1).

```
c4^\markup {
  \rounded-box {
    Overtura
  }
}
c,8. c16 c4 r
```



Note that the box does not horizontally displace its argument. Use markup commands like `\left-align` or `\table` to make LilyPond realign it.

```
\markup {
  \override #'(box-padding . 1.5)
  \column {
    "text"
    \rounded-box "text"
    \left-align \rounded-box "text"
  }
}
```



Used properties:

- `box-padding` (`0.5`)
- `font-size` (`0`)
- `corner-radius` (`1`)
- `thickness` (`1`)

`\scale` *factor-pair* (pair of numbers) *arg* (markup)

Scale *arg*.

factor-pair is a pair of numbers representing the scaling factor of the X and Y axes. Negative values may be used to produce mirror images.

```
\markup {
  \line {
    \scale #'(2 . 1)
    stretched
    \scale #'(1 . -1)
    mirrored
  }
}
```



`\triangle` *filled* (boolean)

Draw a triangle.

Fill the triangle if *filled* is set to `#t`.

The appearance can be controlled with properties `extroversion`, `font-size`, and `thickness`.

```
\markup {
  \triangle ##t
  \triangle ##f
  \override #'(font-size . 5)
  \override #'(thickness . 5) {
    \override #'(extroversion . 1)
    \triangle ##f
    \override #'(extroversion . -1)
    \triangle ##f
  }
}
```



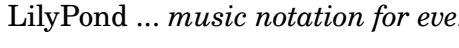
Used properties:

- `thickness` (1)
- `font-size` (0)
- `extroversion` (0)

`\with-url` *url* (string) *arg* (markup)
: Add a link to URL *url* around *arg*.

  This only works in the PDF backend.[2]

```
\markup {
  \with-url "https://lilypond.org/" {
    LilyPond ... \italic {
      music notation for everyone
    }
  }
}
```

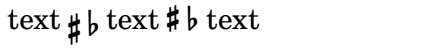


A.1.4 Markup for music and musical symbols

`\accidental` *alteration* (an exact rational number)
: Select an accidental glyph for *alteration*, given as a rational number.

  Use `\text-accidental` instead if you need glyph representation forms that fit and align well with text.

```
\markup {
  text
  \tiny { \accidental #1/2 \accidental #-1/2 }
  text
  \tiny { \text-accidental #1/2 \text-accidental #-1/2 }
  text
}
```



  Used properties:

  - `alteration-glyph-name-alist`

`\bar-line` *strg* (string)
: Print a bar line in markup.

  The allowed characters for input string *strg* are ‘`;|.:!S[]{}`’, having the same meaning as with the `\bar` command. The additional characters ‘`{`’ and ‘`}`’ denote left and right braces, respectively.

  The output is vertically centered.

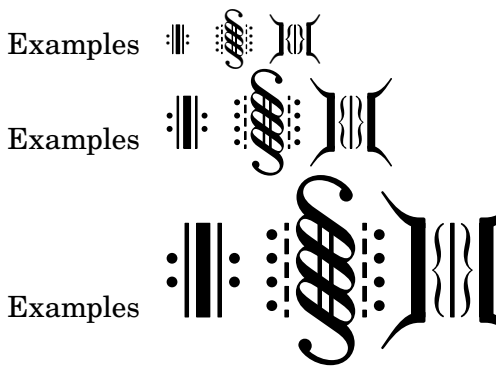
  Changes of `font-size` are respected.

  The default of `height` is 4 staff-space units. Apart from the bracket tips of a bracket bar line and the segno bar line all other bar lines are scaled with `height`. We don’t scale bracket tips and segno to meet the behavior of `SystemStartBracket` and the segno bar line.

[2] Due to technical limitations the link doesn’t work here in the Notation Reference.

\bar-line is further customizable by overriding dot-count and dash-count for dotted and dashed bar lines. The values for hair-thickness, kern and thick-thickness are customizable as well; defaults are the same as the values of the corresponding BarLine grob properties.

```
\markup {
   \override #'(word-space . 2)
   \column {
      \line {
        Examples
        \fontsize #-5 \translate-scaled #'(0 . 2) {
          \bar-line ":|.|:"
          \bar-line ";!S!;"
          \bar-line "]{|}["
        }
      }
      \line {
        Examples
        \fontsize #0 \translate-scaled #'(0 . 2) {
          \bar-line ":|.|:"
          \bar-line ";!S!;"
          \bar-line "]{|}["
        }
      }
      \line {
        Examples
        \fontsize #5 \translate-scaled #'(0 . 2) {
          \bar-line ":|.|:"
          \bar-line ";!S!;"
          \bar-line "]{|}["
        }
      }
   }
}
```



Used properties:

- thick-thickness (6.0)
- kern (3.0)
- hair-thickness (1.9)
- dash-count (5)
- dot-count (4)

- `height` (4)
- `font-size` (0)

`\coda` Draw a coda sign.

```
\markup {
  \coda
}
```

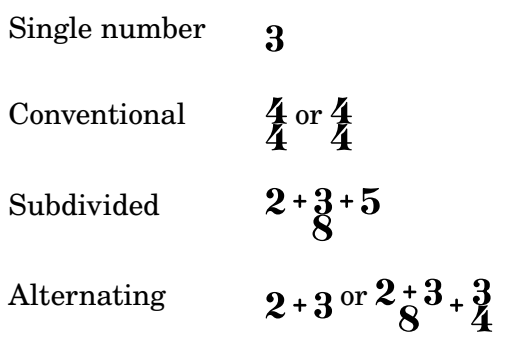


`\compound-meter` *time-sig* (number or pair)

Draw a numeric time signature based on *time-sig*.

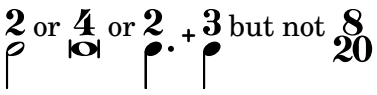
time-sig can be a single number, a pair of numbers, a simple list, or a list of lists, as the following example demonstrates.

```
\markuplist {
  \override #'(baseline-skip . 4.5)
  \override #'(padding . 4.5)
  \table #'(-1 -1) {
    "Single number"  \compound-meter #3
    "Conventional"   \line {
                       \compound-meter #'(4 . 4) or
                       \compound-meter #'(4 4)
                     }
    "Subdivided"     \compound-meter #'(2 3 5 8)
    "Alternating"    \line {
                       \compound-meter #'((2) (3)) or
                       \compound-meter #'((2 3 8) (3 4))
                     }
  }
}
```



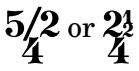
Setting the `denominator-style` property to `note` prints denominators as a note and dots when exact representation is possible. Example:

```
\markup {
  \override #'(denominator-style . note)
  \line {
    \compound-meter #'(2 2) or
    \compound-meter #'(4 1/2) or
    \compound-meter #'((2 8/3) (3 4)) but not
    \compound-meter #'(8 20)
  }
}
```



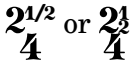
The nested-fraction-mixed property controls whether fractional parts are printed as mixed numbers or as common fractions. Example:

```
\markup {
  \override #'(nested-fraction-mixed . #f)
  \compound-meter #'(5/2 4) or
  \override #'(nested-fraction-mixed . #t)
  \compound-meter #'(5/2 4)
}
```



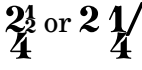
The nested-fraction-orientation property controls how nested fractions are arranged. Supported values are horizontal and vertical. Example:

```
\markup {
  \override #'(nested-fraction-orientation . horizontal)
  \compound-meter #'(5/2 4) or
  \override #'(nested-fraction-orientation . vertical)
  \compound-meter #'(5/2 4)
}
```



The nested-fraction-relative-font-size property controls the size of the numerals in nested fractions. Recommended values are -5.5 and 0. Using large numerals may take precedence over related properties. Example:

```
\markup {
  \override #'(nested-fraction-relative-font-size . -5.5)
  \compound-meter #'(5/2 4) or
  \override #'(nested-fraction-relative-font-size . 0)
  \compound-meter #'(5/2 4)
}
```



Used properties:

- note-staff-position (-2)
- note-head-style (())
- note-flag-style (())
- note-dots-direction (0)
- nested-fraction-relative-font-size (())
- nested-fraction-orientation (default)
- nested-fraction-mixed (#t)
- font-size (0)
- denominator-style (default)

\customTabClef *num-strings* (integer) *staff-space* (number)

Draw a clef in sans-serif style for a tablature with *num-strings* lines spaced by *staff-space*.

This markup command is used to implement \clef moderntab within a TabStaff context.

```
\markup {
  \customTabClef #4 #1
}
```



\doubleflat

Draw a double flat symbol.

```
\markup {
  \doubleflat
}
```



\doublesharp

Draw a double sharp symbol.

```
\markup {
  \doublesharp
}
```



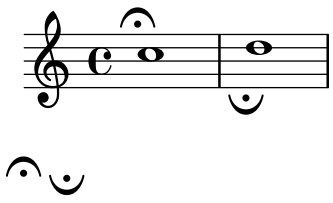
\fermata Create a fermata glyph.

If property direction is DOWN, use an inverted glyph.

Note that within music, one would normally use the \fermata articulation instead of a markup.

```
{ c''1^\markup \fermata d''1_\markup \fermata }

\markup { \fermata \override #`(direction . ,DOWN) \fermata }
```



Used properties:

- direction (1)

\flat Draw a flat symbol.

```
\markup {
  \flat
}
```

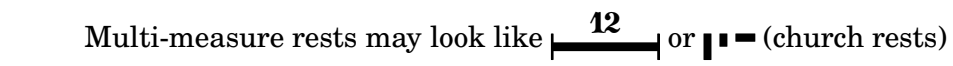


\multi-measure-rest-by-number *length* (non-negative, exact integer)

Return a multi-measure rest symbol for *length* measures.

If the number of measures is greater than the number given by `expand-limit` a thick horizontal line is printed. For every multi-measure rest lasting more than one measure a number is printed on top. However, if property `multi-measure-rest-number` is set to `#t`, this number gets suppressed.

```
\markup {
  Multi-measure rests may look like
  \multi-measure-rest-by-number #12
  or
  \override #'(multi-measure-rest-number . #f)
    \multi-measure-rest-by-number #7
  (church rests)
}
```



Used properties:

- `multi-measure-rest-number` (`#t`)
- `width` (`8`)
- `expand-limit` (`10`)
- `hair-thickness` (`2.0`)
- `thick-thickness` (`6.6`)
- `word-space`
- `style` (`()`)
- `font-size` (`0`)

\musicglyph *glyph-name* (string)

Print music symbol *glyph-name*.

See Section "The Emmentaler font" in *Notation Reference* for a complete listing of the possible glyph names.

```
\markup {
  \musicglyph "f"
  \musicglyph "rests.2"
  \musicglyph "clefs.G_change"
}
```



\natural Draw a natural symbol.

```
\markup {
  \natural
}
```



\note *duration* (duration) *dir* (number)

Draw a note with a given duration and a stem.

The note duration is specified with *duration* (setting both the note head type and the number of augmentation dots). The stem direction and length is given by *dir*.

This function is wrapper around \note-by-number; its documentation gives more details on the available properties.

```
\markup {
  \note {4..} #UP
  \hspace #2
  \override #'(style . cross)
    \note {4..} #0.75
  \hspace #2
  \note {\breve} #0
}
```



Used properties:

- style (default)
- dots-direction (0)
- flag-style (())
- font-size (0)

\note-by-number *log* (number) *dot-count* (number) *dir* (number)

Draw a note with a given length, a number of dots, and a stem.

The note length is specified with *log*, the number of dots with *dot-count*, and the stem direction and length with *dir*. Fractional values for *dir* can be used to control the length of the stem. Value 0 suppresses the stem completely; this is useful for note head styles that don't take stems or come with built-in stems (like Kievan).

Ancient note head styles (via the style property, see Section B.9 [Note head styles], page 922) get mensural-style flags by default; use the flag-style property to override this. Supported flag styles are default, old-straight-flag, modern-straight-flag, flat-flag, stacked, mensural, and neomensural. The last flag style is the same as mensural and provided for convenience.

```
\markup {
  \note-by-number #3 #0 #DOWN
  \hspace #2
  \note-by-number #1 #2 #0.8
  \hspace #2
  \override #'(style . petrucci)
    \note-by-number #3 #0 #UP
  \hspace #2
  \override #'(flag-style . modern-straight-flag)
    \note-by-number #4 #0 #DOWN
  \override #'(style . kievan)
```

```
    \note-by-number #2 #0 #0
}
```



See also function \note.

Used properties:

- style (default)
- dots-direction (0)
- flag-style (())
- font-size (0)

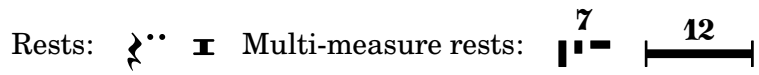
\rest *duration* (duration)

Return a rest symbol with length *duration*.

If the multi-measure-rest property is set to #t, a multi-measure rest symbol may be returned. In this case the duration needs to be entered as { 1*N* } to get a multi-measure rest for *N* bars. Actually, only the scaling factor (i.e., the number after '*') determines the length; the basic duration is disregarded.

See also functions \rest-by-number and \multi-measure-rest-by-number for more information on the used properties.

```
\markup {
  Rests:
  \hspace #2
  \rest { 4.. }
  \hspace #2
  \rest { \breve }
  \hspace #2
  Multi-measure rests:
  \override #'(multi-measure-rest . #t)
  {
    \hspace #2
    \rest { 1*7 }
    \hspace #2
    \rest { 1*12 }
  }
}
```



Used properties:

- multi-measure-rest-number (#t)
- width (8)
- expand-limit (10)
- hair-thickness (2.0)
- thick-thickness (6.6)
- word-space
- style (())
- font-size (0)

- `style` (())
- `ledgers` ((-1 0 1))
- `font-size` (0)

`\rest-by-number` *log* (integer) *dot-count* (integer)
Draw a rest of length *log*, with *dot-count* dots.

For duration logs that appear in the `ledgers` property, rest symbols with ledger lines are selected.

```
\markup {
  \rest-by-number #3 #2
  \hspace #2
  \rest-by-number #0 #1
  \hspace #2
  \rest-by-number #-1 #0
  \hspace #2
  \override #'(ledgers . ())
  \rest-by-number #-1 #0
}
```
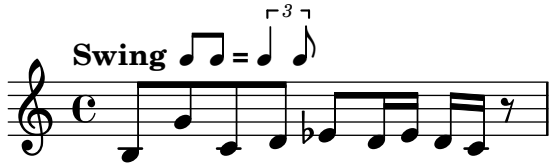


Used properties:

- `style` (())
- `ledgers` ((-1 0 1))
- `font-size` (0)

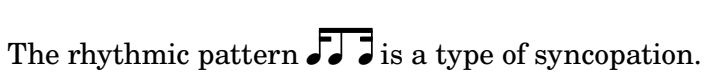
`\rhythm` *music* (music)
Draw embedded rhythmic pattern as specified by *music*.

```
\relative {
  \tempo \markup {
    Swing
    \hspace #0.4
    \rhythm { 8[ 8] } = \rhythm { \tuplet 3/2 { 4 8 } }
  }
  b8 g' c, d ees d16 ees d c r8
}
```



Within `\rhythm`, there is no time signature and no division in measures (as with `\cadenzaOn`, see Section 2.3.5 [Unmetered music], page 93). Beaming must be added explicitly with the syntax explained in Section 2.4.3 [Manual beams], page 115.

```
\markup {
  The rhythmic pattern \rhythm { 16[ 8 16] } is
  a type of syncopation.
}
```



\stemDown can be used to flip the stems.

```
\markup \rhythm { \stemDown 8 16 8 }
```



\rhythm works by creating a StandaloneRhythmVoice context, which is enclosed in a StandaloneRhythmStaff context, which is enclosed in a StandaloneRhythmScore context. It is possible to apply global tweaks to the output by using a \layout block.

```
\layout {
  \context {
    \StandaloneRhythmVoice
    \xNotesOn
  }
}

\markup \rhythm { 8 16 8 }
```



Used properties:

- font-size (-2)

\score *score* (score)
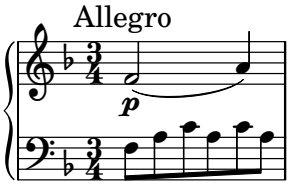
Inline an image of music as specified by *score*.

The reference point (usually the middle staff line) of the lowest staff in the top system is placed on the baseline.

No page breaks and no MIDI output, i.e., both \pageBreak commands and \midi{} blocks get ignored.

```
\markup {
  Text before the score.
  \score {
    \new PianoStaff <<
      \new Staff \relative c' {
        \key f \major
        \time 3/4
        \mark \markup { Allegro }
        f2\p( a4)
      }
      \new Staff \relative c {
        \clef bass
        \key f \major
        \time 3/4
        f8( a c a c a
      }
    >>

    \layout {
      indent = 0.0\cm
    }
  }
  Text after the score.
}
```



Text before the score. Text after the score.

Used properties:

- `baseline-skip`

`\segno` Draw a segno symbol.

```
\markup {
  \segno
}
```

𝄋

`\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
}
```



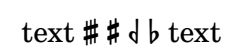
`\text-accidental` *alteration* (an exact rational number)

Get an accidental for *alteration* that aligns well with text.

alteration must be a rational number; the function uses the `alteration-glyph-name-alist` property to map this number to an accidental glyph.

If no text variant glyph for the accidental is found, markup is constructed to fittingly scale and position the (non-text) accidental glyph.

```
\markup {
  text
  \tiny { \text-accidental #3/4 \text-accidental #1/2
          \text-accidental #-1/4 \text-accidental #-1/2 }
  text
}
```



Used properties:

- `alteration-glyph-name-alist`

`\text-common-time`

Draw a common time symbol for text.

```
\markup {
  \text-common-time
}
```



`\text-cut-time`

Draw a cut time symbol for text.

```
\markup {
  \text-cut-time
}
```



`\text-doubleflat`

Draw a double flat symbol for text.

```
\markup {
  \text-doubleflat
}
```



`\text-doublesharp`

Draw a double sharp symbol for text.

```
\markup {
  \text-doublesharp
}
```



`\text-flat`

Draw a flat symbol for text.

```
\markup {
  \text-flat
}
```



`\text-natural`

Draw a natural symbol for text.

```
\markup {
  \text-natural
}
```



`\text-sharp`

Draw a sharp symbol for text.

```
\markup {
  \text-sharp
}
```



`\tied-lyric` *str* (string)

Replace '~' tilde symbols with tie characters in *str*.

```
\markup \column {
  \tied-lyric
    "Siam navi~all'onde~algenti Lasciate~in abbandono"
  \tied-lyric
    "Impetuosi venti I nostri~affetti sono"
  \tied-lyric
    "Ogni diletto~e scoglio Tutta la vita~e~un mar."
}
```

Siam navi all'onde algenti Lasciate in abbandono
Impetuosi venti I nostri affetti sono
Ogni diletto e scoglio Tutta la vita e un mar.

Used properties:

- `word-space`

`\varcoda` Draw a varcoda sign.

```
\markup {
  \varcoda
}
```



A.1.5 Conditional markup

`\if` *condition?* (procedure) *argument* (markup)

Test *condition?*, and only insert *argument* if it is true.

The condition is provided as a procedure taking an output definition and a property alist chain. The procedure is applied, and its result determines whether to print the markup. This command is most useful inside `oddHeaderMarkup` or similar. Here is an example printing page numbers in bold:

```
\paper {
  oddHeaderMarkup =
    \markup \fill-line {
      ""
      \if #print-page-number
           \bold \fromproperty #'page:page-number-string
    }
  evenHeaderMarkup =
    \markup \fill-line {
      \if #print-page-number
           \bold \fromproperty #'page:page-number-string
      ""
    }
}
```

`\unless` *condition?* (procedure) *argument* (markup)

Test *condition?*, and only insert *argument* if it is false.

This function is similar to `\if`; the following example shows how to print the copyright notice on all pages but the last instead of just the first page.

```
\paper {
  oddFooterMarkup = \markup {
    \unless #on-last-page-of-part \fill-line {
      \fromproperty #'header:copyright
    }
  }
}

\header {
  copyright = "© LilyPond Authors. License: GFDL."
  tagline = "© LilyPond Authors.  Documentation placed
under the GNU Free Documentation License
version 1.3."
}
```

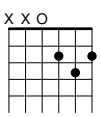
A.1.6 Instrument-specific markup

`\fret-diagram` *definition-string* (string)

Make a (guitar) fret diagram based on *definition-string*.

For example, say

```
\markup \fret-diagram "s:1.25;6-x;5-x;4-o;3-2;2-3;1-2;"
```



for fret spacing 5/4 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 (relative to normal line thickness). Default: 0.5.
  - `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.

For example,

```
\markup \fret-diagram-terse "x;x;o;2;3;2;"
```



displays 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:

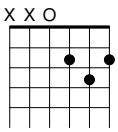
- `thickness` (`0.5`)
- `fret-diagram-details`
- `size` (`1.0`)
- `align-dir` (`-0.4`)

`\fret-diagram-verbose` *marking-list* (pair)

Make a fret diagram containing the symbols indicated in *marking-list*.

The following example produces a standard D chord diagram without fingering indications.

```
\markup \scale #'(1.5 . 1.5)
  \fret-diagram-verbose
    #'((mute 6) (mute 5) (open 4)
       (place-fret 3 2) (place-fret 2 3) (place-fret 1 2))
```



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.

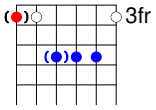
`(place-fret` *string-number fret-number* [*finger-value*] [*color-modifier*] [*color*] ['parenthesized ['default-paren-color]])

Place a fret playing indication on string *string-number* at fret *fret-number* with an optional fingering label *finger-value*, an optional color modifier *color-modifier*, an optional color *color*, an optional parenthesis `'parenthesized` and an optional parenthesis color `'default-paren-color`.

By default, the fret playing indicator is a solid dot. This can be globally changed by setting the value of the property `dot-color` or for a single dot by setting the value of *color*. The dot can be parenthesized by adding `'parenthesized`. By default the color for the parenthesis is taken from the dot. Adding `'default-paren-color` will take the parenthesis color from the global `dot-color` property; as a fallback black will be used. Setting *color-modifier* to `inverted` inverts the dot color for a specific fingering.

The values for *string-number*, *fret-number*, and the optional *finger* should be entered first in that order. The order of the other optional arguments does not matter. 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.

```
\markup \scale #'(1.5 . 1.5)
  \fret-diagram-verbose #'(
    (place-fret 6 3 1 red parenthesized default-paren-color)
    (place-fret 5 3 1 inverted)
    (place-fret 4 5 2 blue parenthesized)
    (place-fret 3 5 3 blue)
    (place-fret 2 5 4 blue)
    (place-fret 1 3 1 inverted)
  )
```



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 containing the symbols indicated in *definition-string*.

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)

```
\markup \harp-pedal "^-v|--ov^"
```



The function also checks whether the string has the typical form of three pedals, then the divider, and then the remaining four pedals, printing a warning otherwise (without preventing the non-standard order).

Use the `size` property to control the overall size, and the `thickness` property for the line thickness of the horizontal line and the divider.

The remaining configuration is done via the `harp-pedal-details` property; it contains the following elements:

`box-offset`

vertical shift of the box center for up/down pedals

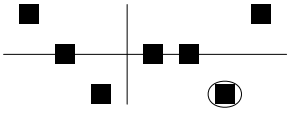
`box-width`
: box width

`box-height`
: box height

`space-before-divider`
: spacing between two boxes before the divider

`space-after-divider`
: spacing between two boxes after the divider

```
\markup {
  \override #'((size . 1.5)
               (harp-pedal-details . ((box-width . 1)
                                      (box-offset . 2))))
  \harp-pedal "^-v|--ov^"
}
```



For global modification of `harp-pedal-details`, i.e., outside of the current `\markup` block, you can also use code similar to

```
\override Voice.TextScript.harp-pedal-details.box-width = 1
```

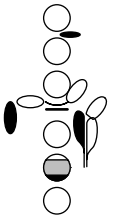
Used properties:

- `thickness` (`0.5`)
- `harp-pedal-details` (`()`)
- `size` (`1.2`)

`\woodwind-diagram` *instrument* (symbol) *user-draw-commands* (list)
: Make a woodwind-instrument diagram for *instrument* using *user-draw-commands*.

  user-draw-commands is a list of alists, specifying the left-hand keys, the elements on the central column, and the right-hand keys. For example, this diagram

```
\markup \woodwind-diagram
  #'oboe #'((lh . (d ees))
            (cc . (five3qT1q))
            (rh . (gis)))
```



shows 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 effectuating a trill between 1/4 and 3/4 is 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 call (`print-keys '`*instrument*) in your `.ly` file, where *instrument* is the instrument whose keys you want to print.

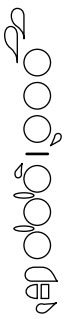
Certain keys allow for special configurations. The entire gamut of configurations possible is as follows:

`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 results in a diagram with all of the keys drawn but none filled, for example

```
\markup \woodwind-diagram #'flute #'()
```



Used properties:

- `woodwind-diagram-details` (())
- `font-size` (0)
- `graphical` (#t)
- `thickness` (0.1)
- `size` (1)

A.1.7 Accordion registers

`\discant` *name* (string)
: Generate a discant accordion register symbol for *name*.

To make it available,

```
#(use-modules (lily accreg))
```

is required near the top of your input file.

The register names in the default `\discant` register set have been modeled after the numeric Swiss notation (as depicted in `http://de.wikipedia.org/wiki/Register_%28Akkordeon%29`), omitting the slashes and dropping leading zeros.

The string *name* is basically a three-digit number with the lowest digit specifying the number of 16' reeds, the tens the number of 8' reeds, and the hundreds specifying the number of 4' reeds. Without modification, the specified number of reeds in 8' is centered in the symbol. Newer instruments may have registrations where 8' can be used either within or without a tone chamber, 'cassotto'. Notationally, the central dot then indicates use of cassotto. One can suffix the tens' digits '`1`' and '`2`' with '`+`' or '`-`' to indicate clustering the dots at the right or left, respectively, rather than centered.

Some examples are



```
\discant "1"
```



```
\discant "1+0"
```



```
\discant "120"
```



```
\discant "131"
```

Used properties:

- `font-size` (0)

`\freeBass` *name* (string)

Generate a free bass/converter accordion register symbol for the usual two-reed layout as given by *name*.

To make it available,

```
#(use-modules (lily accreg))
```

is required near the top of your input file.

Available registrations are



```
\freeBass "1"
```



```
\freeBass "11"
```



```
\freeBass "10"
```

Used properties:

- `font-size` (0)

`\stdBass` *name* (string)

Generate a standard bass accordion register symbol for *name*.

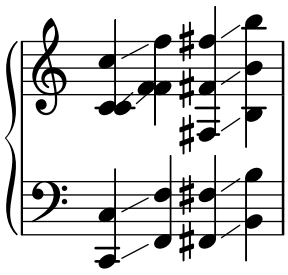
To make it available,

```
#(use-modules (lily accreg))
```

is required near the top of your input file.

The default bass register definitions have been modeled after the article `http://www.accordions.com/index/art/stradella.shtml` originally appearing in *Accord Magazine*.

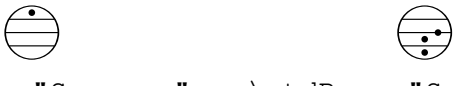
The underlying register model is



This kind of overlapping arrangement is common for Italian instruments though the exact location of the octave breaks differ.

When not composing for a particular target instrument, using the five-reed definitions makes more sense than using a four-reed layout: in that manner, the 'Master' register is unambiguous. This is rather the rule in literature bothering about bass registrations at all.

Available registrations are



`\stdBass "Soprano"`

`\stdBass "Soft Bass"`



`\stdBass "Alto"`



`\stdBass "Soft Tenor"`



`\stdBass "Tenor"`



`\stdBass "Bass/Alto"`



`\stdBass "Master"`

Used properties:

- `font-size` (0)

`\stdBassIV` *name* (string)
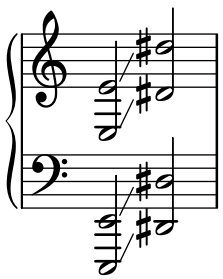
Generate a standard bass accordion register symbol for *name*.

To make it available,

```
#(use-modules (lily accreg))
```

is required near the top of your input file.

The main use is for four-reed standard bass instruments with reedbank layout



Notable instruments are Morino models with MIII (the others are five-reed instead) and the Atlantic IV. Most of those models have three register switches. Some newer Morinos with MIII might have five or even seven.

The prevalent three-register layout uses the middle three switches '`Tenor`', '`Master`', '`Soft Bass`'. Note that the sound is quite darker than the same registrations of '`c,`'-based instruments.

Available registrations are



`\stdBassIV "Soprano"`



`\stdBassIV "Soft Bass"`



`\stdBassIV "Alto"`



`\stdBassIV "Bass/Alto"`



`\stdBassIV "Tenor"`



`\stdBassIV "Soft Bass/Alto"`



`\stdBassIV "Master"`



`\stdBassIV "Soft Tenor"`

Used properties:

- `font-size` (0)

`\stdBassV` *name* (string)
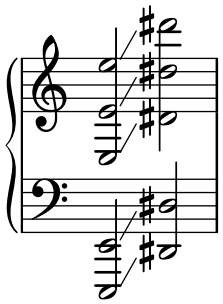
Generate a standard bass accordion register symbol for *name*.

To make it available,

```
#(use-modules (lily accreg))
```

is required near the top of your input file.

The main use is for five-reed standard bass instruments with reedbank layout



This tends to be the bass layout for Hohner's Morino series without converter or MIII manual.

With the exception of the rather new 7-register layout, the highest two chord reeds are usually sounded together. Older instruments offer 5 or 3 bass registers. The Tango VM offers an additional '`Solo Bass`' setting that mutes the chord reeds. The symbol on the register buttons of the Tango VM would actually match the physical five-octave layout reflected here, but it is not used in literature.

Composers should likely prefer the five-reed versions of these symbols. The mismatch of a four-reed instrument with five-reed symbols is easier to resolve for the player than the other way round.

Available registrations are



Used properties:

- `font-size` (0)

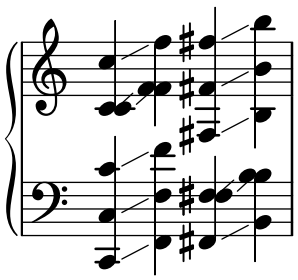
`\stdBassVI` *name* (string)
Generate a standard bass accordion register symbol for six-reed basses as given by *name*.

To make it available,

```
#(use-modules (lily accreg))
```

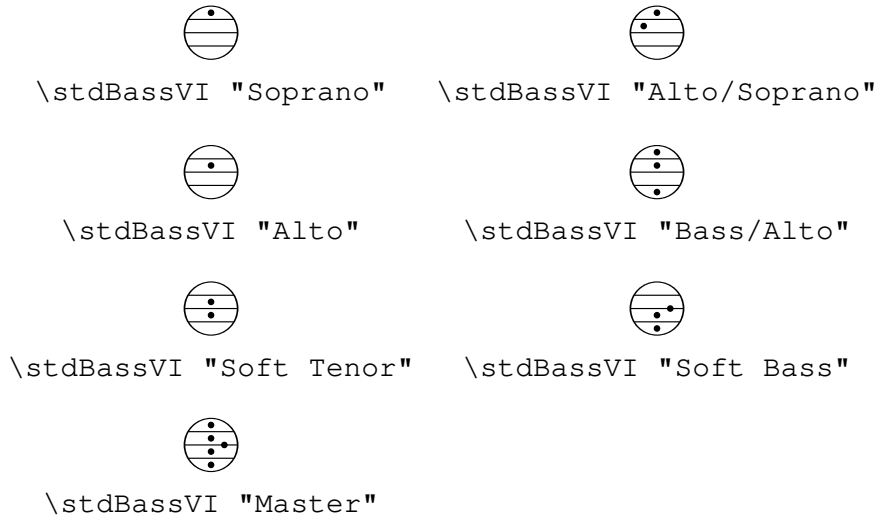
is required near the top of your input file.

This is primarily the register layout for the Hohner "Gola" model. The layout is



The registers are effectively quite similar to that of `\stdBass`. An additional bass reed at alto pitch is omitted for aesthetical reasons from the '`Master`' setting, so the symbols are almost the same except for the '`Alto/Soprano`' register with bass notes at Alto pitch and chords at Soprano pitch.

Available registrations are



Used properties:

- `font-size` (0)

A.1.8 Other markup commands

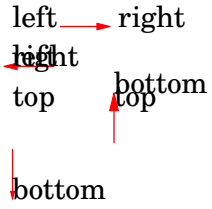
`\annotate-moving` *arg* (markup)

Indicate \vspace and \hspace movement with an arrow.

The arrow changes its size and thickness depending on the printed length; the maximum size of the arrow head can be controlled with the `size` property. If `size` exceeds a third of the length of the final arrow, it falls back to that third.

Note that the arrows do not reflect the actual extents of the objects created by \vspace and \hspace; you might use \box for that.

```
\markup
  \column {
    \line { left \annotate-moving \hspace #4 right }
    \line { left \annotate-moving \hspace #-4 right }
    \line {
      \column {
        top
        \override #'(size . 0.6)
          \annotate-moving \vspace #4/3 bottom
      }
      \column {
        top
        \override #'(size . 2.0)
          \annotate-moving \vspace #-4/3 bottom
      }
    }
  }
```



Used properties:

- `size` (`1`)
- `color` (`"red"`)

`\append-to-tag` *tag* (symbol) *more* (markup) *arg* (markup)

Append *more* to all markup in var *arg* tagged with *tag*.

It works similar to `\appendToTag` for music, but only with markups.

```
tagged = \markup {
  \tag #'foo A
  \tag #'bar B
}

\markup { \append-to-tag #'foo postfoo \tagged }
```



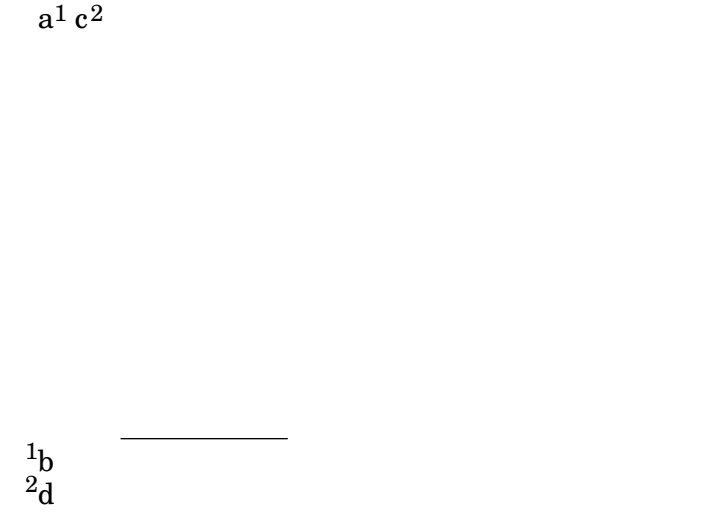
Used properties:

- `tags-with-appends-alist` (`()`)

`\auto-footnote` *mkup* (markup) *note* (markup)

Have footnote *note* act as an annotation to the markup *mkup*.

```
\markup {
  \auto-footnote a b
  \override #'(padding . 0.2)
  \auto-footnote c d
}
```



The footnote will be annotated automatically.

Used properties:

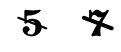
- `padding` (`0.0`)
- `raise` (`0.5`)

`\backslashed-digit` *num* (integer)

Print number *num* with the Emmentaler font, crossed through with a backslash.

This is for use in the context of figured bass notation.

```
\markup {
  \backslashed-digit #5
  \hspace #2
  \override #'(thickness . 3)
  \backslashed-digit #7
}
```



Used properties:

- `thickness` (`1.6`)
- `font-size` (`0`)

`\char` *num* (integer)
: Produce a single Unicode character with code *num*.

  Characters encoded in hexadecimal format require the prefix `#x`.

```
\markup {
  \char #65 \char ##x00a9
}
```



`\eyeglasses`
: Prints out eyeglasses, indicating strongly to look at the conductor.

```
\markup { \eyeglasses }
```



`\first-visible` *args* (markup list)
: Use the first markup in *args* that yields a non-empty stencil and ignore the rest.

```
\markup {
  \first-visible {
    \fromproperty #'header:composer
    \italic Unknown
  }
}
```



`\footnote` *mkup* (markup) *note* (markup)
: Have footnote *note* act as an annotation to the markup *mkup*.

```
\markup {
  \footnote a b
  \override #'(padding . 0.2)
  \footnote c d
}
```

a c

b
d

The footnote will not be annotated automatically.

`\fraction` *arg1* (markup) *arg2* (markup)
: Make a fraction of markups *arg1* and *arg2*.

```
\markup {
  π ≈ \fraction 355 113
}
```

$$\pi \approx \frac{355}{113}$$

Used properties:

- `font-size` (0)

`\fromproperty` *symbol* (symbol)
: Read *symbol* from the property settings and produce a stencil from the markup contained within.

If *symbol* is not defined or is not a markup, return an empty markup.

Currently, the following properties can be accessed.

- Within a `\paper` block defining titles, headers, or footers, or within a `\header` block: all fields from the `\header` block (that produce markup) are available, with `header:` as a name prefix.
- Within a `\paper` block defining headers or footers: the current page number (symbol `page:page-number-string`).
- Within the `tocItemMarkup` paper variable (or in custom-made Scheme code that uses function `add-toc-item!`) defining a table of contents entry: the entry's text and page number are available as `toc:text` and `toc:page`, respectively. An entry's indentation markup is available as `toc:indent`.

```
\header {
  myTitle = "myTitle"
  title = \markup {
    from
    \italic
    \fromproperty #'header:myTitle
  }
}
\markup {
```

```
  \null
}
```

from *myTitle*

\keep-with-tag *tags* (symbol list or symbol) *arg* (markup)

Keep markup from *arg* that is untagged or tagged with *tags*.

All other parts of *arg* that are using a different tag are replaced with empty stencils. It works similar to \keepWithTag for music, but only with markups.

```
tagged = \markup {
  untagged
  \tag #'foo A
  \tag #'bar B
}

\markup { \keep-with-tag #'bar \tagged }
\markup { \keep-with-tag #'foo \tagged }
```

untagged B

untagged A

Used properties:

- tags-to-keep (())

\left-brace *size* (number)

Print a brace from the music font, of height *size* (in points).

```
\markup {
  \left-brace #35
  \hspace #2
  \left-brace #45
}
```



\lookup *glyph-name* (string)

Print a brace glyph with name *glyph-name*.

This is a historical command; \left-brace (which looks up the glyph by absolute size and is independent of the font size) is recommended instead.

```
\markup \lookup "brace200"
```



\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
}
```

H Z

`\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
}
```

H AA

`\null`
: An empty markup with extents of a single point.

```
\markup {
  \null
}
```

`\on-the-fly` *procedure* (procedure) *arg* (markup)
: Apply the *procedure* markup command to *arg*.

  procedure takes the same arguments as `interpret-markup` and returns a stencil.

`\override` *new-prop* (pair) *arg* (markup)
: Add the argument *new-prop* to the property list for printing *arg*.

  In general, any property may be overridden that is part of `font-interface` (Section "font-interface" in *Internals Reference*), `text-interface` (Section "text-interface" in *Internals Reference*), or `instrument-specific-markup-interface` (Section "instrument-specific-markup-interface" in *Internals Reference*). Additionally, various markup commands listen to other properties, too, as described in a markup function's documentation.

  new-prop is either a single alist pair or a non-empty list of alist pairs.

```
\markup {
  \undertie "undertied"
  \override #'(offset . 15)
  \undertie "offset undertied"
  \override #'((offset . 15) (thickness . 3))
  \undertie "offset thick undertied"
}
```

undertied offset undertied offset thick undertied

`\page-link` *page-number* (number) *arg* (markup)
: Add a link to a score's page *page-number* around *arg*.

  This only works in the PDF backend.

```
\markup {
  \page-link #2 { \italic { This links to page 2... } }
}
```

This links to page 2...

\page-ref *label* (symbol) *gauge* (markup) *default* (markup)
: Print a page number reference.

  label is the label set on the referenced page (using \label or \tocItem), *gauge* a markup used to estimate the maximum width of the page number, and *default* the value to display when *label* is not found.

  If the current book or book part is set to use roman numerals for page numbers, the reference will be formatted accordingly – in which case the *gauge*'s width may require additional tweaking.

  Used properties:

  - x-align (1)

\pattern *count* (non-negative, exact integer) *axis* (non-negative, exact integer) *space* (number) *pattern* (markup)
: Print a *pattern* markup *count* times.

  Patterns are spaced apart by *space* (defined as for \hspace or \vspace, respectively) and distributed on *axis*.

```
\markup \column {
  "Horizontally repeated:"
  \pattern #7 #X #2 \flat
  \null
  "Vertically repeated:"
  \pattern #3 #Y #0.5 \flat
}
```

Horizontally repeated:
♭ ♭ ♭ ♭ ♭ ♭ ♭

Vertically repeated:
♭
♭
♭

\property-recursive *symbol* (symbol)
: Print out a warning when header field markup in *symbol* contains some recursive markup definition.

\push-to-tag *tag* (symbol) *more* (markup) *arg* (markup)
: Prepend *more* to all markup in *arg* tagged with *tag*.

  It works similar to \pushToTag for music, but only with markups.

```
tagged = \markup {
  \tag #'foo A
  \tag #'bar B
}

\markup { \push-to-tag #'foo prefoo \tagged }
```

prefoo A B

Used properties:

- `tags-with-pushes-alist` (`()`)

`\qr-code` *width* (non-negative number) *str* (string)

Insert a QR code for string *str*, usually a URL, with a given *width*.

```
\markup \vcenter {
  \center-column { Engraved with LilyPond }
  \hspace #1.5
  \qr-code #10.0 "https://lilypond.org"
}
```

Engraved with LilyPond 

The `error-correction-level` property can be set to one of the symbols `low`, `medium`, `quarter`, or `high`. The higher the level of error correction is, the more the QR code contains redundancy, potentially helping detectors, e.g., in poor lighting conditions; however, a higher correction level also makes the code denser.

```
\markup \vcenter {
  \center-column { Engraved with LilyPond }
  \hspace #1.5
  \override #'(error-correction-level . high)
    \qr-code #10.0 "https://lilypond.org"
}
```

Engraved with LilyPond 

The `quiet-zone-size` property specifies the width of the "quiet zone", namely the white area around the QR code. It is expressed as a multiple of the width of one little square inside the QR code. Use at least 4 for best results.

Used properties:

- `quiet-zone-size` (`4`)
- `error-correction-level` (`low`)

`\remove-with-tag` *tags* (symbol list or symbol) *arg* (markup)

Remove markup from *arg* that is tagged with *tags*.

The removed markup is replaced with empty stencils. It works similar to `\removeWithTag` for music, but only with markups.

```
tagged = \markup {
  \tag #'foo A
  \tag #'bar B
}

\markup { \remove-with-tag #'foo \tagged }
\markup { \remove-with-tag #'bar \tagged }
```

B

A

Used properties:

- `tags-to-remove` (())

`\replace-with-tag` *tag* (symbol) *replacement* (markup) *arg* (markup)

Replace tagged markups in *arg*.

Everything tagged with *tag* (including the tagging itself) gets replaced with *replacement*.

It works similar to `\replaceWithTag` for music, but only with markups.

```
tagged = \markup {
  \tag #'foo A
  \tag #'bar B
}

\markup { \replace-with-tag #'foo C \tagged }
```

C B

Used properties:

- `tags-with-replacement-alist` (())

`\right-brace` *size* (number)

A music brace in point size *size*, rotated 180 degrees.

```
\markup {
  \right-brace #45
  \hspace #2
  \right-brace #35
}
```



`\slashed-digit` *num* (integer)

Print number *num* with the Emmentaler font, crossed through with a slash.

This is for use in the context of figured bass notation.

```
\markup {
  \slashed-digit #5
  \hspace #2
  \override #'(thickness . 3)
  \slashed-digit #7
}
```



Used properties:

- `thickness` (`1.6`)
- `font-size` (`0`)

`\stencil` *stil* (stencil)
: Use stencil *stil* as markup.

```
\markup {
  \stencil #(make-circle-stencil 2 0 #t)
}
```



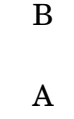
`\strut`
: Create a box of the same height as the space in the current font.

`\tag` *tags* (symbol list or symbol) *arg* (markup)
: Tag markup *arg* with *tag*.

  tag can be one or multiple tags. This allows later on to reference *arg*; for example, to remove it or to add markup before or after the tagged markup. It works similar to `\tag` for music, but only with markups.

```
tagged = \markup {
  \tag #'foo A
  \tag #'bar B
}

\markup { \keep-with-tag #'bar \tagged }
\markup { \keep-with-tag #'foo \tagged }
```



Used properties:

- `tags-with-replacement-alist` (`()`)
- `tags-with-appends-alist` (`()`)
- `tags-with-pushes-alist` (`()`)
- `tags-to-remove` (`()`)
- `tags-to-keep` (`()`)

`\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 "en/included/simple.ly"
}
```

```
% A simple piece in LilyPond, a scale.
\version "2.19.21"
\relative {
  c' d e f g a b c
}
```

Use \withRelativeDir as a prefix to *name* if the file should be found relative to the input file.

\whiteout *arg* (markup)
: Provide a white background for *arg*.

The shape of the white background is determined by the style property. The default is box which produces a rectangle. rounded-box produces a rounded rectangle, and outline approximates the outline of the markup.

The color of the background can be controlled with the color property, defaulting to "white".

```
\markup {
  \combine
    \filled-box #'(-1 . 62) #'(-3 . 4) #1
    \override #'(line-width . 60)
      \fill-line {
        \override #'(thickness . 1.5)
          \whiteout box
        \override #'((style . rounded-box) (thickness . 3))
          \whiteout rounded-box
        \override #'((style . outline) (thickness . 3))
          \whiteout outline
        \override #'((color . "red") (style . outline))
          \whiteout red-outline
      }
}
```



Used properties:

- color ("white")
- thickness (())

- `style` (box)

`\with-color` *col* (color) *arg* (markup)
: Use color *col* to draw *arg*.

  See Section 7.1.5 [Coloring objects], page 289, for valid color specifications.

```
\markup {
  \with-color #red red
  \hspace #2
  \with-color #green green
  \hspace #2
  \with-color "#0000ff" blue
}
```

red green blue

`\with-dimension` *axis* (integer) *val* (pair of numbers) *arg* (markup)
: Set the dimension of *arg* along *axis* to *val*.

  If *axis* is equal to `X`, set the horizontal dimension. If *axis* is equal to `Y`, set the vertical dimension.

`\with-dimension-from` *axis* (integer) *arg1* (markup) *arg2* (markup)
: Print *arg2* but replace the dimension along *axis* with the one from *arg1*.

  If *axis* is set to `X`, replace the horizontal dimension. If *axis* is set to `Y`, replace the vertical dimension.

`\with-dimensions` *x* (pair of numbers) *y* (pair of numbers) *arg* (markup)
: Set the horizontal and vertical dimensions of *arg* to *x* and *y*.

`\with-dimensions-from` *arg1* (markup) *arg2* (markup)
: Print *arg2* with the horizontal and vertical dimensions of *arg1*.

`\with-link` *label* (symbol) *arg* (markup)
: Add a link to the page holding label *label* around *arg*.

  This only works in the PDF backend.

```
\markup {
  \with-link #'label {
    \italic { This links to the page
              containing the label... }
  }
}
```

`\with-outline` *outline* (markup) *arg* (markup)
: Print *arg* with the outline and dimensions of *outline*.

  The outline is used by skylines to resolve collisions (not for whiteout).

`\with-true-dimension` *axis* (integer) *arg* (markup)
: Give *arg* its actual dimension (extent) on *axis*.

  Sometimes, the extents of a markup's printed ink differs from the default extents. The main case is if glyphs are involved. By default, the extents of a glyph are based on the glyph's *metrics* (i.e., a default vertical and horizontal size for the glyph), which, for various reasons, are often not identical to its *bounding box* (i.e., the smallest rectangle that completely encompasses the glyph's outline) – in most cases, the outline protrudes the box spanned up by the metrics.

```
\markup {
  text
  \fontsize #10
  \override #'((box-padding . 0) (thickness . 0.2))
  \box
    \musicglyph "scripts.trill"
  text
}
```



For purposes other than setting text, this behavior may not be wanted. You can use `\with-true-dimension` in order to give the markup its actual printed extent.

```
\markup {
  text
  \fontsize #10
  \override #'((box-padding . 0) (thickness . 0.2))
  \box
    \with-true-dimension #X
    \musicglyph "scripts.trill"
  text
}
```



`\with-true-dimensions` *arg* (markup)

Give *arg* its actual dimensions (extents).

Calling

```
\markup \with-true-dimensions arg
```

is short for

```
\markup
  \with-true-dimension #X
  \with-true-dimension #Y
  arg
```

i.e., `\with-true-dimensions` has the effect of `\with-true-dimension` on both axes.

A.2 Text markup list commands

The following commands can all be used with `\markuplist`.

`\column-lines` *args* (markup list)

Stack the markups in *args* vertically.

Like `\column`, but return a list of lines instead of a single markup. The property `baseline-skip` determines the space between each markup in *args*.

Used properties:

- `baseline-skip`

`\justified-lines` *args* (markup list)

Print *args* as lines aligned both at the left and the right.

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)

Add the argument *new-prop* to the property list for printing *args*.

Like \override but for markup lists.

\score-lines *score* (score)

Inline an image of music as specified by *score*.

Like \score but return a list of lines instead of a single markup.

Used properties:

- tags-with-replacement-alist (())
- tags-with-appends-alist (())
- tags-with-pushes-alist (())
- tags-to-remove (())
- tags-to-keep (())

\string-lines *str* (string)

Split string *str* into lines.

The character to split at is specified by the property split-char, defaulting to #\newline. Surrounding whitespace is removed from every resulting string. The returned list of markups is ready to be formatted by other markup or markup list commands like \column, \line, etc.

```
\markup {
  \column
    \string-lines
     "foo, foo,
     bar, bar,
     buzz, buzz!"
}
```

foo, foo,
bar, bar,
buzz, buzz!

Used properties:

- split-char (#\newline)

\table *column-align* (number list) *lst* (markup list)

Print a table.

column-align specifies how each column is aligned; possible values are -1, 0, and 1. The number of elements in *column-align* determines how many columns will be printed.

The entries to print are given by *lst*, a markup list. If needed, the last row is filled up with point-stencils.

Override the padding property to increase the horizontal distance between columns. Override baseline-skip to increase the vertical distance between rows.

```
% A markup command to print a fixed-width number.
\markup fwnum =
  \markup \override #'(font-features . ("ss01" "-kern"))
    \number \etc

\markuplist {
  \override #'(padding . 2)
  \table #'(0 1 0 -1) {
    \underline { center-aligned right-aligned
                 center-aligned left-aligned }
    one      \fwnum    1 thousandth \fwnum 0.001
    eleven   \fwnum   11 hundredth  \fwnum 0.01
    twenty   \fwnum   20 tenth      \fwnum 0.1
    thousand \fwnum 1000 one        \fwnum 1.0
  }
}
```

center-aligned	right-aligned	center-aligned	left-aligned
one	1	thousandth	0.001
eleven	11	hundredth	0.01
twenty	20	tenth	0.1
thousand	1000	one	1.0

Used properties:

- baseline-skip
- padding (0)

\table-of-contents

Print a table of contents.

This function uses the paper variable tocTitleMarkup for the title; it then prints \tocItem entries line by line.

See Section 21.7 [Table of contents], page 625, for a complete discussion.

Used properties:

- baseline-skip

\tag-list *tags* (symbol list or symbol) *arg* (markup list)

Tag markup list *arg* with *tag*.

tag can be one or multiple tags. This allows later on to reference *arg*; for example, to remove it or to add markup before or after the tagged markup list.

It works like the \tag command for markups but with markup lists. You will need it if you have to reference a whole list; for example, to use \push-to-tag and \append-to-tag without pushing or appending before or after every single item of the list, but before or after the whole list instead.

```
tagged = \markuplist {
  \tag-list #'foo { foo bar }
}

\markup { \push-to-tag #'foo test \tagged }
```



Used properties:

- `tags-with-replacement-alist` (`()`)
- `tags-with-appends-alist` (`()`)
- `tags-with-pushes-alist` (`()`)
- `tags-to-remove` (`()`)
- `tags-to-keep` (`()`)

`\wordwrap-lines` *args* (markup list)
Print *args* as left-aligned lines.

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.

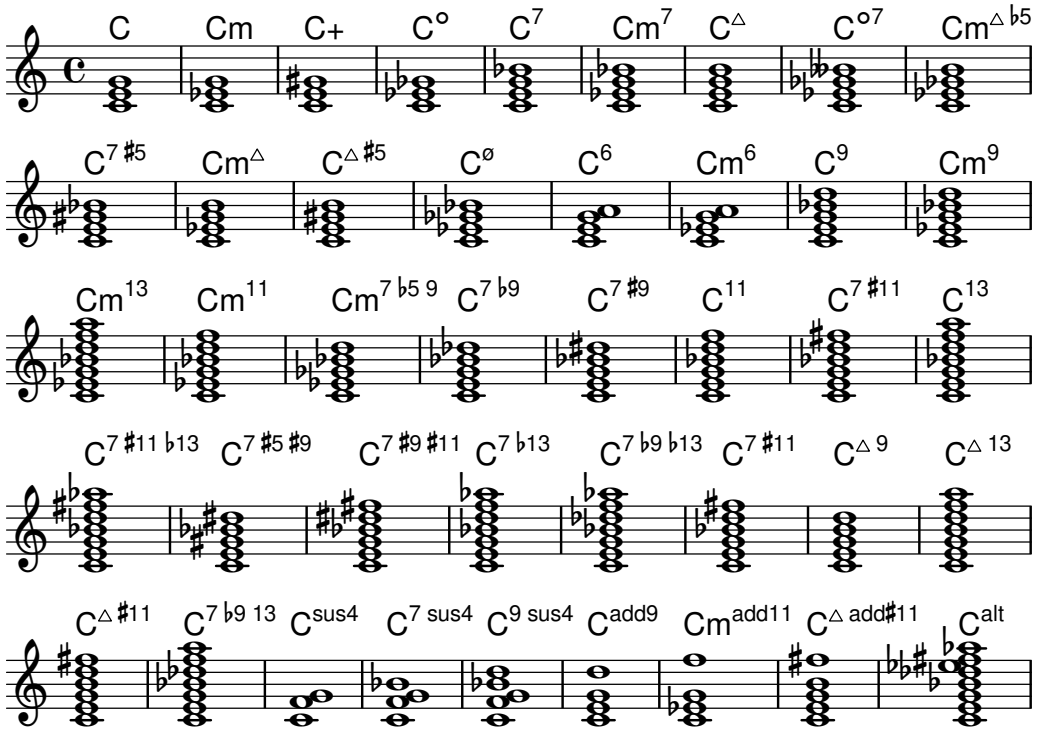
Used properties:

- `text-direction` (`1`)
- `word-space`
- `line-width` (`#f`)
- `baseline-skip`

Appendix B Notation manual tables

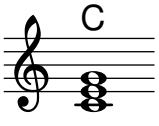
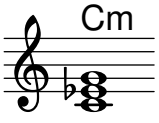
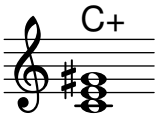
B.1 Chord name chart
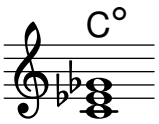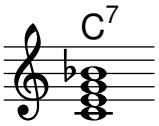
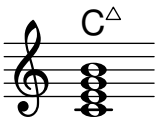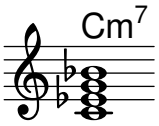
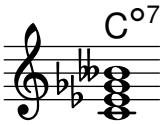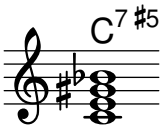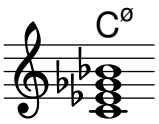
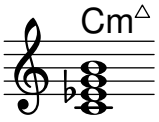
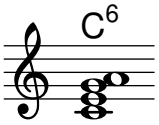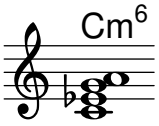
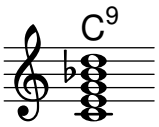
The following chart shows LilyPond's standard system for printing chord names, along with the pitches they represent. Additional (unsupported) naming systems are also demonstrated in the "Chord names alternative" snippet in Section "Chord notation" in *Snippets*, including the notation inspired by Harald Banter (1982) that was used by default in early LilyPond releases (up to version 1.7).

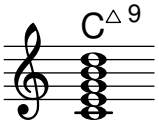
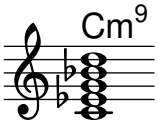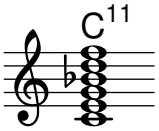
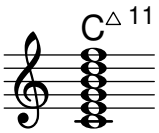
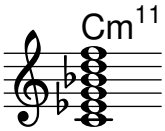
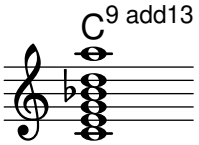
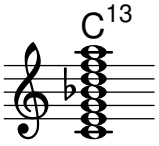
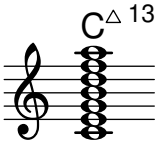
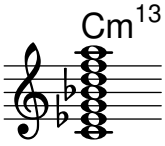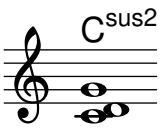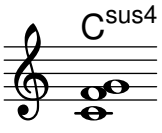


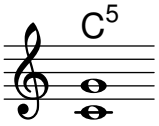
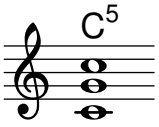
B.2 Common chord modifiers

The following table shows chord modifiers that can be used to generate standard chord structures.

Type	**Interval**	**Modifier**	**Example**	**Output**
Major	Major third, perfect fifth	(nothing)	`c1`	
Minor	Minor third, perfect fifth	`m` or `m5`	`c1:m`	
Augmented	Major third, augmented fifth	`aug`	`c1:aug`	

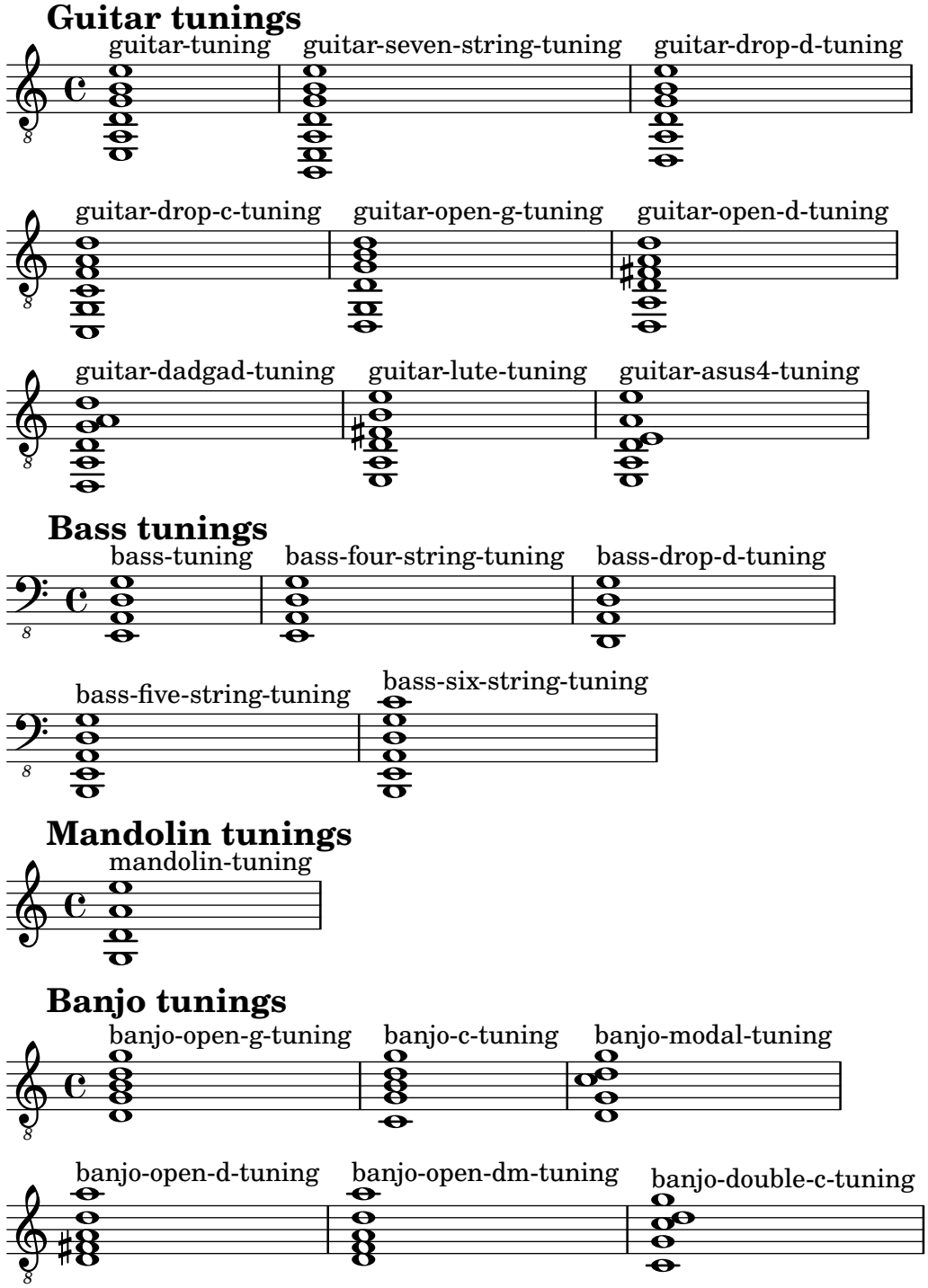
Diminished	Minor third, diminished fifth	`dim`	`c1:dim`	
Dominant seventh	Major triad, minor seventh	`7`	`c1:7`	
Major seventh	Major triad, major seventh	`maj7` or `maj`	`c1:maj7`	
Minor seventh	Minor triad, minor seventh	`m7`	`c1:m7`	
Diminished seventh	Diminished triad, diminished seventh	`dim7`	`c1:dim7`	
Augmented seventh	Augmented triad, minor seventh	`aug7`	`c1:aug7`	
Half-diminished seventh	Diminished triad, minor seventh	`m7.5-`	`c1:m7.5-`	
Minor-major seventh	Minor triad, major seventh	`m7+`	`c1:m7+`	
Major sixth	Major triad, sixth	`6`	`c1:6`	
Minor sixth	Minor triad, sixth	`m6`	`c1:m6`	
Dominant ninth	Dominant seventh, major ninth	`9`	`c1:9`	

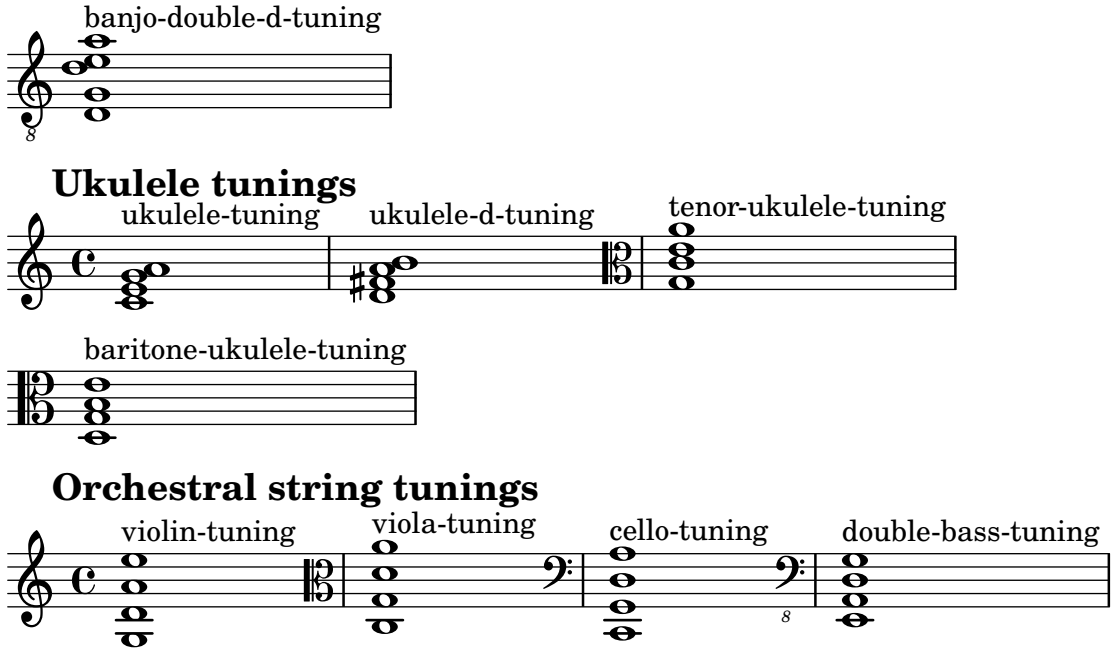
Major ninth	Major seventh, major ninth	`maj9`	`c1:maj9`	
Minor ninth	Minor seventh, major ninth	`m9`	`c1:m9`	
Dominant eleventh	Dominant ninth, perfect eleventh	`11`	`c1:11`	
Major eleventh	Major ninth, perfect eleventh	`maj11`	`c1:maj11`	
Minor eleventh	Minor ninth, perfect eleventh	`m11`	`c1:m11`	
Dominant thirteenth	Dominant ninth, major thirteenth	`13`	`c1:13`	
Dominant thirteenth	Dominant eleventh, major thirteenth	`13.11`	`c1:13.11`	
Major thirteenth	Major eleventh, major thirteenth	`maj13.11`	`c1:maj13.11`	
Minor thirteenth	Minor eleventh, major thirteenth	`m13.11`	`c1:m13.11`	
Suspended second	Major second, perfect fifth	`sus2`	`c1:sus2`	
Suspended fourth	Perfect fourth, perfect fifth	`sus4`	`c1:sus4`	

Power chord (two-voiced)	Perfect fifth	`1.5`	`c1:5`	
Power chord (three-voiced)	Perfect fifth, octave	`1.5.8`	`c1:5.8`	

B.3 Predefined string tunings

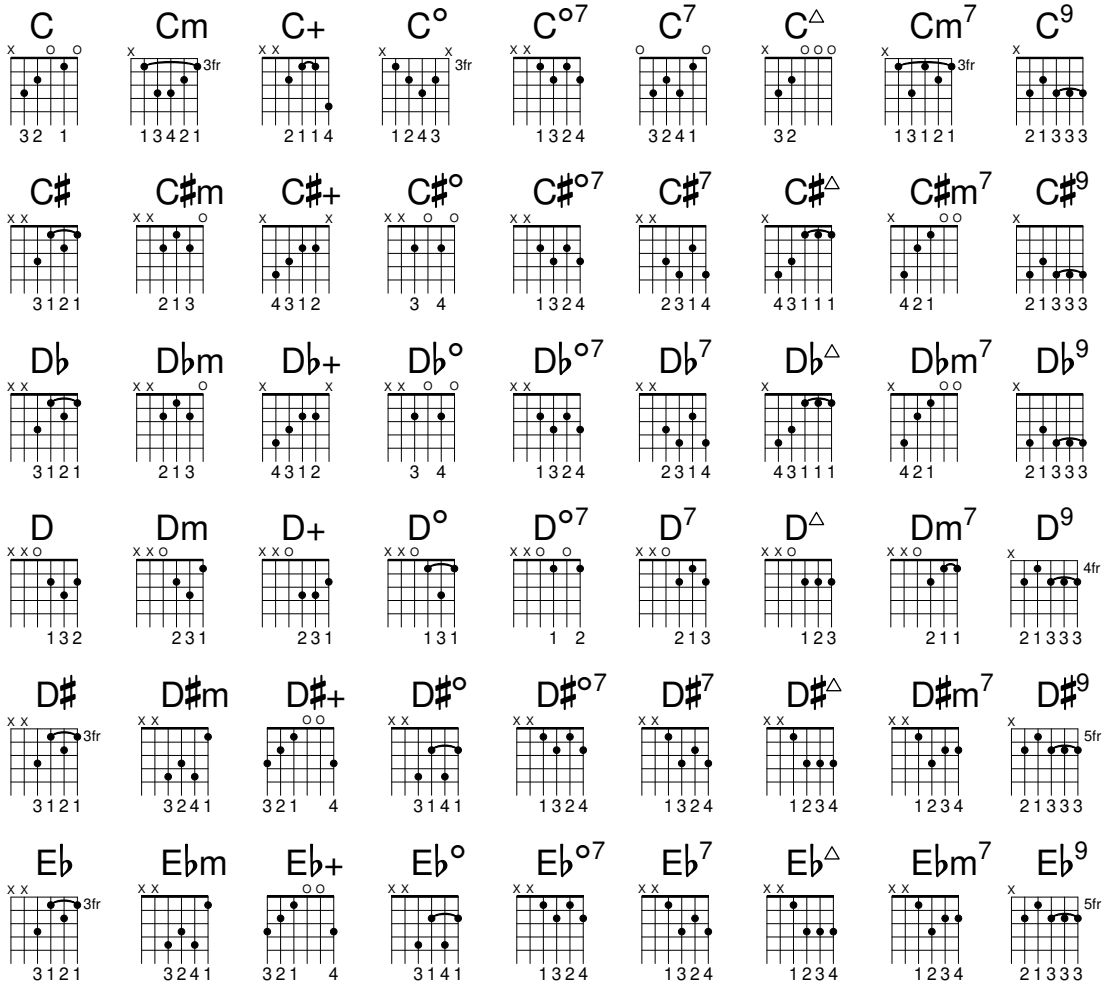
The chart below shows the predefined string tunings.

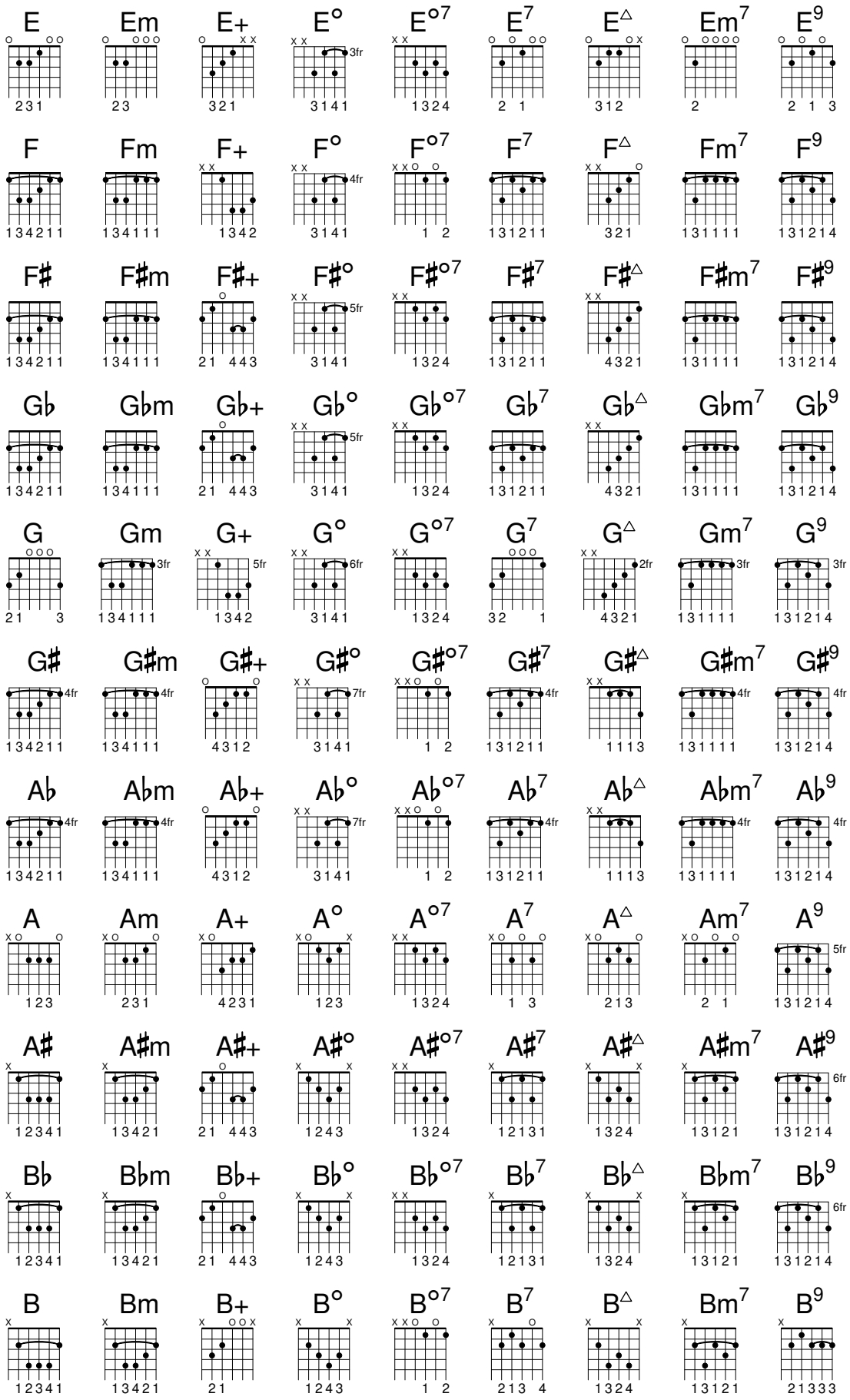




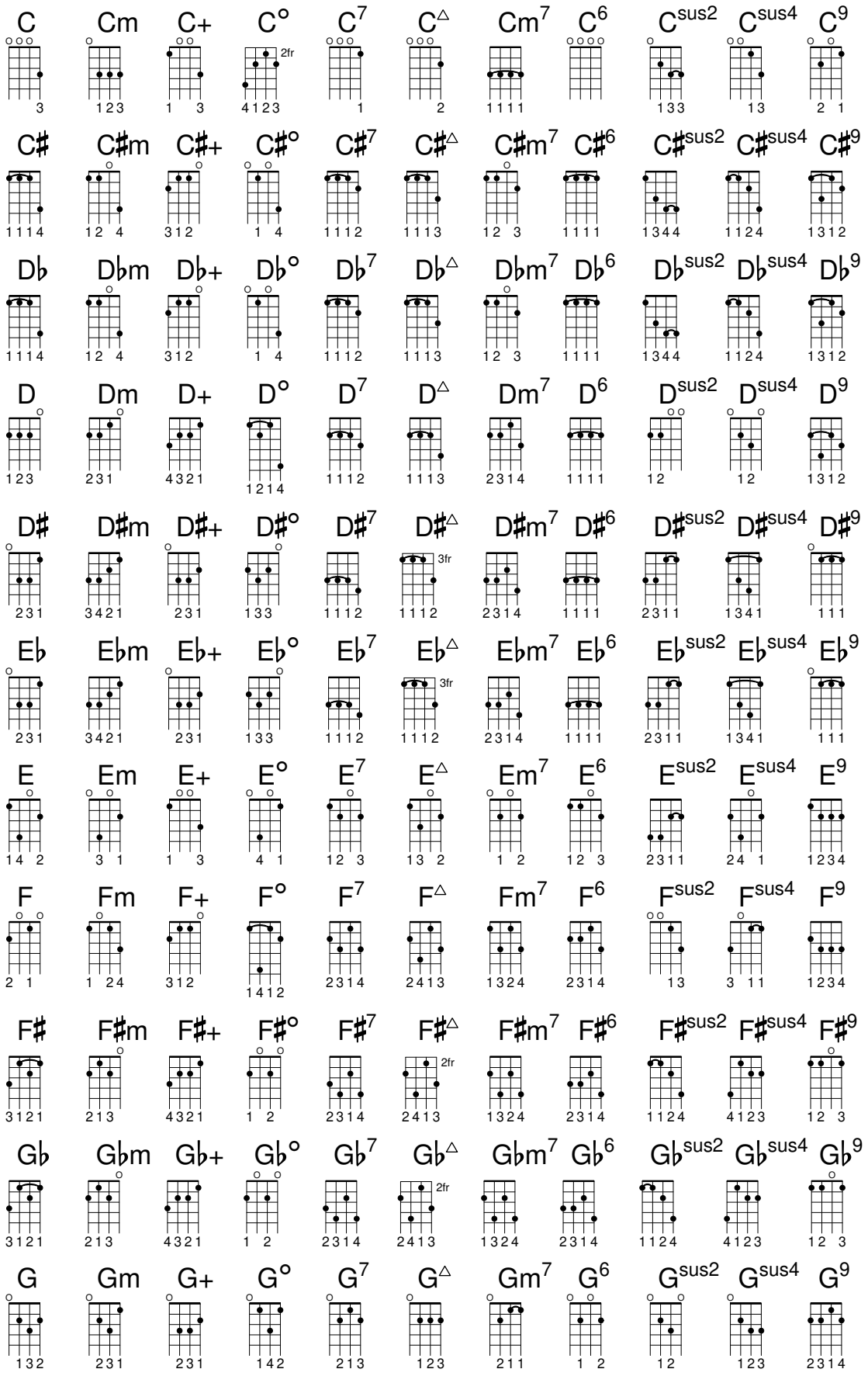
B.4 Predefined fretboard diagrams

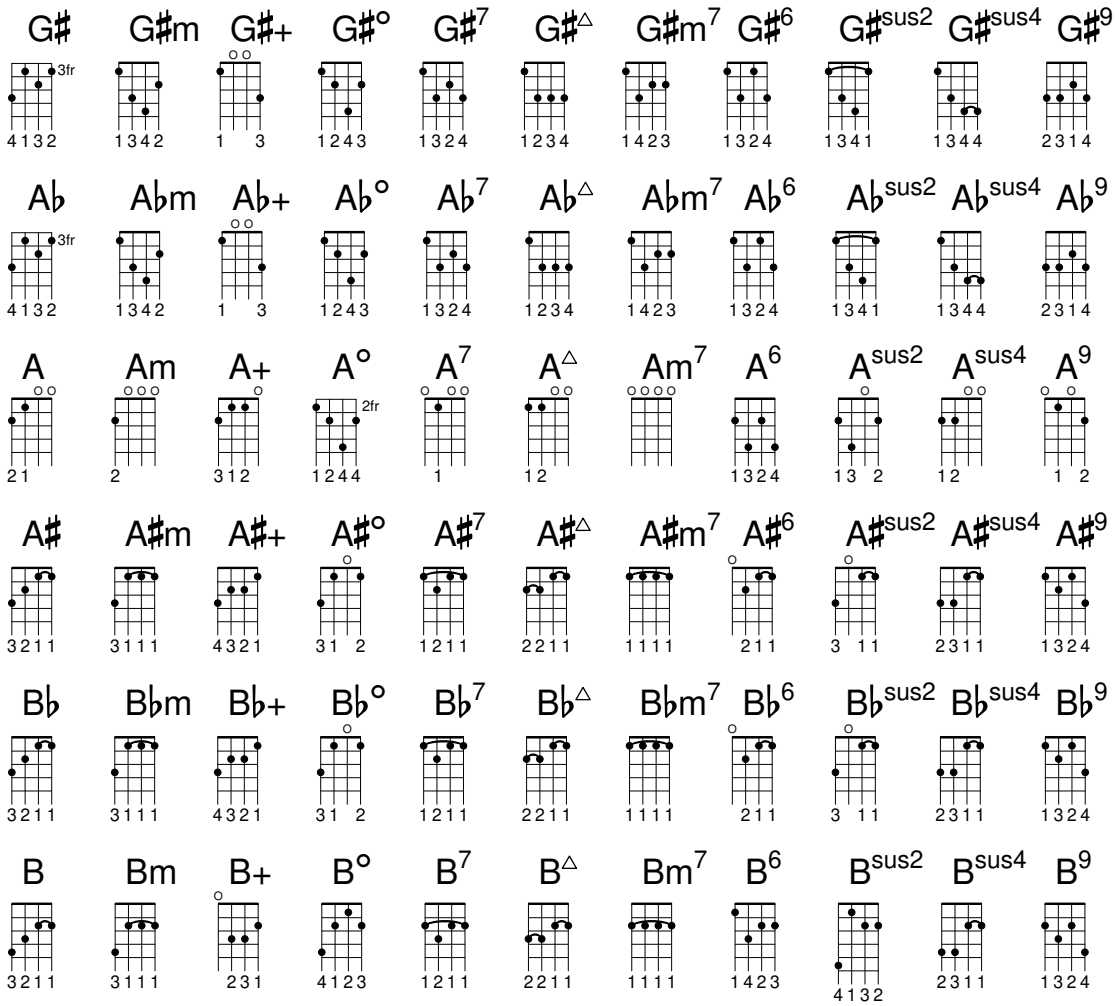
B.4.1 Diagrams for Guitar



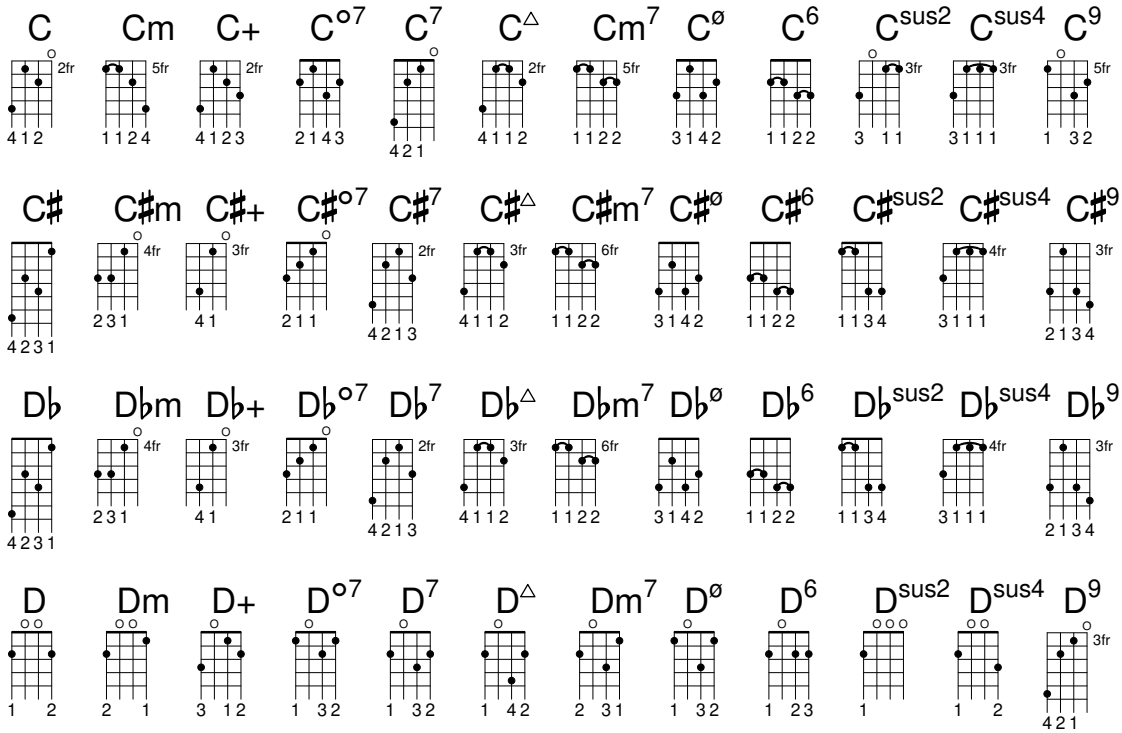

E Em E+ E° E°7 E7 E△ Em7 E9
F Fm F+ F° F°7 F7 F△ Fm7 F9
F♯ F♯m F♯+ F♯° F♯°7 F♯7 F♯△ F♯m7 F♯9
G♭ G♭m G♭+ G♭° G♭°7 G♭7 G♭△ G♭m7 G♭9
G Gm G+ G° G°7 G7 G△ Gm7 G9
G♯ G♯m G♯+ G♯° G♯°7 G♯7 G♯△ G♯m7 G♯9
A♭ A♭m A♭+ A♭° A♭°7 A♭7 A♭△ A♭m7 A♭9
A Am A+ A° A°7 A7 A△ Am7 A9
A♯ A♯m A♯+ A♯° A♯°7 A♯7 A♯△ A♯m7 A♯9
B♭ B♭m B♭+ B♭° B♭°7 B♭7 B♭△ B♭m7 B♭9
B Bm B+ B° B°7 B7 B△ Bm7 B9
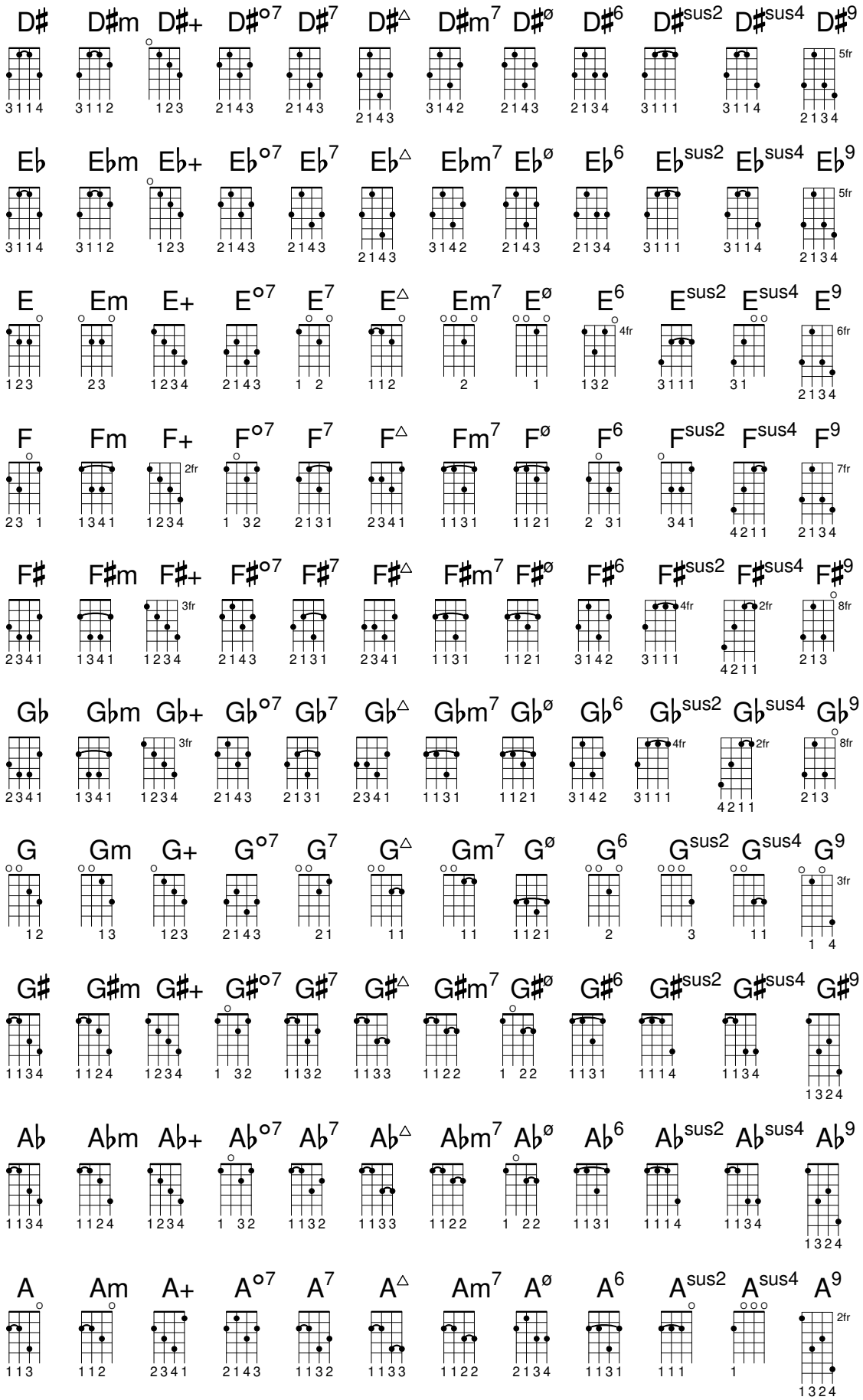
B.4.2 Diagrams for Ukulele

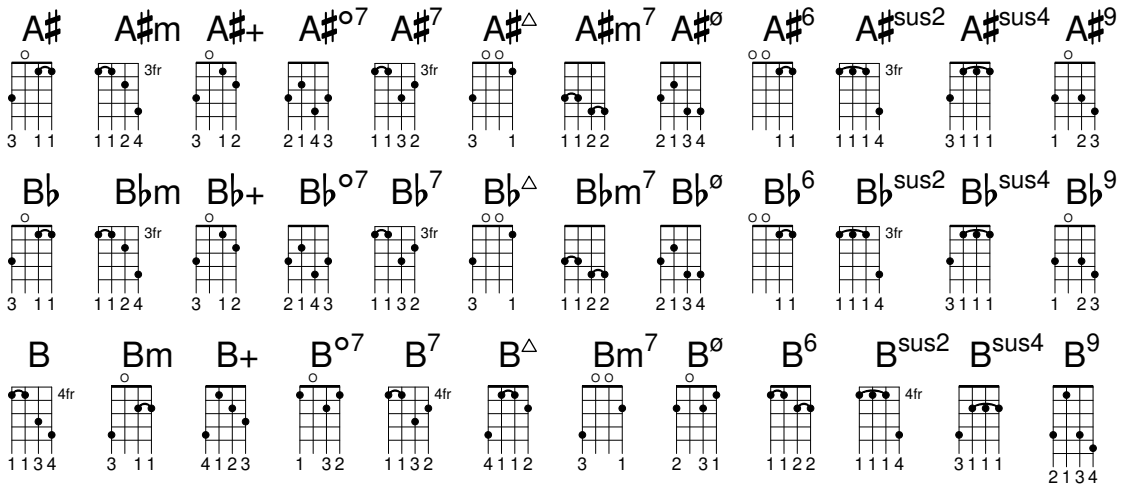




B.4.3 Diagrams for Mandolin




D♯ D♯m D♯+ D♯o7 D♯7 D♯△ D♯m7 D♯ø D♯6 D♯sus2 D♯sus4 D♯9
E♭ E♭m E♭+ E♭o7 E♭7 E♭△ E♭m7 E♭ø E♭6 E♭sus2 E♭sus4 E♭9
E Em E+ Eo7 E7 E△ Em7 Eø E6 Esus2 Esus4 E9
F Fm F+ Fo7 F7 F△ Fm7 Fø F6 Fsus2 Fsus4 F9
F♯ F♯m F♯+ F♯o7 F♯7 F♯△ F♯m7 F♯ø F♯6 F♯sus2 F♯sus4 F♯9
G♭ G♭m G♭+ G♭o7 G♭7 G♭△ G♭m7 G♭ø G♭6 G♭sus2 G♭sus4 G♭9
G Gm G+ Go7 G7 G△ Gm7 Gø G6 Gsus2 Gsus4 G9
G♯ G♯m G♯+ G♯o7 G♯7 G♯△ G♯m7 G♯ø G♯6 G♯sus2 G♯sus4 G♯9
A♭ A♭m A♭+ A♭o7 A♭7 A♭△ A♭m7 A♭ø A♭6 A♭sus2 A♭sus4 A♭9
A Am A+ Ao7 A7 A△ Am7 Aø A6 Asus2 Asus4 A9



B.5 Predefined paper sizes

Paper sizes are defined in file `scm/paper.scm`.

ISO 216, A series

`"a10"`	26 mm x 37 mm (1.02 in x 1.46 in)
`"a9"`	37 mm x 52 mm (1.46 in x 2.05 in)
`"a8"`	52 mm x 74 mm (2.05 in x 2.91 in)
`"a7"`	74 mm x 105 mm (2.91 in x 4.13 in)
`"a6"`	105 mm x 148 mm (4.13 in x 5.83 in)
`"a5"`	148 mm x 210 mm (5.83 in x 8.27 in)
`"a4"`	210 mm x 297 mm (8.27 in x 11.69 in)
`"a3"`	297 mm x 420 mm (11.69 in x 16.54 in)
`"a2"`	420 mm x 594 mm (16.54 in x 23.39 in)
`"a1"`	594 mm x 841 mm (23.39 in x 33.11 in)
`"a0"`	841 mm x 1189 mm (33.11 in x 46.81 in)

Two extended sizes as defined in DIN 476

`"2a0"`	1189 mm x 1682 mm (46.81 in x 66.22 in)
`"4a0"`	1682 mm x 2378 mm (66.22 in x 93.62 in)

ISO 216, B series

`"b10"`	31 mm x 44 mm (1.22 in x 1.73 in)
`"b9"`	44 mm x 62 mm (1.73 in x 2.44 in)
`"b8"`	62 mm x 88 mm (2.44 in x 3.46 in)
`"b7"`	88 mm x 125 mm (3.46 in x 4.92 in)
`"b6"`	125 mm x 176 mm (4.92 in x 6.93 in)
`"b5"`	176 mm x 250 mm (6.93 in x 9.84 in)
`"b4"`	250 mm x 353 mm (9.84 in x 13.90 in)
`"b3"`	353 mm x 500 mm (13.90 in x 19.69 in)
`"b2"`	500 mm x 707 mm (19.69 in x 27.83 in)
`"b1"`	707 mm x 1000 mm (27.83 in x 39.37 in)
`"b0"`	1000 mm x 1414 mm (39.37 in x 55.67 in)

ISO 269, C series

`"c10"`	28 mm x 40 mm (1.10 in x 1.57 in)
`"c9"`	40 mm x 57 mm (1.57 in x 2.24 in)
`"c8"`	57 mm x 81 mm (2.24 in x 3.19 in)
`"c7"`	81 mm x 114 mm (3.19 in x 4.49 in)
`"c6"`	114 mm x 162 mm (4.49 in x 6.38 in)

`"c5"`	162 mm x 229 mm (6.38 in x 9.02 in)
`"c4"`	229 mm x 324 mm (9.02 in x 12.76 in)
`"c3"`	324 mm x 458 mm (12.76 in x 18.03 in)
`"c2"`	458 mm x 648 mm (18.03 in x 25.51 in)
`"c1"`	648 mm x 917 mm (25.51 in x 36.10 in)
`"c0"`	917 mm x 1297 mm (36.10 in x 51.06 in)

North American paper sizes

`"junior-legal"`	5.0 in x 8.0 in (127 mm x 203 mm)
`"legal"`	8.5 in x 14.0 in (216 mm x 356 mm)
`"ledger"`	17.0 in x 11.0 in (432 mm x 279 mm)
`"17x11"`	17.0 in x 11.0 in (432 mm x 279 mm)
`"letter"`	8.5 in x 11.0 in (216 mm x 279 mm)
`"tabloid"`	11.0 in x 17.0 in (279 mm x 432 mm)
`"11x17"`	11.0 in x 17.0 in (279 mm x 432 mm)

Sizes by IEEE Printer Working Group, for children's writing

`"government-letter"`	8.0 in x 10.5 in (203 mm x 267 mm)
`"government-legal"`	8.5 in x 13.0 in (216 mm x 330 mm)
`"philippine-legal"`	8.5 in x 13.0 in (216 mm x 330 mm)

ANSI sizes

`"ansi a"`	8.5 in x 11.0 in (216 mm x 279 mm)
`"ansi b"`	11.0 in x 17.0 in (279 mm x 432 mm)
`"ansi c"`	17.0 in x 22.0 in (432 mm x 559 mm)
`"ansi d"`	22.0 in x 34.0 in (559 mm x 864 mm)
`"ansi e"`	34.0 in x 44.0 in (864 mm x 1118 mm)
`"engineering f"`	28.0 in x 40.0 in (711 mm x 1016 mm)

North American architectural sizes

`"arch a"`	9.0 in x 12.0 in (229 mm x 305 mm)
`"arch b"`	12.0 in x 18.0 in (305 mm x 457 mm)
`"arch c"`	18.0 in x 24.0 in (457 mm x 610 mm)
`"arch d"`	24.0 in x 36.0 in (610 mm x 914 mm)
`"arch e"`	36.0 in x 48.0 in (914 mm x 1219 mm)
`"arch e1"`	30.0 in x 42.0 in (762 mm x 1067 mm)

Other sizes, including antique sizes still used in the United Kingdom

`"statement"`	5.5 in x 8.5 in (140 mm x 216 mm)
`"half letter"`	5.5 in x 8.5 in (140 mm x 216 mm)
`"quarto"`	8.0 in x 10.0 in (203 mm x 254 mm)
`"octavo"`	6.75 in x 10.5 in (171 mm x 267 mm)
`"executive"`	7.25 in x 10.5 in (184 mm x 267 mm)
`"monarch"`	7.25 in x 10.5 in (184 mm x 267 mm)
`"foolscap"`	8.27 in x 13.0 in (210 mm x 330 mm)
`"folio"`	8.27 in x 13.0 in (210 mm x 330 mm)
`"super-b"`	13.0 in x 19.0 in (330 mm x 483 mm)
`"post"`	15.5 in x 19.5 in (394 mm x 495 mm)
`"crown"`	15.0 in x 20.0 in (381 mm x 508 mm)
`"large post"`	16.5 in x 21.0 in (419 mm x 533 mm)
`"demy"`	17.5 in x 22.5 in (445 mm x 572 mm)
`"medium"`	18.0 in x 23.0 in (457 mm x 584 mm)
`"broadsheet"`	18.0 in x 24.0 in (457 mm x 610 mm)
`"royal"`	20.0 in x 25.0 in (508 mm x 635 mm)
`"elephant"`	23.0 in x 28.0 in (584 mm x 711 mm)

"double demy"	22.5 in x 35.0 in (572 mm x 889 mm)
"quad demy"	35.0 in x 45.0 in (889 mm x 1143 mm)
"atlas"	26.0 in x 34.0 in (660 mm x 864 mm)
"imperial"	22.0 in x 30.0 in (559 mm x 762 mm)
"antiquarian"	31.0 in x 53.0 in (787 mm x 1346 mm)

PA4-based sizes

"pa10"	26 mm x 35 mm (1.02 in x 1.38 in)
"pa9"	35 mm x 52 mm (1.38 in x 2.05 in)
"pa8"	52 mm x 70 mm (2.05 in x 2.76 in)
"pa7"	70 mm x 105 mm (2.76 in x 4.13 in)
"pa6"	105 mm x 140 mm (4.13 in x 5.51 in)
"pa5"	140 mm x 210 mm (5.51 in x 8.27 in)
"pa4"	210 mm x 280 mm (8.27 in x 11.02 in)
"pa3"	280 mm x 420 mm (11.02 in x 16.54 in)
"pa2"	420 mm x 560 mm (16.54 in x 22.05 in)
"pa1"	560 mm x 840 mm (22.05 in x 33.07 in)
"pa0"	840 mm x 1120 mm (33.07 in x 44.09 in)

Additional format for use in Southeast Asia and Australia

"f4"	210 mm x 330 mm (8.27 in x 12.99 in)

B.6 MIDI instruments

The following is a list of names that can be used for the `midiInstrument` property. Each is presented with its number among the General MIDI Standard's 128 Program Numbers.

1	acoustic grand	2	bright acoustic	3	electric grand
4	honky-tonk	5	electric piano 1	6	electric piano 2
7	harpsichord	8	clav	9	celesta
10	glockenspiel	11	music box	12	vibraphone
13	marimba	14	xylophone	15	tubular bells
16	dulcimer	17	drawbar organ	18	percussive organ
19	rock organ	20	church organ	21	reed organ
22	accordion	23	harmonica	24	concertina
25	acoustic guitar (nylon)	26	acoustic guitar (steel)	27	electric guitar (jazz)
28	electric guitar (clean)	29	electric guitar (muted)	30	overdriven guitar
31	distorted guitar	32	guitar harmonics	33	acoustic bass
34	electric bass (finger)	35	electric bass (pick)	36	fretless bass
37	slap bass 1	38	slap bass 2	39	synth bass 1
40	synth bass 2	41	violin	42	viola
43	cello	44	contrabass	45	tremolo strings
46	pizzicato strings	47	orchestral harp	48	timpani
49	string ensemble 1	50	string ensemble 2	51	synthstrings 1
52	synthstrings 2	53	choir aahs	54	voice oohs
55	synth voice	56	orchestra hit	57	trumpet
58	trombone	59	tuba	60	muted trumpet
61	french horn	62	brass section	63	synthbrass 1
64	synthbrass 2	65	soprano sax	66	alto sax
67	tenor sax	68	baritone sax	69	oboe
70	english horn	71	bassoon	72	clarinet
73	piccolo	74	flute	75	recorder
76	pan flute	77	blown bottle	78	shakuhachi
79	whistle	80	ocarina	81	lead 1 (square)
82	lead 2 (sawtooth)	83	lead 3 (calliope)	84	lead 4 (chiff)
85	lead 5 (charang)	86	lead 6 (voice)	87	lead 7 (fifths)
88	lead 8 (bass+lead)	89	pad 1 (new age)	90	pad 2 (warm)
91	pad 3 (polysynth)	92	pad 4 (choir)	93	pad 5 (bowed)
94	pad 6 (metallic)	95	pad 7 (halo)	96	pad 8 (sweep)
97	fx 1 (rain)	98	fx 2 (soundtrack)	99	fx 3 (crystal)
100	fx 4 (atmosphere)	101	fx 5 (brightness)	102	fx 6 (goblins)

103	fx 7 (echoes)	104	fx 8 (sci-fi)	105	sitar
106	banjo	107	shamisen	108	koto
109	kalimba	110	bagpipe	111	fiddle
112	shanai	113	tinkle bell	114	agogo
115	steel drums	116	woodblock	117	taiko drum
118	melodic tom	119	synth drum	120	reverse cymbal
121	guitar fret noise	122	breath noise	123	seashore
124	bird tweet	125	telephone ring	126	helicopter
127	applause	128	gunshot		

The drum kits listed below are also possible.

1	standard kit	9	room kit	17	power kit
25	electronic kit	26	tr-808 kit	33	jazz kit
41	brush kit	49	orchestra kit	57	sfx kit
128	mt-32 kit/cm-64 kit				

B.7 List of colors

Normal colors

See Section 7.1.5 [Coloring objects], page 289, for detailed usage syntax.

black	white	red	green
blue	cyan	magenta	yellow
grey	darkred	darkgreen	darkblue
darkcyan	darkmagenta	darkyellow	

CSS color names

CSS color names may be used as-is in string arguments.

aliceblue	antiquewhite	aqua
aquamarine	azure	beige
bisque	black	blanchedalmond
blue	blueviolet	brown
burlywood	cadetblue	chartreuse
chocolate	coral	cornflowerblue
cornsilk	crimson	cyan
darkblue	darkcyan	darkgoldenrod
darkgray	darkgreen	darkgrey
darkkhaki	darkmagenta	darkolivegreen
darkorange	darkorchid	darkred
darksalmon	darkseagreen	darkslateblue
darkslategray	darkslategrey	darkturquoise
darkviolet	deeppink	deepskyblue
dimgray	dimgrey	dodgerblue
firebrick	floralwhite	forestgreen
fuchsia	gainsboro	ghostwhite
gold	goldenrod	gray
green	greenyellow	grey
honeydew	hotpink	indianred
indigo	ivory	khaki
lavender	lavenderblush	lawngreen
lemonchiffon	lightblue	lightcoral
lightcyan	lightgoldenrodyellow	lightgray
lightgreen	lightgrey	lightpink
lightsalmon	lightseagreen	lightskyblue

lightslategray	lightslategrey	lightsteelblue
lightyellow	lime	limegreen
linen	magenta	maroon
mediumaquamarine	mediumblue	mediumorchid
mediumpurple	mediumseagreen	mediumslateblue
mediumspringgreen	mediumturquoise	mediumvioletred
midnightblue	mintcream	mistyrose
moccasin	navajowhite	navy
oldlace	olive	olivedrab
orange	orangered	orchid
palegoldenrod	palegreen	paleturquoise
palevioletred	papayawhip	peachpuff
peru	pink	plum
powderblue	purple	rebeccapurple
red	rosybrown	royalblue
saddlebrown	salmon	sandybrown
seagreen	seashell	sienna
silver	skyblue	slateblue
slategray	slategrey	snow
springgreen	steelblue	tan
teal	thistle	tomato
turquoise	violet	wheat
white	whitesmoke	yellow
yellowgreen		

CSS color definitions differ from X color names for the following colors: `green`, `grey`, `maroon`, `purple`.

X color names

X11 color names (`https://en.wikipedia.org/wiki/X11_color_names`) offer a wider choice than CSS names. They come in several variants:

- Any name that is spelled as a single word with capitalization (e.g., 'LightSlateBlue') can also be spelled as space-separated words with or without capitalization (e.g., 'light slate blue').
- The word 'grey' can always be spelled 'gray' (e.g., 'DarkSlateGray'), without any difference in the output.
- Some names can take a numerical suffix (e.g., 'LightSalmon4').

Use the Scheme function `x11-color` to access them.

Color Names without a numerical suffix

The following table presents all color names that may be used without a numerical suffix.

aliceblue	antiquewhite	aquamarine
azure	beige	bisque
black	blanchedalmond	blue
blueviolet	brown	burlywood
cadetblue	chartreuse	chocolate
coral	cornflowerblue	cornsilk
cyan	darkblue	darkcyan
darkgoldenrod	darkgray	darkgreen
darkgrey	darkkhaki	darkmagenta

darkolivegreen darkorange darkorchid
darkred darksalmon darkseagreen
darkslateblue darkslategray darkslategrey
darkturquoise darkviolet deeppink
deepskyblue dimgray dimgrey
dodgerblue firebrick floralwhite
forestgreen gainsboro ghostwhite
gold goldenrod gray
green greenyellow grey
honeydew hotpink indianred
ivory khaki lavender
lavenderblush lawngreen lemonchiffon
lightblue lightcoral lightcyan
lightgoldenrod lightgoldenrodyellow lightgray
lightgreen lightgrey lightpink
lightsalmon lightseagreen lightskyblue
lightslateblue lightslategray lightslategrey
lightsteelblue lightyellow limegreen
linen magenta maroon
mediumaquamarine mediumblue mediumorchid
mediumpurple mediumseagreen mediumslateblue
mediumspringgreen mediumturquoise mediumvioletred
midnightblue mintcream mistyrose
moccasin navajowhite navy
navyblue oldlace olivedrab
orange orangered orchid
palegoldenrod palegreen paleturquoise
palevioletred papayawhip peachpuff
peru pink plum
powderblue purple red
rosybrown royalblue saddlebrown
salmon sandybrown seagreen
seashell sienna skyblue
slateblue slategray slategrey
snow springgreen steelblue
tan thistle tomato
turquoise violet violetred
wheat white whitesmoke
yellow yellowgreen

Color names with a numerical suffix

A subset of the X11 color names may be used with a numerical suffix in the range 1 to 4 to get darker shades. Note that for such colors, color `foo` is identical to color `foo1`.

antiquewhite1 antiquewhite2 antiquewhite3 antiquewhite4
aquamarine1 aquamarine2 aquamarine3 aquamarine4
azure1 azure2 azure3 azure4
bisque1 bisque2 bisque3 bisque4
blue1 blue2 blue3 blue4
brown1 brown2 brown3 brown4
burlywood1 burlywood2 burlywood3 burlywood4

cadetblue1	cadetblue2	cadetblue3	cadetblue4
chartreuse1	chartreuse2	chartreuse3	chartreuse4
chocolate1	chocolate2	chocolate3	chocolate4
coral1	coral2	coral3	coral4
cornsilk1	cornsilk2	cornsilk3	cornsilk4
cyan1	cyan2	cyan3	cyan4
darkgoldenrod1	darkgoldenrod2	darkgoldenrod3	darkgoldenrod4
darkolivegreen1	darkolivegreen2	darkolivegreen3	darkolivegreen4
darkorange1	darkorange2	darkorange3	darkorange4
darkorchid1	darkorchid2	darkorchid3	darkorchid4
darkseagreen1	darkseagreen2	darkseagreen3	darkseagreen4
darkslategray1	darkslategray2	darkslategray3	darkslategray4
deeppink1	deeppink2	deeppink3	deeppink4
deepskyblue1	deepskyblue2	deepskyblue3	deepskyblue4
dodgerblue1	dodgerblue2	dodgerblue3	dodgerblue4
firebrick1	firebrick2	firebrick3	firebrick4
gold1	gold2	gold3	gold4
goldenrod1	goldenrod2	goldenrod3	goldenrod4
green1	green2	green3	green4
honeydew1	honeydew2	honeydew3	honeydew4
hotpink1	hotpink2	hotpink3	hotpink4
indianred1	indianred2	indianred3	indianred4
ivory1	ivory2	ivory3	ivory4
khaki1	khaki2	khaki3	khaki4
lavenderblush1	lavenderblush2	lavenderblush3	lavenderblush4
lemonchiffon1	lemonchiffon2	lemonchiffon3	lemonchiffon4
lightblue1	lightblue2	lightblue3	lightblue4
lightcyan1	lightcyan2	lightcyan3	lightcyan4
lightgoldenrod1	lightgoldenrod2	lightgoldenrod3	lightgoldenrod4
lightpink1	lightpink2	lightpink3	lightpink4
lightsalmon1	lightsalmon2	lightsalmon3	lightsalmon4
lightskyblue1	lightskyblue2	lightskyblue3	lightskyblue4
lightsteelblue1	lightsteelblue2	lightsteelblue3	lightsteelblue4
lightyellow1	lightyellow2	lightyellow3	lightyellow4
magenta1	magenta2	magenta3	magenta4
maroon1	maroon2	maroon3	maroon4
mediumorchid1	mediumorchid2	mediumorchid3	mediumorchid4
mediumpurple1	mediumpurple2	mediumpurple3	mediumpurple4
mistyrose1	mistyrose2	mistyrose3	mistyrose4
navajowhite1	navajowhite2	navajowhite3	navajowhite4
olivedrab1	olivedrab2	olivedrab3	olivedrab4
orange1	orange2	orange3	orange4
orangered1	orangered2	orangered3	orangered4
orchid1	orchid2	orchid3	orchid4
palegreen1	palegreen2	palegreen3	palegreen4
paleturquoise1	paleturquoise2	paleturquoise3	paleturquoise4
palevioletred1	palevioletred2	palevioletred3	palevioletred4
peachpuff1	peachpuff2	peachpuff3	peachpuff4
pink1	pink2	pink3	pink4
plum1	plum2	plum3	plum4

purple1	purple2	purple3	purple4
red1	red2	red3	red4
rosybrown1	rosybrown2	rosybrown3	rosybrown4
royalblue1	royalblue2	royalblue3	royalblue4
salmon1	salmon2	salmon3	salmon4
seagreen1	seagreen2	seagreen3	seagreen4
seashell1	seashell2	seashell3	seashell4
sienna1	sienna2	sienna3	sienna4
skyblue1	skyblue2	skyblue3	skyblue4
slateblue1	slateblue2	slateblue3	slateblue4
slategray1	slategray2	slategray3	slategray4
snow1	snow2	snow3	snow4
springgreen1	springgreen2	springgreen3	springgreen4
steelblue1	steelblue2	steelblue3	steelblue4
tan1	tan2	tan3	tan4
thistle1	thistle2	thistle3	thistle4
tomato1	tomato2	tomato3	tomato4
turquoise1	turquoise2	turquoise3	turquoise4
violetred1	violetred2	violetred3	violetred4
wheat1	wheat2	wheat3	wheat4
yellow1	yellow2	yellow3	yellow4

Grey Scale

The set of X11 colors also contains a palette of 101 gray shades.

gray0	gray1	gray2	gray3	gray4
gray5	gray6	gray7	gray8	gray9
gray10	gray11	gray12	gray13	gray14
gray15	gray16	gray17	gray18	gray19
gray20	gray21	gray22	gray23	gray24
gray25	gray26	gray27	gray28	gray29
gray30	gray31	gray32	gray33	gray34
gray35	gray36	gray37	gray38	gray39
gray40	gray41	gray42	gray43	gray44
gray45	gray46	gray47	gray48	gray49
gray50	gray51	gray52	gray53	gray54
gray55	gray56	gray57	gray58	gray59
gray60	gray61	gray62	gray63	gray64
gray65	gray66	gray67	gray68	gray69
gray70	gray71	gray72	gray73	gray74
gray75	gray76	gray77	gray78	gray79
gray80	gray81	gray82	gray83	gray84
gray85	gray86	gray87	gray88	gray89
gray90	gray91	gray92	gray93	gray94
gray95	gray96	gray97	gray98	gray99
gray100				

The table for color names `grey`*N* looks the same.

Color-blind-safe colors

The Scheme function `universal-color` provides a set of eight colors (`https://jfly.uni-koeln.de/color`) designed to be unambiguous to people with dichromatism.

black orange skyblue bluegreen
yellow blue vermillion redpurple

B.8 The Emmentaler font

The Emmentaler font consists of two *subsets* of glyphs: "Feta", used for classical notation, and "Parmesan", used for ancient notation.

Any glyph within the Emmentaler font can be accessed directly by using text markup along with the name of the glyph (as shown in the tables below). For example;

```
g^\markup {\musicglyph "scripts.segno" }
```

or

```
\markup {\musicglyph "five"}
```

See Section 8.2 [Formatting text], page 320, for more information.

B.8.1 Modern glyph charts

All glyphs of the "Feta" subset.

Clef glyphs

clefs.C		clefs.C_change	
clefs.varC		clefs.varC_change	
clefs.F		clefs.F_change	
clefs.G		clefs.G_change	
clefs.GG		clefs.GG_change	
clefs.tenorG		clefs.tenorG_change	
clefs.percussion		clefs.percussion_change	
clefs.varpercussion		clefs .varpercussion_change	
clefs.tab		clefs.tab_change	

Time Signature glyphs

timesig.C44		timesig.C44.text	
timesig.C22		timesig.C22.text	
timesig.X			

Number glyphs

plus	+	comma	,
hyphen	-	equal	=
period	.	figuredash	‒
endash	–	parenleft	(
parenright	)	slash	/
zero	0	one	1
two	2	three	3
four	4	four.alt	4
five	5	six	6
seven	7	seven.alt	7
eight	8	nine	9
fixedwidth.zero	0	fixedwidth.one	1
fixedwidth.two	2	fixedwidth.three	3
fixedwidth.four	4	fixedwidth.four.alt	4
fixedwidth.five	5	fixedwidth.six	6
fixedwidth.seven	7	fixedwidth.seven.alt	7
fixedwidth.eight	8	fixedwidth.nine	9
fattened.zero	0	fattened.one	1
fattened.two	2	fattened.three	3
fattened.four	4	fattened.four.alt	4
fattened.five	5	fattened.six	6
fattened.seven	7	fattened.seven.alt	7
fattened.eight	8	fattened.nine	9
fattened.fixedwidth.zero	0	fattened.fixedwidth.one	1
fattened.fixedwidth.two	2	fattened.fixedwidth.three	3
fattened.fixedwidth.four	4	fattened.fixedwidth.four.alt	4

fattened.fixedwidth.five 

fattened.fixedwidth.six 

fattened.fixedwidth.seven

fattened.fixedwidth.seven.alt 

fattened.fixedwidth.eight

fattened.fixedwidth.nine 

u2007

u2009

u200A

Figured bass symbol glyphs

figbass.twoplus

figbass.fourplus 

figbass.fiveplus

figbass.sixstroked 

figbass.sevenstroked 

figbass.ninestroked 

Accidental glyphs

accidentals.sharp

accidentals.sharp.figbass 

accidentals.sharp.arrowup

accidentals.sharp.arrowdown 

accidentals.sharp.arrowboth

accidentals.sharp.slashslash.stem 

accidentals.sharp.slashslashslash.stemstem 

accidentals.sharp.slashslashslash.stem 

accidentals.sharp.slash.stem

accidentals.sharp.slashslash.stemstemstem 

accidentals.doublesharp 

accidentals.doublesharp.figbass 

accidentals.natural 

accidentals.natural.figbass 

accidentals.natural.arrowup 

accidentals.natural.arrowdown 

accidentals.natural.arrowboth 

accidentals.flat 

accidentals.flat.figbass 

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.figbass

accidentals.flatflat.slash

accidentals.sharp.sori

accidentals.flat.koron

accidentals.rightparen

accidentals.leftparen

Default note head glyphs

noteheads.uM2		noteheads.dM2	
noteheads.sM1		noteheads.s0	
noteheads.s1		noteheads.s2	

Special note head glyphs

noteheads.sM1double		noteheads.s0diamond	
noteheads.s1diamond		noteheads.s2diamond	
noteheads.s0triangle		noteheads.s1triangle	
noteheads.s2triangle		noteheads.s0slash	
noteheads.s1slash		noteheads.s2slash	
noteheads.s0cross		noteheads.s1cross	
noteheads.s2cross		noteheads.s2xcircle	
noteheads.d0arrow		noteheads.u0arrow	
noteheads.d2arrow		noteheads.u2arrow	
noteheads.s0harmonic		noteheads.s2harmonic	

Note head glyphs for shape notes

noteheads.s0do		noteheads.s1do	
noteheads.s2do		noteheads.s0doThin	
noteheads.s1doThin		noteheads.s2doThin	
noteheads.s0re		noteheads.s1re	
noteheads.s2re		noteheads.s0reThin	
noteheads.s1reThin		noteheads.s2reThin	
noteheads.s0mi		noteheads.s1mi	
noteheads.s2mi		noteheads.s0miMirror	
noteheads.s1miMirror		noteheads.s2miMirror	
noteheads.s0miThin		noteheads.s1miThin	

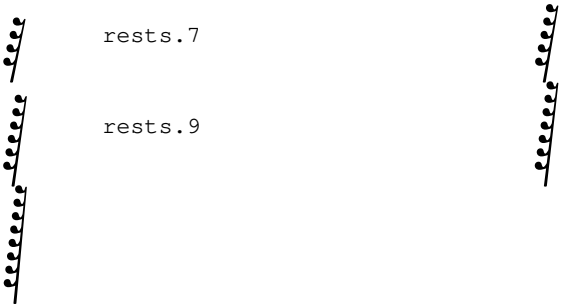
noteheads.s2miThin		noteheads.u0fa	
noteheads.d0fa		noteheads.u1fa	
noteheads.d1fa		noteheads.u2fa	
noteheads.d2fa		noteheads.u0faThin	
noteheads.d0faThin		noteheads.u1faThin	
noteheads.d1faThin		noteheads.u2faThin	
noteheads.d2faThin		noteheads.s0sol	
noteheads.s1sol		noteheads.s2sol	
noteheads.s0la		noteheads.s1la	
noteheads.s2la		noteheads.s0laThin	
noteheads.s1laThin		noteheads.s2laThin	
noteheads.s0ti		noteheads.s1ti	
noteheads.s2ti		noteheads.s0tiThin	
noteheads.s1tiThin		noteheads.s2tiThin	
noteheads.u0doFunk		noteheads.d0doFunk	
noteheads.u1doFunk		noteheads.d1doFunk	
noteheads.u2doFunk		noteheads.d2doFunk	
noteheads.u0reFunk		noteheads.d0reFunk	
noteheads.u1reFunk		noteheads.d1reFunk	
noteheads.u2reFunk		noteheads.d2reFunk	
noteheads.u0miFunk		noteheads.d0miFunk	
noteheads.u1miFunk		noteheads.d1miFunk	
noteheads.s2miFunk		noteheads.u0faFunk	
noteheads.d0faFunk		noteheads.u1faFunk	
noteheads.d1faFunk		noteheads.u2faFunk	
noteheads.d2faFunk		noteheads.s0solFunk	
noteheads.s1solFunk		noteheads.s2solFunk	

noteheads.s0laFunk		noteheads.s1laFunk	
noteheads.s2laFunk		noteheads.u0tiFunk	
noteheads.d0tiFunk		noteheads.u1tiFunk	
noteheads.d1tiFunk		noteheads.u2tiFunk	
noteheads.d2tiFunk		noteheads.s0doWalker	
noteheads.u1doWalker		noteheads.d1doWalker	
noteheads.u2doWalker		noteheads.d2doWalker	
noteheads.s0reWalker		noteheads.u1reWalker	
noteheads.d1reWalker		noteheads.u2reWalker	
noteheads.d2reWalker		noteheads.s0miWalker	
noteheads.s1miWalker		noteheads.s2miWalker	
noteheads.s0faWalker		noteheads.u1faWalker	
noteheads.d1faWalker		noteheads.u2faWalker	
noteheads.d2faWalker		noteheads.s0laWalker	
noteheads.s1laWalker		noteheads.s2laWalker	
noteheads.s0tiWalker		noteheads.u1tiWalker	
noteheads.d1tiWalker		noteheads.u2tiWalker	
noteheads.d2tiWalker			

Rest glyphs

rests.0		rests.1	
rests.0o		rests.1o	
rests.M3		rests.M2	
rests.M1		rests.M1o	
rests.2		rests.2classical	
rests.2z		rests.3	
rests.4		rests.5	

rests.6 rests.7

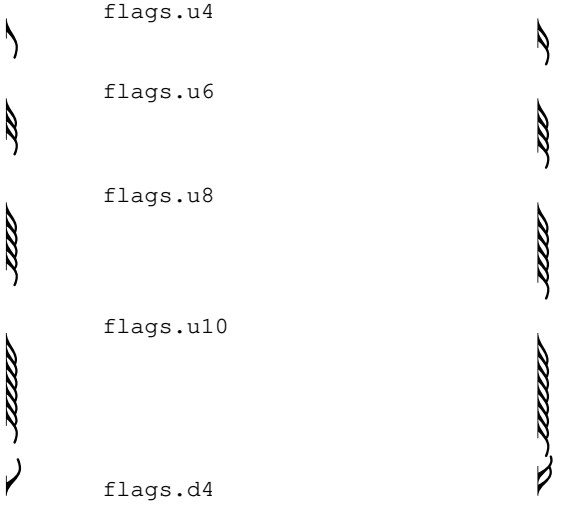


rests.8 rests.9

rests.10

Flag glyphs

flags.u3 flags.u4

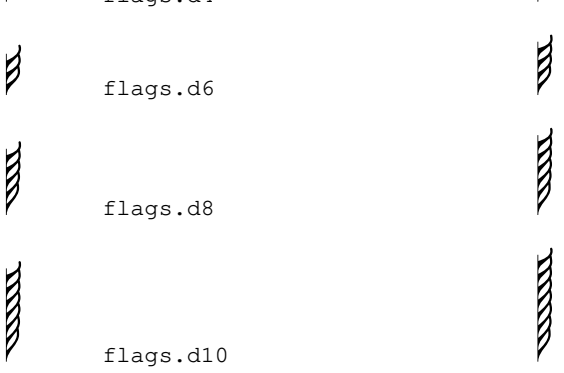


flags.u5 flags.u6

flags.u7 flags.u8

flags.u9 flags.u10

flags.d3 flags.d4

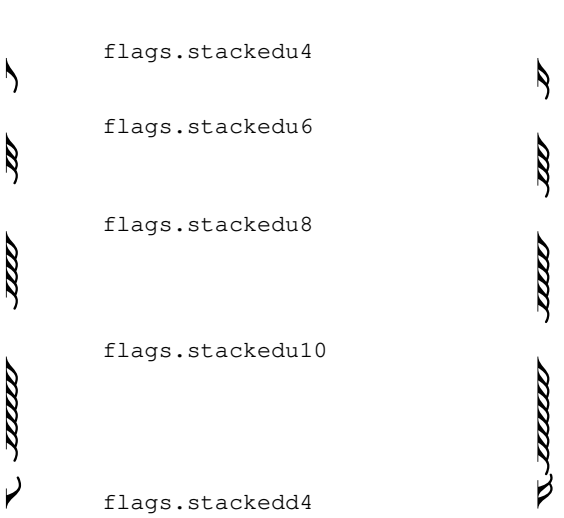


flags.d5 flags.d6

flags.d7 flags.d8

flags.d9 flags.d10

flags.stackedu3 flags.stackedu4



flags.stackedu5 flags.stackedu6

flags.stackedu7 flags.stackedu8

flags.stackedu9 flags.stackedu10

flags.stackedd3 flags.stackedd4

flags.stackedd5		flags.stackedd6	
flags.stackedd7		flags.stackedd8	
flags.stackedd9		flags.stackedd10	
flags.ugrace		flags.dgrace	

Dot glyphs

dots.dot	

Dynamic glyphs

space		f	
m		n	
p		r	
s		z	

Script glyphs

scripts.ufermata		scripts.dfermata	
scripts .uhenzeshortfermata		scripts .dhenzeshortfermata	
scripts .uhenzelongfermata		scripts .dhenzelongfermata	
scripts.ushortfermata		scripts.dshortfermata	
scripts .uveryshortfermata		scripts .dveryshortfermata	
scripts.ulongfermata		scripts.dlongfermata	
scripts.uverylongfermata		scripts.dverylongfermata	
scripts.thumb		scripts.sforzato	
scripts.espr		scripts.staccato	
scripts.ustaccatissimo		scripts.dstaccatissimo	
scripts.tenuto		scripts.uportato	
scripts.dportato		scripts.umarcato	

scripts.dmarcato		scripts.open	
scripts.halfopen		scripts.halfopenvertical	
scripts.stopped		scripts.uupbow	
scripts.dupbow		scripts.udownbow	
scripts.ddownbow		scripts.reverseturn	
scripts.turn		scripts.slashturn	
scripts.haydnturn		scripts.trill	
scripts.upedalheel		scripts.dpedalheel	
scripts.upedaltoe		scripts.dpedaltoe	
scripts.pedalheelcircle		scripts.flageolet	
scripts.segno		scripts.varsegno	
scripts.coda		scripts.varcoda	
scripts.rcomma		scripts.lcomma	
scripts.rvarcomma		scripts.lvarcomma	
scripts.raltcomma		scripts.laltcomma	
scripts.arpeggio		scripts.trill_element	
scripts.arpeggio .arrow.M1		scripts.arpeggio.arrow.1	
scripts.prall		scripts.mordent	
scripts.prallprall		scripts.prallmordent	
scripts.upprall		scripts.upmordent	
scripts.prallup		scripts.downprall	
scripts.downmordent		scripts.pralldown	
scripts.lineprall		scripts.bachschleifer	
scripts.caesura.curved		scripts.caesura.straight	
scripts.tickmark		scripts.snappizzicato	
scripts.ictus		scripts.uaccentus	

scripts.daccentus

scripts.usemicirculus

scripts.dsemicirculus

scripts.circulus

scripts.usignumcongruentiae 

scripts.dsignumcongruentiae

Arrowhead glyphs

arrowheads.open.01

arrowheads.open.0M1

arrowheads.open.11 

arrowheads.open.1M1 

arrowheads.close.01

arrowheads.close.0M1

arrowheads.close.11

arrowheads.close.1M1

Bracket-tip glyphs

brackettips.up 

brackettips.down 

Pedal glyphs

pedal.* 

pedal.M

pedal..

pedal.P 

pedal.d 

pedal.e 

pedal.Ped 

Accordion glyphs

accordion.discant 

accordion.dot

accordion.freebass 

accordion.stdbass 

accordion.bayanbass 

accordion.oldEE 

accordion.push 

accordion.pull 

Tie glyphs

ties.lyric.short

ties.lyric.default

B.8.2 Ancient glyph charts

All glyphs of the “Parmesan” subset.

Vaticana glyphs

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

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

Medicaea glyphs

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

noteheads.smedicaea .inclinatum

noteheads .smedicaea.punctum

noteheads .smedicaea.rvirga

noteheads .smedicaea.virga

Hufnagel glyphs

clefs.hufnagel.do		clefs.hufnagel.do_change	
clefs.hufnagel.fa		clefs.hufnagel.fa_change	
clefs.hufnagel.do.fa		clefs.hufnagel .do.fa_change	
custodes.hufnagel.u0		custodes.hufnagel.u1	
custodes.hufnagel.u2		custodes.hufnagel.d0	
custodes.hufnagel.d1		custodes.hufnagel.d2	
accidentals.hufnagelM1		noteheads .shufnagel.punctum	
noteheads .shufnagel.virga		noteheads.shufnagel.lpes	

Mensural glyphs

rests.M3mensural		rests.M2mensural	
rests.M1mensural		rests.0mensural	
rests.1mensural		rests.2mensural	
rests.3mensural		rests.4mensural	
clefs.mensural.c		clefs.mensural.c_change	
clefs.blackmensural.c		clefs.blackmensural .c_change	
clefs.mensural.f		clefs.mensural.f_change	
clefs.mensural.g		clefs.mensural.g_change	
custodes.mensural.u0		custodes.mensural.u1	
custodes.mensural.u2		custodes.mensural.d0	
custodes.mensural.d1		custodes.mensural.d2	
accidentals.mensural1		accidentals.mensuralM1	
flags.mensuralu03		flags.mensuralu13	
flags.mensuralu23		flags.mensurald03	
flags.mensurald13		flags.mensurald23	
flags.mensuralu04		flags.mensuralu14	

flags.mensuralu24

flags.mensurald14

flags.mensuralu05

flags.mensuralu25

flags.mensurald15

flags.mensuralu06

flags.mensuralu26

flags.mensurald16

timesig.mensural44

timesig.mensural32

timesig.mensural94

timesig.mensural68

timesig.mensural48

timesig.mensural24

noteheads.dM3mensural

noteheads.uM2mensural

noteheads.sM2ligmensural

noteheads.urM3mensural

noteheads
.srM3ligmensural

noteheads.drM2mensural

noteheads.srM1mensural

noteheads
.dM3semimensural

noteheads
.uM2semimensural

noteheads
.sM2semiligmensural

flags.mensurald04

flags.mensurald24

flags.mensuralu15

flags.mensurald05

flags.mensurald25

flags.mensuralu16

flags.mensurald06

flags.mensurald26

timesig.mensural22

timesig.mensural64

timesig.mensural34

timesig.mensural98

timesig.mensural68alt

noteheads.uM3mensural

noteheads.sM3ligmensural

noteheads.dM2mensural

noteheads.sM1mensural

noteheads.drM3mensural

noteheads.urM2mensural

noteheads
.srM2ligmensural

noteheads
.uM3semimensural

noteheads
.sM3semiligmensural

noteheads
.dM2semimensural

noteheads
.sM1semimensural

noteheads .urM3semimensural		noteheads .drM3semimensural	
noteheads .srM3semiligmensural		noteheads .urM2semimensural	
noteheads .drM2semimensural		noteheads .srM2semiligmensural	
noteheads .srM1semimensural		noteheads .uM3blackmensural	
noteheads .dM3blackmensural		noteheads .sM3blackligmensural	
noteheads .uM2blackmensural		noteheads .dM2blackmensural	
noteheads .sM2blackligmensural		noteheads .sM1blackmensural	
noteheads.s0mensural		noteheads.s1mensural	
noteheads.s2mensural		noteheads .s0blackmensural	

Neomensural glyphs

rests.M3neomensural		rests.M2neomensural	
rests.M1neomensural		rests.0neomensural	
rests.1neomensural		rests.2neomensural	
rests.3neomensural		rests.4neomensural	
clefs.neomensural.c		clefs.neomensural .c_change	
timesig.neomensural44		timesig.neomensural22	
timesig.neomensural32		timesig.neomensural64	
timesig.neomensural94		timesig.neomensural34	
timesig.neomensural68		timesig.neomensural98	
timesig.neomensural48		timesig.neomensural68alt	
timesig.neomensural24		noteheads.uM3neomensural	
noteheads.dM3neomensural		noteheads.uM2neomensural	
noteheads.dM2neomensural		noteheads.sM1neomensural	
noteheads .urM3neomensural		noteheads .drM3neomensural	
noteheads .urM2neomensural		noteheads .drM2neomensural	
noteheads .srM1neomensural		noteheads.s0neomensural	
noteheads.s1neomensural		noteheads.s2neomensural	

Petrucci glyphs

clefs.petrucci.c1 

clefs.petrucci.c1_change 

clefs.petrucci.c2 

clefs.petrucci.c2_change 

clefs.petrucci.c3 

clefs.petrucci.c3_change 

clefs.petrucci.c4 

clefs.petrucci.c4_change 

clefs.petrucci.c5 

clefs.petrucci.c5_change 

clefs.petrucci.f 

clefs.petrucci.f_change 

clefs.petrucci.g 

clefs.petrucci.g_change 

noteheads.s0petrucci 

noteheads.s1petrucci 

noteheads.s2petrucci 

noteheads.s0blackpetrucci 

noteheads.s1blackpetrucci 

noteheads.s2blackpetrucci 

Solesmes glyphs

noteheads.ssolesmes.incl.parvum

noteheads.ssolesmes.auct.asc

noteheads.ssolesmes.auct.desc

noteheads.ssolesmes.incl.auctum

noteheads.ssolesmes.stropha

noteheads.ssolesmes.stropha.aucta

noteheads.ssolesmes.oriscus

Kievan Notation glyphs

clefs.kievan.do 

clefs.kievan.do_change 

accidentals.kievan1 

accidentals.kievanM1 

scripts.barline.kievan 

dots.dotkievan

noteheads.sM2kievan 

noteheads.sM1kievan 

noteheads.s0kievan 

noteheads.d2kievan 

noteheads.u2kievan 

noteheads.s1kievan 
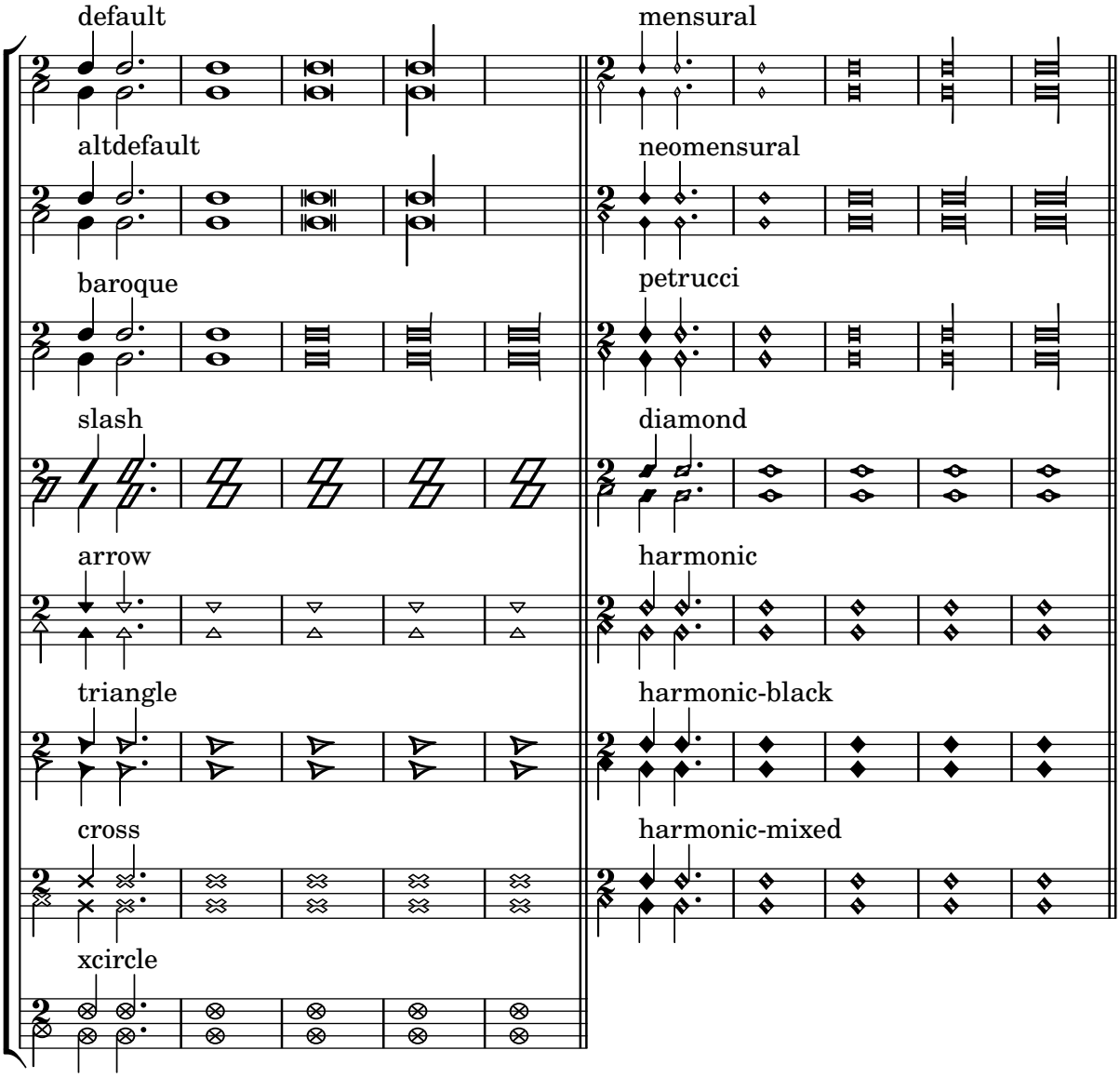
noteheads.sr1kievan

noteheads.d3kievan

noteheads.u3kievan

B.9 Note head styles
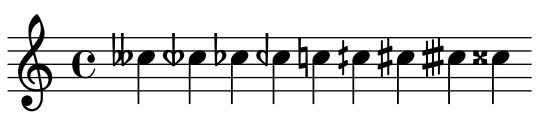
The following styles may be used for note heads. Shown are the note heads for a quarter, a dotted half, a whole, a breve, a longa, and a maxima. Not all note head styles support all possible note lengths (in a meaningful way).
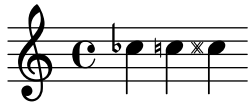


B.10 Accidental glyph sets

The following sets of accidental glyphs are available.
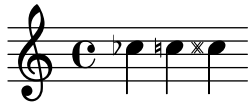
`standard-alteration-glyph-name-alist`
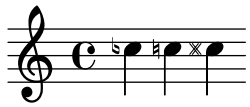


`alteration-hufnagel-glyph-name-alist`
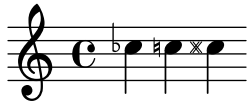


`alteration-medicaea-glyph-name-alist`
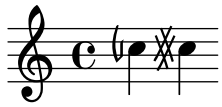


`alteration-vaticana-glyph-name-alist`



`alteration-mensural-glyph-name-alist`
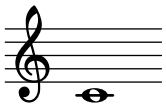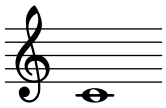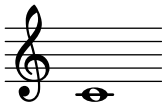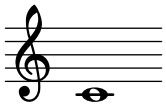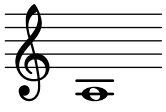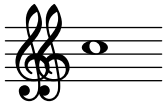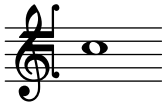


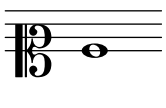
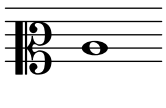
`alteration-kievan-glyph-name-alist`
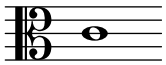


B.11 Clef styles

The following table shows all the clef styles possible (including the cases where *middle C* sits relative to the clef). For more modification possibilities like ottavation digits, see Section 1.3.1 [Clef], page 19.
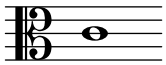
B.11.1 Standard clefs

Example	**Output**	**Example**	**Output**
`\clef G`		`\clef "G2"`	
`\clef treble`		`\clef violin`	
`\clef french`		`\clef GG`	
`\clef tenorG`			
`\clef soprano`		`\clef mezzosoprano`	

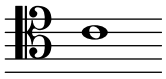
```
\clef C
```



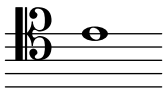
```
\clef alto
```



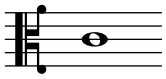
```
\clef tenor
```



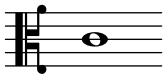
```
\clef baritone
```
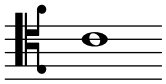


```
\clef varC
```



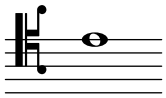
```
\clef altovarC
```
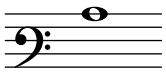


```
\clef tenorvarC
```
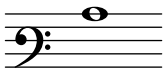


```
\clef baritonevarC
```
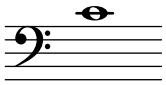


```
\clef varbaritone
```
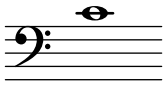


```
\clef baritonevarF
```
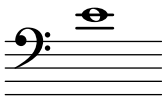


```
\clef F
```



```
\clef bass
```



```
\clef subbass
```



B.11.2 Percussion staff clef

Example

```
\clef percussion
```

Output



Example

```
\clef varpercussion
```

Output

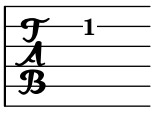


B.11.3 Tab staff clefs

Example

```
\new TabStaff {
  \clef tab
}
```

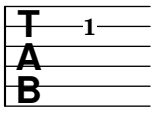
Output



Example

```
\new TabStaff {
  \clef moderntab
}
```

Output



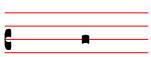
B.11.4 Ancient music clefs

Gregorian

Example

```
\clef "vaticana-do1"
```

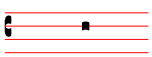
Output



Example

```
\clef "vaticana-do2"
```

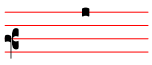
Output

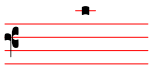


```
\clef "vaticana-do3"
```

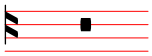


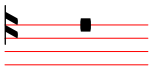
```
\clef "vaticana-fa1"
```

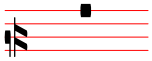


```
\clef "vaticana-fa2"
```


```
\clef "medicaea-do1"
```


```
\clef "medicaea-do2"
```


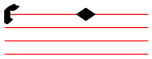
```
\clef "medicaea-do3"
```


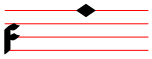
```
\clef "medicaea-fa1"
```


```
\clef "medicaea-fa2"
```

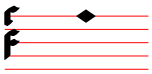

```
\clef "hufnagel-do1"
```


```
\clef "hufnagel-do2"
```


```
\clef "hufnagel-do3"
```


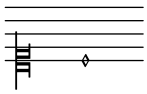
```
\clef "hufnagel-fa1"
```


```
\clef "hufnagel-fa2"
```

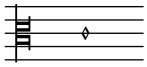

```
\clef "hufnagel-do-fa"
```


Mensural

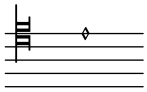
Example	Output	Example	Output

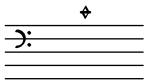
```
\clef "mensural-c1"
```


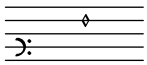
```
\clef "mensural-c2"
```

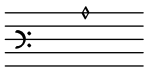

```
\clef "mensural-c3"
```


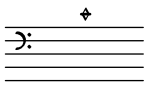
```
\clef "mensural-c4"
```

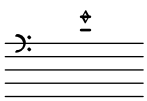

```
\clef "mensural-c5"
```
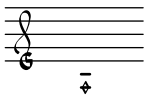

```
\clef "mensural-f"
```


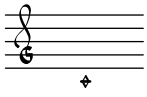
```
\clef "mensural-f2"
```
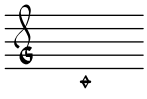

```
\clef "mensural-f3"
```


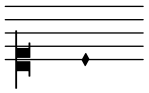
```
\clef "mensural-f4"
```


```
\clef "mensural-f5"
```
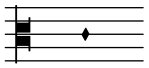

```
\clef "mensural-g1"
```


```
\clef "mensural-g2"
```


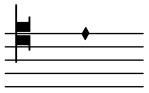
```
\clef "mensural-g"
```

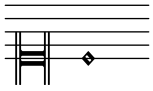

\clef "blackmensural-c1" 
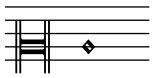
\clef "blackmensural-c2" 
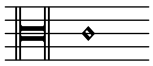
\clef "blackmensural-c3" 
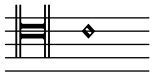
\clef "blackmensural-c4" 
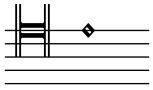
\clef "blackmensural-c5" 
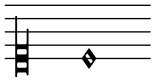
\clef "neomensural-c1" 

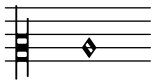
\clef "neomensural-c2" 

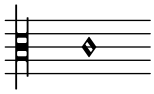
\clef "neomensural-c3" 
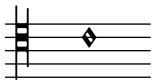
\clef "neomensural-c4" 

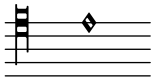
\clef "neomensural-c5" 

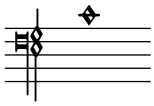
\clef "petrucci-c1" 

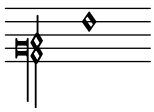
\clef "petrucci-c2" 

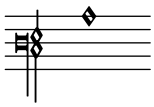
\clef "petrucci-c3" 

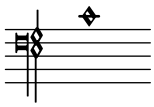
\clef "petrucci-c4" 
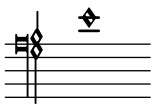
\clef "petrucci-c5" 
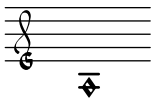
\clef "petrucci-f" 

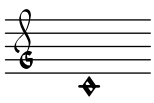
\clef "petrucci-f2" 

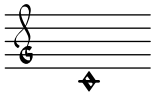
\clef "petrucci-f3" 

\clef "petrucci-f4" 

\clef "petrucci-f5" 

\clef "petrucci-g1" 

\clef "petrucci-g2" 

\clef "petrucci-g" 

Kievan

Example	Output
`\clef "kievan-do"`	

B.12 List of special characters

The following special characters references can be used; for more details, see Section 22.4.3 [ASCII aliases], page 648.

The HTML syntax is used and most of these references are the same as HTML. The rest of them are inspired by LaTeX.

The characters are boxed so that you can see their size. A small padding has been added between the character and the box for more readability.

¡	¡	¿	¿	&solidus;	⁄	&flq;	‹		
&frq;	›	&flqq;	«	&frqq;	»	&glq;	‚		
&grq;	‘	&glqq;	„	&grqq;	“	&elq;	‘		
&erq;	’	&elqq;	“	&erqq;	”				
						&nnbsp;			
‍		‌		·	·	•	•		
&copyright;	©	&registered;	®	&trademark;	™	†	†		
‡	‡	№	№	ª	ª	º	º		
¶	¶	§	§	°	°	№	№		
‰	‰	¦	¦	´	´	&acutedbl;	˝		
`	`	˘	˘	ˇ	ˇ	&cedilla;	¸		
&circumflex;	^	&diaeresis;	¨	&macron;	¯	&aa;	å		
&AA;	Å	&ae;	æ	&AE;	Æ	ä	ä		
Ä	Ä	&dh;	ð	&DH;	Ð	&dj;	đ		
&DJ;	Đ	&l;	ł	&L;	Ł	&ng;	ŋ		
&NG;	Ŋ	&o;	ø	&O;	Ø	&oe;	œ		
&OE;	Œ	ö	ö	Ö	Ö	&s;	ſ		
&ss;	ß	&th;	þ	&TH;	Þ	ü	ü		

Ü	Ü	+	+	−	−	×	×		
÷	÷	¹	¹	²	²	³	³		
&sqrt;	√	&increment;	Δ	&infty;	∞	∑	∑		
±	±	&bulletop;	∙	&partial;	∂	&neg;	¬		
&currency;	¤	$	$	€	€	&pounds;	£		
¥	¥	¢	¢						
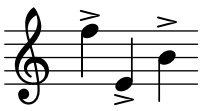
B.13 List of articulations

In LilyPond's internal logic, an 'articulation' is any object (other than dynamics) that may be attached directly after a rhythmic event: notes, chords; even silences and skips, or the empty chord construct <> (see Section "Structure of a note entry" in *Learning Manual*). Even slurs, fingerings and text scripts are technically articulations, although these are not shown here.

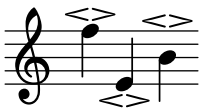
Therefore, the following lists include not only articulation marks, but also all other scripts in the Emmentaler font that may be attached to notes (the way an accent is entered as '`c'\accent`' or '`c'->`'). Each example shows the script in its two possible vertical positions: respectively *up* and *down*, as well as its default (*neutral*) position. See also [Script glyphs], page 914, for a more extensive list of glyphs, for use with the `\musicglyph` markup command as explained in Section 8.2.5 [Music notation inside markup], page 335.

B.13.1 Articulation scripts
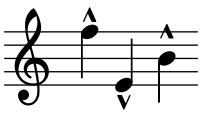
`\accent` or `->`
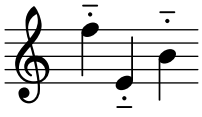


`\espressivo`
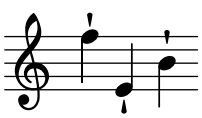


`\marcato` or `-^`
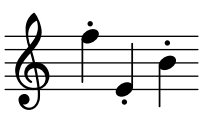


`\portato` or `-_`
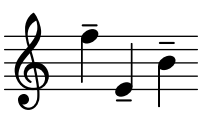


`\staccatissimo` or `-!`



`\staccato` or `-.`

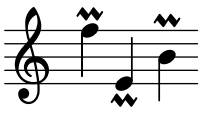


`\tenuto` or `--`
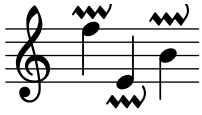


B.13.2 Ornament scripts
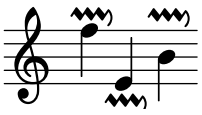
`\prall`
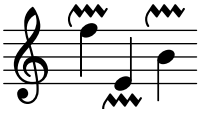


`\prallup`
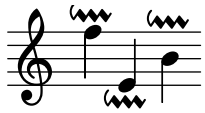


`\pralldown`
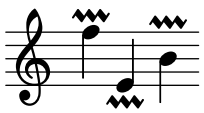


`\upprall`
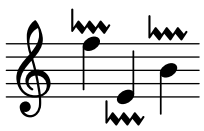


\downprall
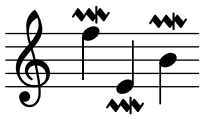


\prallprall



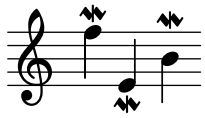
\lineprall
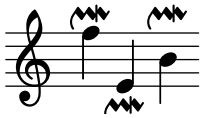


\prallmordent
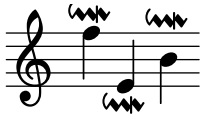


\mordent
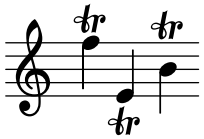


\upmordent
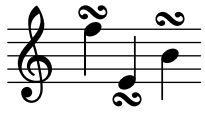


\downmordent
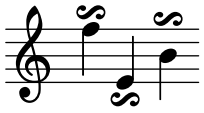


\trill
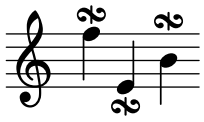


\turn



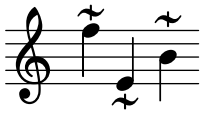
\reverseturn



\slashturn
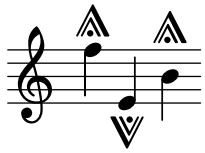


\haydnturn
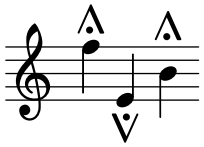


B.13.3 Fermata scripts
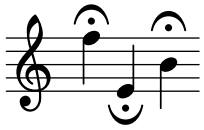
\veryshortfermata



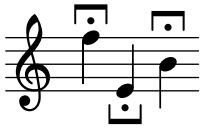
\shortfermata



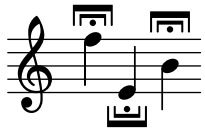
\fermata
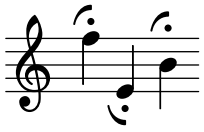


\longfermata



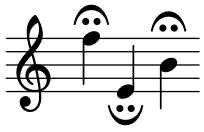
\verylongfermata



\henzeshortfermata

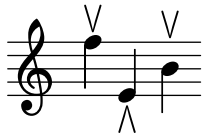


\henzelongfermata
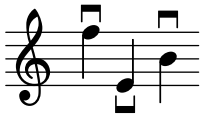


B.13.4 Instrument-specific scripts

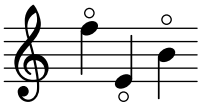
\upbow



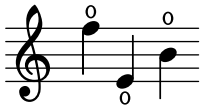
\downbow
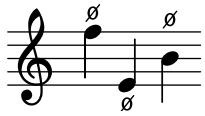


\flageolet
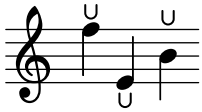


\open



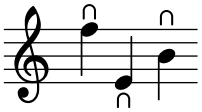
\halfopen



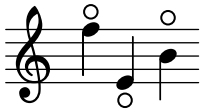
\heel
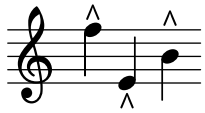


\varheel



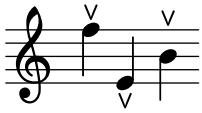
\heelcircle



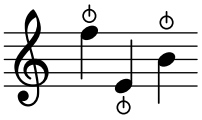
\toe



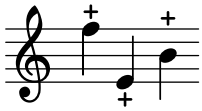
\vartoe



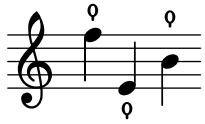
\snappizzicato



\stopped or -+
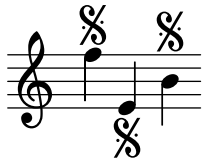


\thumb
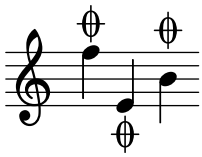


B.13.5 Repeat sign scripts

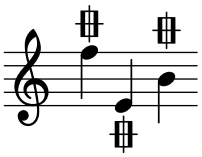
\segno



\coda

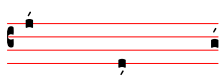


\varcoda



B.13.6 Ancient scripts

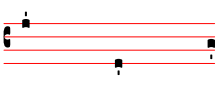
\accentus



\circulus



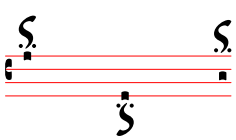
\ictus



\semicirculus
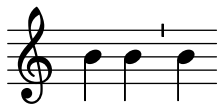


\signumcongruentiae
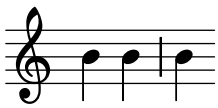


B.14 List of breath marks
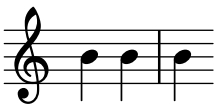
'chantquarterbar
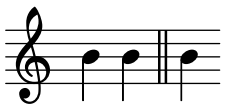


'chanthalfbar
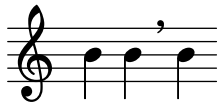


'chantfullbar
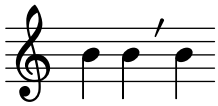


'chantdoublebar
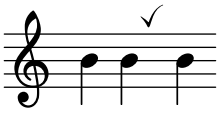


'comma



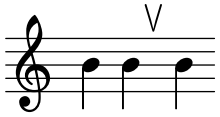
'varcomma
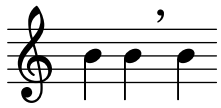


'tickmark
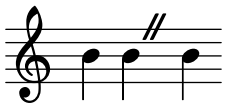


'upbow
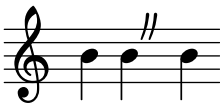


'outsidecomma
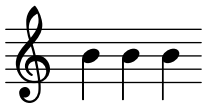


'caesura



'curvedcaesura
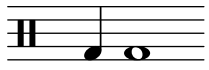


'spacer
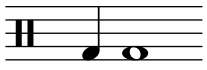


B.15 Percussion notes
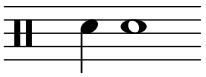
bassdrum
bd
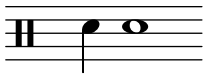


acousticbassdrum
bda
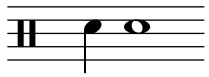


snare
sn
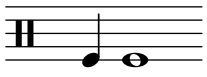


acousticsnare
sna
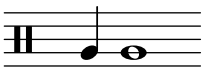


electricsnare
sne
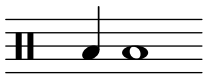


lowfloortom
tomfl
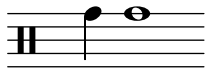


highfloortom
tomfh
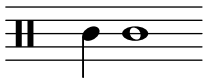


lowtom
toml
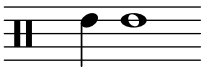


hightom
tomh



lowmidtom
tomml



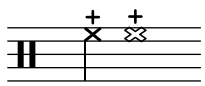
himidtom
tommh



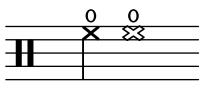
hihat
hh
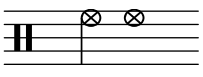


closedhihat
hhc



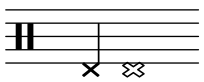
openhihat
hho



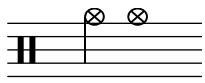
halfopenhihat
hhho



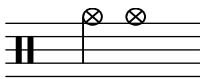
pedalhihat
hhp
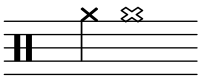


crashcymbal
cymc
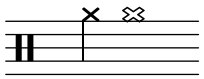


crashcymbala
cymca
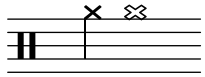


crashcymbalb
cymcb
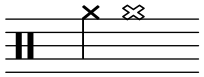


ridecymbal
cymr
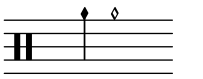


ridecymbala
cymra
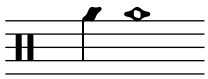


ridecymbalb
cymrb
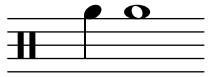


chinesecymbal
cymch
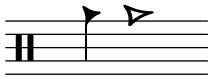


splashcymbal
cyms



ridebell
rb



cowbell
cb
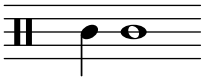


hibongo
boh

openhibongo
boho
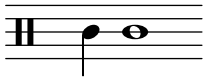
mutehibongo
bohm

lobongo
bol
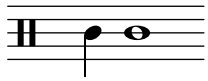
openlobongo
bolo
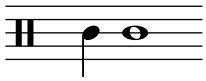


mutelobongo
bolm
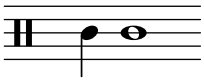


hiconga
cgh
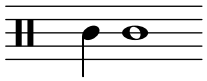


openhiconga
cgho
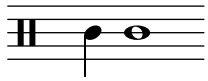


mutehiconga
cghm
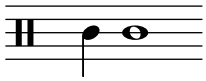


loconga
cgl
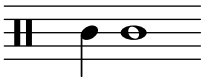


openloconga
cglo
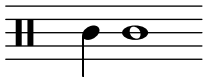


muteloconga
cglm
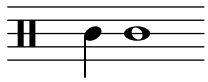


hitimbale
timh
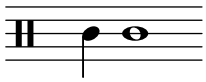


lotimbale
timl
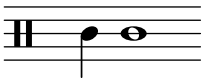


hiagogo
agh
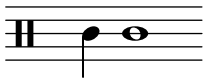


loagogo
agl
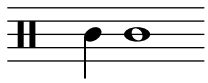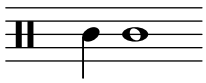


sidestick
ss
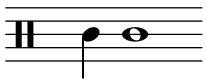


hisidestick
ssh
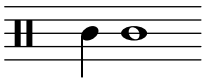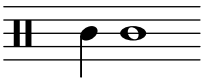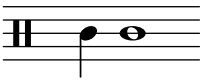










losidestick
ssl
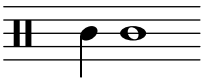


guiro
gui
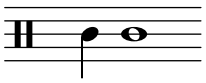


shortguiro
guis
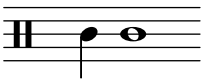


longguiro
guil
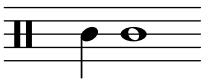


cabasa
cab
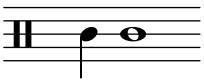


maracas
mar
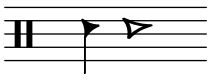


shortwhistle
whs
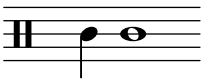


longwhistle
whl
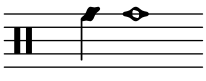


handclap
hc
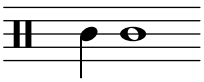


tambourine
tamb



vibraslap
vibs



tamtam
tt
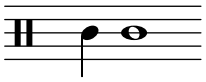


claves
cl

hiwoodblock
wbh
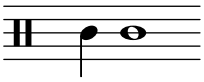
lowoodblock
wbl

opencuica
cuio
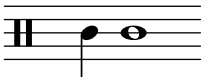
mutecuica
cuim
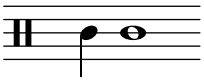


triangle
tri
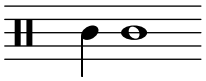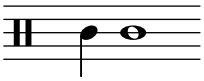


opentriangle
trio
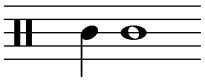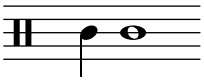


mutetriangle
trim











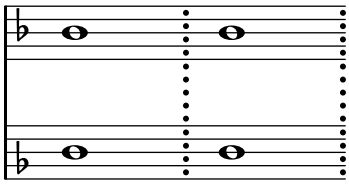
B.16 List of bar lines

Each system following shows the appearance of a particular bar type at the beginning, middle, and end of a line.

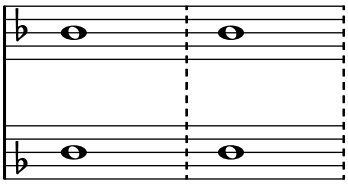
For more information, see Section 2.5.1 [Bar lines], page 121, and Section 2.5.2 [Automatic bar lines], page 131.

Simple bar lines

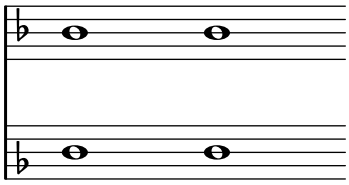
";"



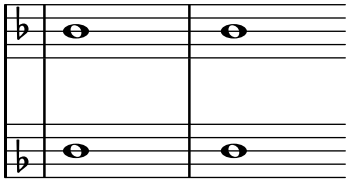
"!"



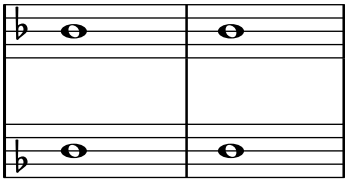
""



"|-s"

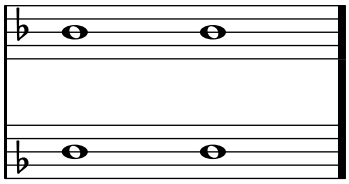


"|"

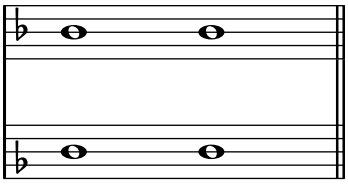


End-of-line bar lines

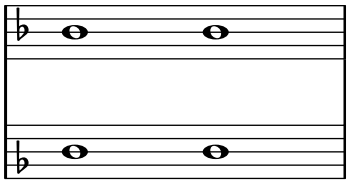
"x-."



"x-||"

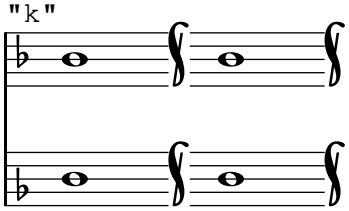


"x-|"

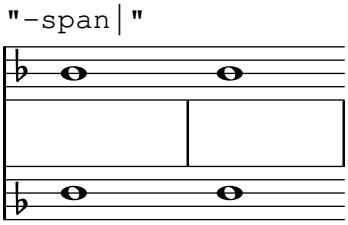


Chant bar lines

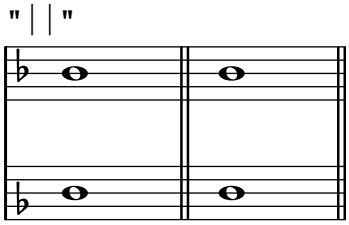
"k"



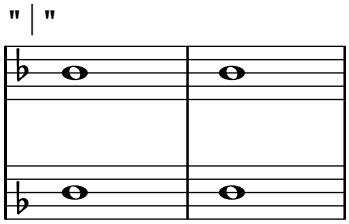
"-span|"
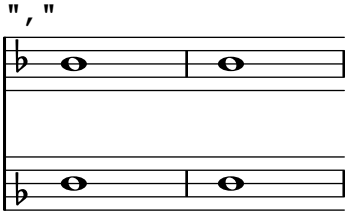


"||"



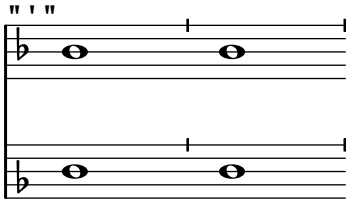
"|"



","

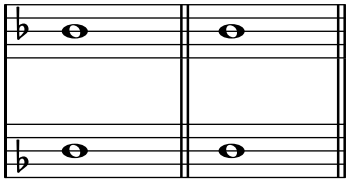


"'"

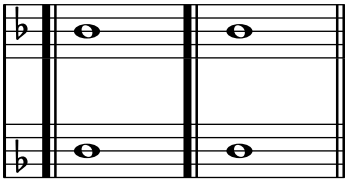


Section bar lines

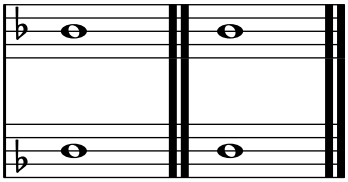
"||"



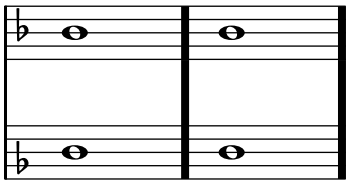
".|-||"



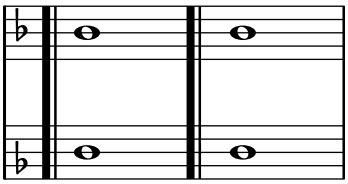
".."



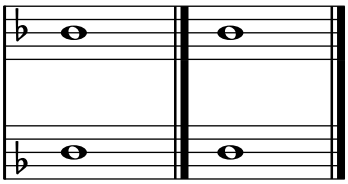
"."



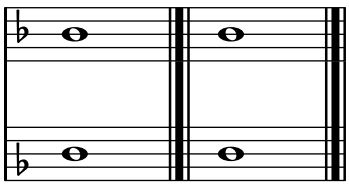
".|-|"



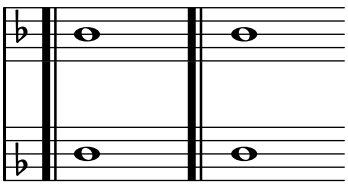
"|."



"|.|"

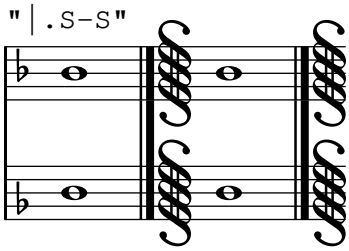


".|"

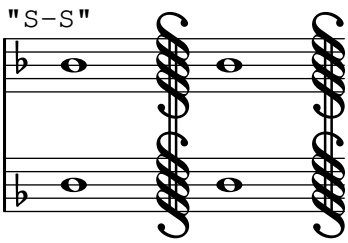


Segno bar lines

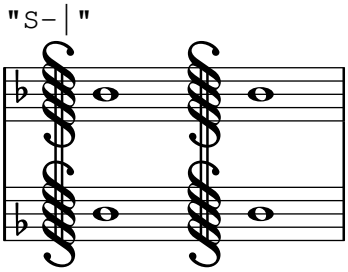
"|.S-S"



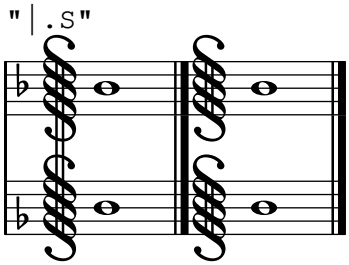
"S-S"



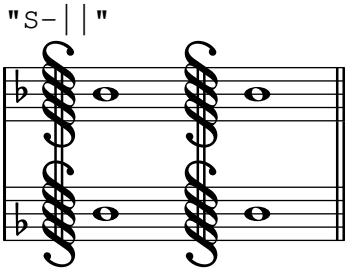
"S-|"



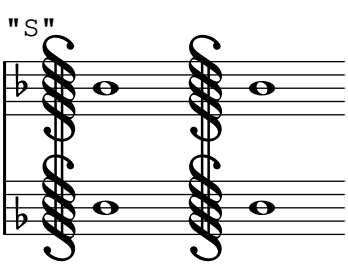
"|.S"



"S-||"

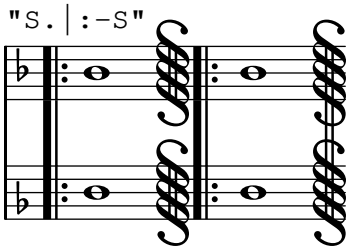


"S"

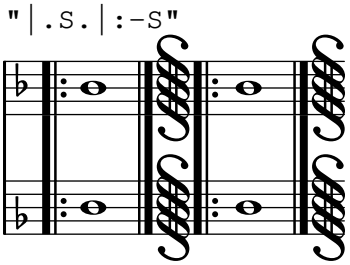


Start-repeat bar lines

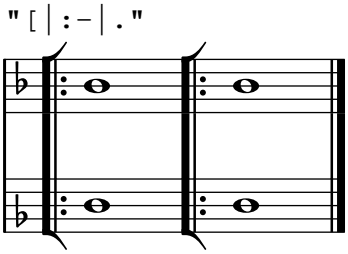
"s.|:-s"



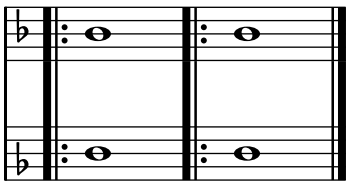
"|.s.|:-s"



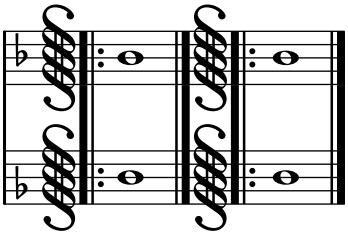
"[|:-|."



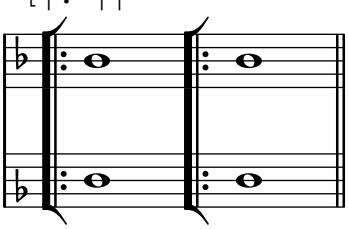
".|:-|."



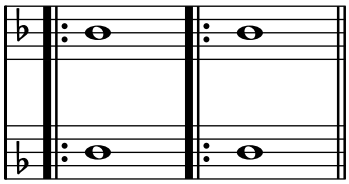
"|.s.|:"



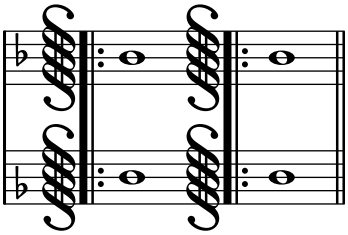
"[|:-||"



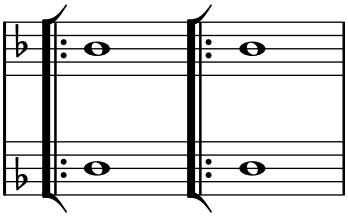
".|:-||"



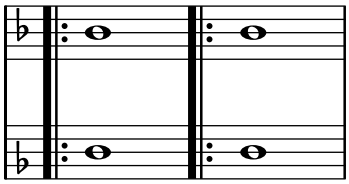
"s.|:-||"



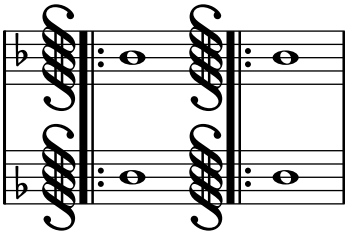
"[|:-|"



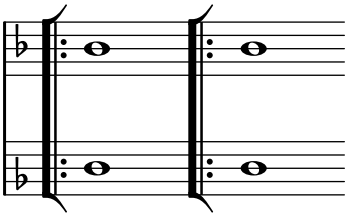
".|:-|"



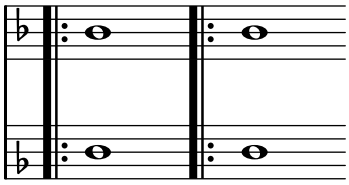
"s.|:-|"



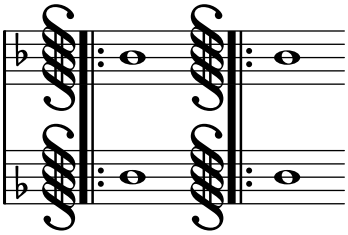
"[|:"



".|:"

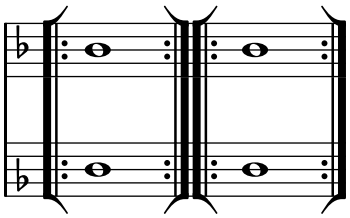


"s.|:"

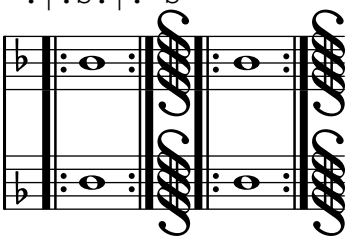


Double-repeat bar lines

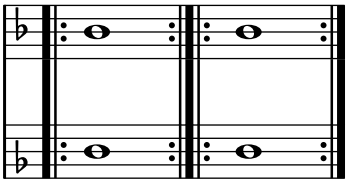
":|][|:"



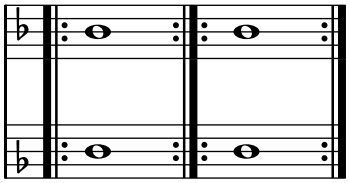
":|.S.|:-S"



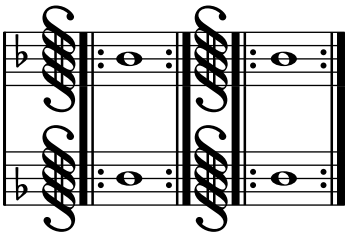
":|.|:"



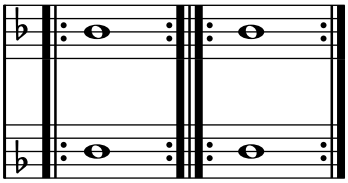
":|.:"



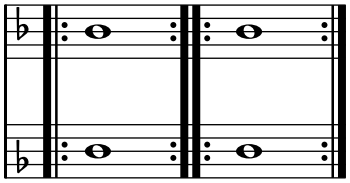
":|.S.|:"



":.|.:"

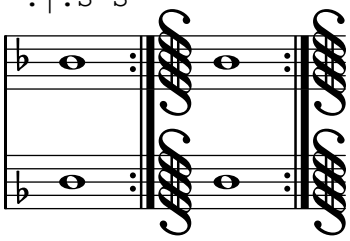


":..:"

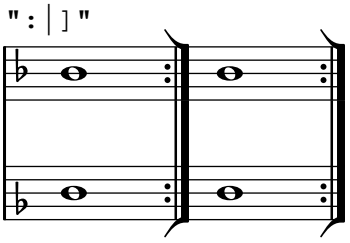


End-repeat bar lines

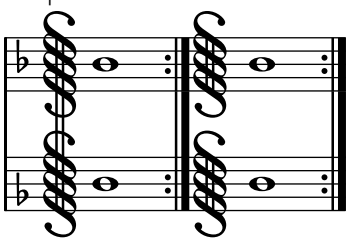
":|.S-S"



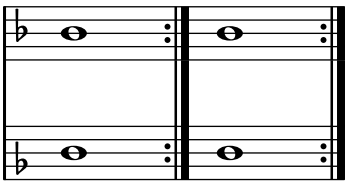
":|]"



":|.S"



":|."



B.17 Default values for `outside-staff-priority`

The following table shows the default `outside-staff-priority` of all outside-staff grobs. Objects with smaller values are placed closer to the staff.

Grob	Priority
AccidentalSuggestion	0
BassFigureAlignmentPositioning	25
MultiMeasureRestScript	40
TrillSpanner	50
BarNumber	100
LigatureBracket	200
DynamicLineSpanner	250
TextSpanner	350
OttavaBracket	400
MultiMeasureRestText	450
TextScript	450
CombineTextScript	475
InstrumentSwitch	500
VoltaBracketSpanner	600
MeasureCounter	750
MeasureSpanner	750
HorizontalBracket	800
SostenutoPedalLineSpanner	1000
SustainPedalLineSpanner	1000
UnaCordaPedalLineSpanner	1000
CenteredBarNumberLineSpanner	1200
TextMark	1250
MetronomeMark	1300
JumpScript	1350
CodaMark	1400
SegnoMark	1400
SectionLabel	1450
RehearsalMark	1500

B.18 Default values for `script-priority`

The following table shows the default `script-priority` of all script-related grobs (starting with an uppercase letter) and script names (starting with a lowercase letter). Objects with smaller values are placed closer to the staff.

The `Fingering`, `StringNumber`, and `StrokeFinger` grobs, if part of a chord, add the vertical position of the associated note head to get the final priority value. The `script-priority` values of other scripts part of a chord stay unmodified.

For scripts not part of a chord, the current event position is added instead. This means that for scripts with the same `script-priority` value, the first one attached to a note head gets the lowest priority value.

Grobs not listed here (and which are related to scripts) get 200 as the default `script-priority` value.

Grob or Script	Priority
AccidentalPlacement	-100
accentus	-100
circulus	-100
ictus	-100
semicirculus	-100
staccato	-100
tenuto	-50

AccidentalSuggestion	0
Arpeggio	0
ChordBracket	0
ChordSlur	0
flageolet	50
Fingering	100
StrokeFinger	125
StringNumber	150
trill	150
fermata	175
henzelongfermata	175
henzeshortfermata	175
longfermata	175
shortfermata	175
verylongfermata	175
veryshortfermata	175
downbow	180
upbow	180
CombineTextScript	200
TextScript	200

B.19 Grobs and their break-align symbols

The ordering and spacing of notational elements like clefs, time signatures, bar lines, etc., is determined using certain Scheme symbols, the so-called *break-align symbols*. More precisely, their ordering is determined by lists of break-align symbols in the `BreakAlignment.break-align-orders` vector (see Section "break-alignment-interface" in *Internals Reference*); their spacing is controlled by the `space-alist` properties of the respective grobs, whose elements are in turn indexed by break-align symbols: For instance, the spacing of a `Clef` grob to a bar line following it is determined by the value of `Clef.space-alist.staff-bar`.

This table lists the default correspondence between break-align symbols and grobs:

break-align symbol	**used by grob(s)**
ambitus	Ambitus
breathing-sign	BreathingSign
clef	Clef
cue-clef	CueClef
cue-end-clef	CueEndClef
custos	Custos
key-cancellation	KeyCancellation
key-signature	KeySignature
left-edge	LeftEdge
optional-material-end-bracket	OptionalMaterialBracket
optional-material-start-bracket	OptionalMaterialBracket
signum-repetitionis	SignumRepetitionis
staff-bar	SpanBar, DoublePercentRepeat, Divisio, and BarLine
staff-ellipsis	StaffEllipsis
time-signature	TimeSignature

Furthermore, there are three abstract break-align symbols that may be used in `space-alist` definitions, namely `first-note`, `next-note` and `right-edge`. For details, see the documentation of the `space-alist` grob property in Section "break-aligned-interface" in *Internals Reference*.

The next table lists the break-align symbol used by each grob:

Grob	break-align symbol(s)
Ambitus	ambitus
BarLine	staff-bar
BreathingSign	breathing-sign
Clef	clef
CueClef	cue-clef
CueEndClef	cue-end-clef
Custos	custos
Divisio	staff-bar
DoublePercentRepeat	staff-bar
KeyCancellation	key-cancellation
KeySignature	key-signature
LeftEdge	left-edge
OptionalMaterialBracket	optional-material-start-bracket and optional-material-end-bracket
SignumRepetitionis	signum-repetitionis
SpanBar	staff-bar
StaffEllipsis	staff-ellipsis
TimeSignature	time-signature

B.20 Technical glossary

This glossary lists some 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 that 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, thus 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 make user-level Scheme code control many low-level actions.

closure — In Scheme, a *closure* is created when a function, usually a lambda expression (i.e., an ad-hoc, anonymous function), 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, which in turn are used in the calculation. One useful property of closures is the preservation of internal variable values between invocations, thus permitting state to be maintained.

glyph — A *glyph* is a particular graphical representation of a typographic character, or a combination of two (or more) 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: Section 8.3 [Fonts], page 337, Section 22.4 [Special characters], page 647.

grob — LilyPond objects that 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: Section "Objects and interfaces" in *Learning Manual*, Section "Properties of layout objects" in *Learning Manual*.
Section B.24 [Naming conventions], page 973.
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 below for mutable objects.

interface Actions and properties that are common to a number of grobs are grouped together in an object called a `grob-interface`, or just *interface* for short.

See also: Section B.24 [Naming conventions], page 973, Section 34.2 [Layout interfaces], page 757.
Section "Objects and interfaces" in *Learning Manual*, Section "Properties found in interfaces" in *Learning Manual*.
Section "Graphical Object Interfaces" in *Internals Reference*.

lexer A *lexer* is a program that 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 (see below). The LilyPond lexer is built using the `flex` program from the lexer file `lily/lexer.ll`, which contains the lexical rules. This file is part of the source code and 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 above for immutable objects.

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 file `lily/parser.yy`. During compilation, this file gets processed by a parser generator, `bison`, to build the parser. It is part of the source code and 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 persists 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.

Among others, the following parser variables are used by LilyPond:

`afterGraceFraction`

`input-language`

`musicQuotes`

`output-count`

`output-suffix`

`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 that have mutable and immutable property alists 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.

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 file `lily/includes/smob.hh`.

spanner — *Spanners* are a class of grobs that are not horizontally fixed on one point of the score but extend from one point to another. Examples include beams, ties, and slurs, as well as hairpins and staff lines. Whereas non-spanners can only be broken into at most two visible pieces (for example, a clef duplicated at the end of the line and the beginning of the next line), spanners are broken into as many pieces as required by their start and end points (such as a long crescendo extending on three systems, or staff lines, which always span the whole score).

Technically, spanners are defined as grobs having the `spanner-interface`; on the C++ side of LilyPond, they are instances of the `Spanner` subclass of `Grob`. The left and right bounds of a spanner can be retrieved and set using `ly:spanner-bound` and `ly:spanner-set-bound!`, respectively. The bounds are always items. The X parent of a spanner has little musical sense, but is usually set to the left bound.

See also: Section 36.4 [Spanners], page 776.
Section “All layout objects” in *Internals Reference*, Section “spanner-interface” in *Internals Reference*.

stencil — An instance of the `Stencil` class holds the information required to print a typographical object, a *stencil*. It is a simple smob containing a confining box, which defines the vertical and horizontal extents of the object, and a Scheme expression, which prints the object when evaluated. Stencils may be combined to form more

complex stencils defined by a tree of Scheme expressions, which in turn are 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 Section "grob-interface" in *Internals Reference*.

B.21 Available music functions

In the following, the first argument of a function is printed with a slanted typeface, followed by the first argument's type in parentheses. Then comes the second argument, followed by the second argument's type, again in parentheses, and so on. After the arrow the type of the return value is printed.

If an argument and its type is enclosed in brackets, the argument is optional and can be omitted, indicating that LilyPond should use a default value instead. However, if a function's last argument is tagged as optional, it *cannot* be omitted: you have to pass `\default` if you do not want to explicitly specify a value. See Section "Scheme function usage" in *Extending* for more details on how optional arguments are handled.

`\absolute` *music* (music) ⇒ music
Make *music* absolute.

This does not actually change the music itself but rather hides it from surrounding `\relative` (page 957) and `\fixed` (page 948) commands.

`\acciaccatura` *music* (music) ⇒ music
Create an acciaccatura from *music*.

`\accidentalStyle` *style* (symbol list) ⇒ music
Set accidental style to *style*.

style is a (predefined) symbol list like `piano-cautionary`; see Section 1.3.5 [Automatic accidentals], page 31, for the available styles. If it is preceded by a context name, the settings are applied to that context (example: `Staff.piano-cautionary`). Otherwise, the context defaults to `Staff`, except for piano styles, which use `GrandStaff` as a context.

`\addChordShape` *key-symbol* (symbol) *tuning* (pair) *shape-definition* (string or pair) ⇒ void
Add *shape-definition* as a chord shape.

It gets added to the `chord-shape-table` hash with the key `(cons` *`key-symbol tuning`*`)`.

`\addInstrumentDefinition` *name* (string) *lst* (list) ⇒ void
Create instrument *name* with properties *lst*.

This function is deprecated.

`\addQuote` *name* (string) *music* (music) ⇒ void
Define *music* as a quotable music expression named *name*.

`\addToTagGroup` *tag-group* (symbol list) *more-tags* (symbol list) ⇒ void
Add *more-tags* to the tag group identified by *tag-group*.

`\after` *delta* (duration-or-music) *ev* (music) *mus* (music) ⇒ music
Add music *ev* with a delay of *delta* after the onset of *mus*.

delta can be specified as music; this is intended for specifying compound durations like `{ 2 4. }` and does not typeset the music.

ev is usually a post-event.

\afterGrace [*fraction* (non-negative rational, fraction, or moment)] *main* (music) *grace* (music) ⇒ music
: Create *grace* as grace notes after a *main* music expression.

  The musical position of the grace expression is after a given fraction of the main note's duration has passed. If optional argument *fraction* is omitted, the fraction value is taken from `afterGraceFraction`, defaulting to 3/4.

\allowPageTurn ⇒ music
: Allow a page turn.

  May be used at top level (i.e., between scores or markups), or inside a score.

\allowVoltaHook *bar* (string) ⇒ void
: Allow the volta bracket hook being drawn over bar line *bar*.

\alterBroken *property* (key list or symbol) *arg* (list) *target* (key list or music) ⇒ music
: Override *property* for pieces of broken spanner *target* with *arg*.

  target may either be music in the form of a starting spanner event, or a symbol list of the form *`Context.Grob`* or just *`Grob`*. If *target* is in the form of a spanner event, *property* may also have the form *`Grob.property`* for specifying a directed tweak.

  arg is a list of values, one for each broken piece.

\ambitusAfter *target* (symbol) ⇒ music
: Move the ambitus after the break-align symbol *target*.

\appendToTag *tag* (symbol) *more* (music) *music* (music) ⇒ music
: Append *more* to *music* tagged with *tag*.

  A post-event can be added to the articulations of rhythmic events or chords; other expressions may be added to chords, sequential or simultaneous music.

\appendToTagMarkup *tag* (symbol) *more* (markup) *music* (music) ⇒ music
: Append *more* to every markup in *music* tagged with *tag*.

\applyContext *proc* (procedure) ⇒ music
: Modify context properties with Scheme procedure *proc*.

\applyMusic *func* (procedure) *music* (music) ⇒ music
: Apply procedure *func* to *music*.

\applyOutput *target* (symbol list or symbol) *proc* (procedure) ⇒ music
: Apply function `proc` to every layout object matched by *target*.

  target takes the form *`Context`* or *`Context.Grob`*.

\appoggiatura *music* (music) ⇒ music
: Create an appoggiatura from *music*.

\approximatePitch *note* (music) ⇒ music
: Indicate that the pitch of *note* approximates a pitch that cannot be known exactly, such as the highest note a singer can sing.

\arpeggioArrowDown ⇒ music
: Let `\arpeggio` produce a wavy line with a down arrow.

\arpeggioArrowUp ⇒ music
: Let `\arpeggio` produce a wavy line with an up arrow.

\arpeggioBracket ⇒ music
: Let `\arpeggio` produce a square bracket.

  Note: For a bracket designating a non-arpeggiated chord, it is better to use `\nonArpeggiato` than to use `\arpeggio` and alter the appearance.

`\arpeggioNormal` ⇒ music
: Let `\arpeggio` produce a wavy line without an arrow.

`\arpeggioParenthesis` ⇒ music
: Let `\arpeggio` produce a vertical slur.

  Note: For a vertical slur designating a quasi-non-arpeggiated chord, it is better to use `\chordSlur` than to use `\arpeggio` and alter the appearance.

`\arpeggioParenthesisDashed` ⇒ music
: Let `\arpeggio` produce a dashed vertical slur.

  Note: For a vertical slur designating a quasi-non-arpeggiated chord, it is better to use `\chordSlur` than to use `\arpeggio` and alter the appearance.

`\assertBeamQuant` *l* (pair) *r* (pair) ⇒ music
: Testing function: check whether the beam quants *l* and *r* are correct.

`\assertBeamSlope` *comp* (procedure) ⇒ music
: Testing function: check whether the slope of the beam is the same as *comp*.

`\atLeft` *m* (music) ⇒ post-event
: Set `side-axis` to `X` and `direction` to `LEFT` for *mus*.

`\atRight` *m* (music) ⇒ post-event
: Set `side-axis` to `X` and `direction` to `RIGHT` for *mus*.

`\augmentum` *expr* (music) ⇒ music
: Add augmentum dots (*morae*) to Gregorian chant *expr*.

`\autoChange` [*pitch* (pitch)] [*clef-1* (context modification)] [*clef-2* (context modification)] *music* (music) ⇒ music
: Make voices for *music* that switch between staves automatically.

  The optional argument *pitch* specifies where to switch staves; the default is `c'`. Optional arguments *clef-1* and *clef-2* specify which clefs to use; this only works for implicitly instantiated staves.

  Example:

```
\autoChange d' \with { \clef alto } { g4 d' g' }
```

`\balloonGrobText` *grob-name* (symbol) *offset* (pair of numbers) *text* (markup) ⇒ music
: Attach *text* to *grob-name* at offset *offset*.

  Use like `\once` (page 952).

`\balloonText` *offset* (pair of numbers) *text* (markup) ⇒ post-event
: Attach *text* at *offset*.

  Use like `\tweak` (page 962).

`\bar` *type* (string) ⇒ music
: Insert a bar line of type *type*, overriding any automatic bar lines.

`\barNumberCheck` *n* (integer) ⇒ music
: Print a warning if the current bar number is not *n*.

`\beamExceptions` *music* (music) ⇒ any type
: Set beam exceptions.

  This function extracts a value suitable for setting `Timing.beamExceptions` from the given pattern with explicit beams in *music*. A bar check '`|`' has to be used between bars of patterns in order to reset the timing.

`\bendAfter` *delta* (real number) ⇒ post-event
: Create a fall or doit of pitch interval *delta*.

\bendHold *mus* (music) ⇒ post-event
: Set `BendSpanner.style` to `'hold` for *mus*.

\bendStartLevel *idx* (non-negative, exact integer) *mus* (music) ⇒ post-event
: Set `Bendspanner.details.successive-level` to *idx* for *mus*.

\bookOutputName *newfilename* (string) ⇒ void
: Direct output for the current book block to *newfilename*.

  This is equivalent to setting `output-filename` in the current book's \paper block.

\bookOutputSuffix *newsuffix* (string) ⇒ void
: Set the output file name suffix for the current book block to *newsuffix*.

  This is equivalent to setting `output-suffix` in the current book's \paper block.

\breakAlignInsert [*positions* (vector)] *symbol-to-move* (symbol) *sorting-symbol* (symbol) *anchor-symbol* (symbol) ⇒ music
: Re-order break-align symbols.

  This function is for tasks like putting a clef change after the bar line, which would be accomplished by \breakAlignInsert `clef after staff-bar`.

  sorting-symbol can be `before` or `after`. Depending on its value, the break-align symbol *symbol-to-move* gets moved directly before/after the symbol *anchor-symbol* in the `break-align-orders` vector of `Score.BreakAlignment`.

  The available break-align symbols and their corresponding grobs are listed in Section "Grobs and their break-align symbols" in *Notation Reference*.

  The optional argument *positions* determines whether the specified change should become visible at the beginning/the end/the middle of a line. Possible values are: `#all-visible` (default), `#begin-of-line-invisible`, `#center-invisible`, `#end-of-line-invisible`, `#begin-of-line-visible`, `#center-visible`, `#end-of-line-visible`, `#all-invisible`.

\breathe ⇒ music
: Insert a breath mark.

\cadenzaOff ⇒ music
: Resume advancing `Timing.measurePosition`.

  This sets `Timing.timing` to `#t`.

\cadenzaOn ⇒ music
: Stop advancing `Timing.measurePosition`.

  This sets `Timing.timing` to `#f`.

  Compare to \senzaMisura (page 958).

\caesura ⇒ music
: Insert a caesura.

\chordRepeats [*event-types* (list)] *music* (music) ⇒ music
: Extend 'q' to also repeat articulation.

  This function walks through *music*, putting the notes of the previous chord into repeat chords, as well as an optional list of *event-types* such as `#'(string-number-event)`.

\clef *type* (string) ⇒ music
: Set the current clef to *type*.

\codaMark [*num* (non-negative, exact integer)] ⇒ music
: Create a coda mark.

num may be 1 for the first mark, 2 for the second, etc., or it may be `\default` to use the next number in sequence automatically.

`\compressMMRests` *music* (music) ⇒ music
Convert empty bars to multi-measure rests in *music*.

`\contextPropertyCheck` *property-path* (symbol list or symbol) [*expected-value* (any type)] ⇒ music
Check that the context property identified by *property-path* is set to *expected-value* in that very context: being set in an enclosing context is insufficient. If *expected-value* is `\default`, check that the property is unset in that context.

If *property-path* does not name a context, the check occurs in the current context.

Print a warning if the requested context is not visible looking upward from the current context or if the state of the property in the requested context is unexpected.

Caveat: Like some other context-specific commands, a context given in *property-path* is ignored in certain cases, such as inside `\with`.

`\crossStaff` *notes* (music) ⇒ music
Create cross-staff stems for *notes*.

`\cueClef` *type* (string) ⇒ music
Set the current cue clef to *type*.

`\cueClefUnset` ⇒ music
Unset the current cue clef.

`\cueDuring` *what* (string) *dir* (direction) *main-music* (music) ⇒ music
Create a cue.

This function inserts the contents of quote *what* corresponding to *main-music*, in a `CueVoice` context called `cue` oriented by *dir*.

`\cueDuringWithClef` *what* (string) *dir* (direction) *clef* (string) *main-music* (music) ⇒ music
Create a cue with clef.

This function inserts the contents of quote *what* corresponding to *main-music*, in a `CueVoice` context called `cue` oriented by *dir* and using clef *clef*.

`\deadNote` *note* (music) ⇒ music
Print *note* with a cross-shaped note head.

`\defineBarLine` *bar* (string) *glyph-list* (list) ⇒ void
Define bar line settings for bar line *bar*.

The list *glyph-list* must have three entries, defining substitute glyphs for the end of a line, the beginning of a line, and a span bar, respectively. The substitute glyphs may be either strings or Booleans: `#t` calls for the same value as *bar* and `#f` calls for no glyph.

`\displayLilyMusic` [*port* (output port)] *music* (music) ⇒ music
Write LilyPond's input representation of *music*.

If *port* is omitted, the output defaults to the console (stdout).

`\displayMusic` [*port* (output port)] *music* (music) ⇒ music
Write the internal representation of *music*.

If *port* is omitted, the output defaults to the console (stdout).

`\displayScheme` [*port* (output port)] *expr* (any type) ⇒ any type
Write the internal Scheme representation of *expr*.

If *port* is omitted, the output defaults to the console (stdout).

\dropNote *num* (integer) *music* (music) ⇒ music
'Drop' the *num*-th note in each chord of *music*.

This function moves the affected notes down (usually by an octave) to be lower than the other notes of the chord. The position in a chord is counted downwards from the top.

The opposite function is \raiseNote (page 957).

\enablePerStaffTiming ⇒ void
Enable polymeter with unaligned measures.

This function moves the timing management from Score to Staff-like contexts. This is done by removing the Timing_translator from Score, and adding it to all contexts having the Staff alias.

Use this within an output definition.

\endSpanners *music* (music) ⇒ music
Terminate spanners.

This function prematurely ends all spanners in *music* by inserting an end-spanner event at the end of the argument, without the need of specific end-spanner commands.

\eventChords *music* (music) ⇒ music
Compatibility function: Handle isolated rhythmic events in *music*.

Use this to wrap EventChord around isolated rhythmic events occuring since version 2.15.28, after expanding repeat chords 'q'.

Not needed for new code.

\featherDurations *scale* (non-negative rational, fraction, or moment) *music* (music) ⇒ music
Adjust feathered beam durations in *music* by *scale*.

\finger *finger* (index or markup) ⇒ post-event
Apply *finger* as a fingering indication.

\fixed *pitch* (pitch) *music* (music) ⇒ music
Use the octave of *pitch* as the default octave for *music*.

\footnote [*mark* (markup)] *offset* (pair of numbers) *footnote* (markup) *item* (symbol list or music) ⇒ music
Make the markup *footnote* a footnote on *item*.

The footnote is marked with a markup *mark* moved by *offset* with respect to the marked music.

If *mark* is not given or specified as \default, it is replaced by an automatically generated sequence number. If *item* is a symbol list of form *Grob* or *Context.Grob*, then grobs of that type are marked at the current time step in the given context (default Bottom).

If *item* is music, the music gets a footnote attached to a grob immediately attached to the event, like \tweak does. For attaching a footnote to an *indirectly* caused grob, write \single\footnote, use *item* to specify the grob, and follow it with the music to annotate.

Like with \tweak, if you use a footnote on a following post-event, the \footnote command itself needs to be attached to the preceding note or rest as a post-event with '-'.

\grace *music* (music) ⇒ music
Insert *music* as grace notes.

`\grobdescriptions` *descriptions* (list) ⇒ any type
: Create a context modification from *descriptions*.

  The argument is a list in the format of `all-grob-descriptions`.

`\harmonicByFret` *fret* (number) *music* (music) ⇒ music
: Convert *music* into mixed harmonics.

  The resulting notes resemble harmonics played on a fretted instrument by touching the strings at *fret*.

`\harmonicByRatio` *ratio* (number) *music* (music) ⇒ music
: Convert *music* into mixed harmonics.

  The resulting notes resemble harmonics played on a fretted instrument by touching the strings at the point given through *ratio*.

`\harmonicNote` *note* (music) ⇒ music
: Print *note* with a diamond-shaped note head.

`\harmonicsOn` ⇒ music
: Set the default note head style to a diamond-shaped style.

`\hide` *item* (symbol list or music) ⇒ music
: Make *item* invisible while still retaining its dimensions.

  If *item* is a symbol list of form *`GrobName`* or *`Context.GrobName`*, the result is an override for the grob name specified by it. If *item* is a music expression, the result is the same music expression with an appropriate tweak applied to it.

  This function sets *item*'s `transparent` property to `#t`.

`\incipit` [*offset* (number)] *incipit-music* (music) ⇒ music
: Output *incipit-music* as an incipit.

  incipit-music is typeset within a `MensuralStaff` context; the result is positioned before the main staff (as part of an `InstrumentName` grob) to indicate the music's original notation. The optional argument *offset* specifies how much the incipit should be moved to the left (default value is 0).

  In the special case that *incipit-music* has the form `\new` *xxx* ..., where *xxx* is a context type not accepted by `MensuralStaff`, it is taken directly.

`\inherit-acceptability` *to* (symbol) *from* (symbol) ⇒ void
: Make two contexts 'compatible'.

  When used in an output definition, modify all context definitions such that context *to* is accepted as a child by all contexts that also accept *from*.

`\initialContextFrom` *music* (music) ⇒ music
: Enter the initial context of *music* and ignore the rest of it.

  This is useful for prepending music while preserving the influence of the original music on the context.

  Example:

```
{
  \initialContextFrom \originalMusic
  \prependedMusic
  \originalMusic
  \appendedMusic
}
```

`\inStaffSegno` ⇒ music
: Put the segno variant `varsegno` at this position into the staff.

  This is compatible with the repeat command.

`\instrumentSwitch` *name* (string) ⇒ music
: Switch instrument to *name*.

  name must have been predefined with function `\addInstrumentDefinition` (page 943).

  This function is deprecated.

`\inversion` *around* (pitch) *to* (pitch) *music* (music) ⇒ music
: Invert *music* about *around* and transpose from *around* to *to*.

`\invertChords` *num* (integer) *music* (music) ⇒ music
: Invert any chords in *music* into their *num*-th position.

  Chord inversions may be directed downwards using negative integers.

`\jump` *text* (markup) ⇒ music
: Use *text* to mark a point of departure, e.g., 'Gavotte I D.C.'.

`\keepWithTag` *tags* (symbol list or symbol) *music* (music) ⇒ music
: Keep tagged music.

  This function only includes elements of *music* that are tagged with one of the tags in *tags*. *tags* may be either a single symbol or a list of symbols.

  Each tag may be declared as a member of at most one tag group (defined with `\tagGroup`). If none of a *music* element's tags share a tag group with one of the specified *tags*, the element is retained.

`\key` [*tonic* (pitch)] [*pitch-alist* (list of number pairs)] ⇒ music
: Set key to *tonic* and scale *pitch-alist*.

  If both arguments are omitted (i.e., replaced by `\default`), just generate a `KeyChangeEvent`, which prints the current key signature again.

`\killCues` *music* (music) ⇒ music
: Remove cue notes from *music*.

`\label` *label* (symbol) ⇒ music
: Create *label* as a referrable label.

  The value stored in *label* is the page number, which can be extracted with the `\page-ref` markup command later on.

`\language` *language* (string) ⇒ void
: Set note names for language *language*.

  This function updates the Scheme variables `pitchnames` and `input-language`. The former is an alist that holds the pitches of all note names for the selected language; the latter holds the language name itself (as a symbol).

`\languageRestore` ⇒ void
: Restore the previously-saved `pitchnames` alist.

`\languageSaveAndChange` *language* (string) ⇒ void
: Save current `pitchnames` alist and change note names to *language*.

`\ligature` *music* (music) ⇒ music
: Make a ligature from Gregorian Chant *music*.

  This is equivalent to enclosing *music* with `\[` and `\]`.

`\magnifyMusic` *mag* (positive number) *music* (music) ⇒ music
: Magnify the size of *music* by factor *mag*.

  This doesn't change the staff size. Stems, beams, slurs, ties, and horizontal spacing are adjusted automatically.

\magnifyStaff *mag* (positive number) ⇒ music
: Change the staff size by factor *mag*.

  This adjusts notation size and horizontal spacing automatically.

\makeClusters *arg* (music) ⇒ music
: Display chords in *arg* as clusters.

\makeDefaultStringTuning *symbol* (symbol) *pitches* (list) ⇒ void
: Define string tuning *symbol* by a list of *pitches*.

  symbol also gets registered in `defaultStringTunings` for documentation purposes.

\mark [*label* (index or markup)] ⇒ music
: Create a rehearsal mark.

  If *label* is an integer, create the rehearsal mark for the given sequence number. If *label* is `\default`, create the next sequential rehearsal mark. If *label* is markup, use it for the mark.

\markupMap *path* (symbol list or symbol) *markupfun* (markup-function) *music* (music) ⇒ music
: Apply *markupfun* to property *path* in *music*.

  Argument *path* is either of the form *property* or *MusicExpression.property*. If *MusicExpression* is not given, *markupfun* gets applied to all properties called *property*, otherwise it is restricted to *MusicExpression* events. If *property* is not a markup, it is ignored.

  In the following example, both the tempo indication and the bowing instruction are printed in red. If you replace `text` with `TempoChangeEvent.text`, only the tempo indication changes the color.

```
\markupMap
  text
  \markup \with-color #red \etc
  { \tempo "Largo" g'2_"arco" c'' }
```

\measure *music* (music) ⇒ music
: Create a complete measure from *music*.

  This is equivalent to \measureRemainder *music*, but with a bar check at the beginning as well as the end.

\measureRemainder *music* (music) ⇒ music
: End the current measure after *music*.

  The effect is equivalent to this:

```
{
  \initialContextFrom %{ copy of music %}
  \setMeasureLengthFromHere %{ duration of music %}
  music |
  \setDefaultMeasureLength
}
```

  See \setMeasureLengthFromHere (page 958) for more information.

  Compare to \premeasure (page 955).

\modalInversion *around* (pitch) *to* (pitch) *scale* (music) *music* (music) ⇒ music
: Invert *music* about *around* using *scale* and transpose from *around* to *to*.

\modalTranspose *from* (pitch) *to* (pitch) *scale* (music) *music* (music) ⇒ music
: Transpose *music* from pitch *from* to pitch *to* using *scale*.

\musicLength *music* (music) ⇒ any type
: Return the duration of *music* as a `ly:moment`.

  Examples:

```
\musicLength 8 ⇒ #(ly:make-moment 1/8)
\musicLength {8. 8 8} ⇒ #(ly:make-moment 7/16)
```

\musicMap *proc* (procedure) *mus* (music) ⇒ music
: Apply *proc* to *mus* and all of the music it contains.

\newTocItem [*tocItemType* (symbol)] ⇒ any type
: Returns a variant of `\tocItem` that creates a table of contents entry for the table-of-content active at the time of its definition rather than at the time of its call. This can be useful if you want to maintain several tables of contents simultaneously.

  Called as `\newTocItem \default`, this is for the default `'tocItemMarkup` category; instead of `\default`, you can specify a different category.

  Normal use is right after the corresponding `\setupTableOfContents`, for example

```
\setupTableOfContents
topTocItem = \newTocItem \default
\markuplist \table-of-contents
...
```

  after which `\topTocItem` can be used for creating entries to that specific table of contents.

\noPageBreak ⇒ music
: Forbid a page break.

  May be used at top level (i.e., between scores or markups), or inside a score.

\noPageTurn ⇒ music
: Forbid a page turn.

  May be used at top level (i.e., between scores or markups), or inside a score.

\noStanza ⇒ music
: Forget the current stanza setting, thereby stopping any stanza reminders derived from it.

\octaveCheck *pitch* (pitch) ⇒ music
: Do an octave check.

  This prints a warning if the interval between the previous note and *pitch* is not within a fourth.

\offset *property* (symbol list or symbol) *offsets* (any type) *item* (key list or music) ⇒ music
: Offset the default value of *property* of *item* by *offsets*.

  If *item* is a string, the result is an override for the specified grob type. If *item* is a music expression, the result is the same music expression with an appropriate tweak applied to it.

\omit *item* (symbol list or music) ⇒ music
: Omit *item* without taking up space.

  If *item* is a symbol list of form *`GrobName`* or *`Context.GrobName`*, the result is an override for the grob name specified by it. If *item* is a music expression, the result is the same music expression with an appropriate tweak applied to it.

  This function sets *item*'s `stencil` property to `#f`.

\once *music* (music) ⇒ music
: Set property `once` to `#t` on all layout instruction events in *music*.

\ottava *octave* (integer) ⇒ music
: Set the octavation to *octave*.

  A positive value *n* indicates *n* octaves higher; a negative value *n* octaves lower, and value 0 means no octavation.

\overrideProperty *grob-property-path* (list of indices or symbols) *value* (any type) ⇒ music
: Set the grob property specified by *grob-property-path* to *value*.

  grob-property-path is a symbol list of the form *Context.GrobName.property* or *GrobName.property*, possibly with subproperties given as well.

  As opposed to \override, which overrides the context-dependent defaults with which a grob is created, this command uses Apply_output_engraver at the grob acknowledge stage. This may be necessary for overriding values set after the initial grob creation.

\overrideTimeSignatureSettings *time-signature* (boolean-or-fraction) *beat-base* (positive exact rational, fraction (as pair), moment, or +inf.0) *structure* (a number list or a list of them) *beam-exceptions* (list) ⇒ music
: Override time signature settings.

  This function sets the timeSignatureSettings entry for *time-signature* to use *beat-base*, *structure*, and *beam-exceptions*.

  beat-base and *beam-exceptions* are used as-is for beatBase and beamExceptions.

  structure is used to derive values for beatStructure and submeasureStructure, as described for \time (page 961).

\pageBreak ⇒ music
: Force a page break.

  May be used at top-level (i.e., between scores or markups), or inside a score.

\pageTurn ⇒ music
: Force a page turn.

  May be used at top-level (i.e., between scores or markups), or inside a score.

\palmMute *note* (music) ⇒ music
: Print *note* with a triangle-shaped note head.

\palmMuteOn ⇒ music
: Set the default note head style to a triangle-shaped style.

\parallelMusic *voice-ids* (list) *music* (music) ⇒ void
: Define parallel music sequences.

  Within *music*, parallel music sequences are separated by '|' characters. The sequences are assigned to the LilyPond music identifiers provided in *voice-ids*.

  For example, this code

```
\parallelMusic A,B,C {
  c c | d d | e e |
  d d | e e | f f |
}
```

  is equivalent to

```
A = { c c | d d }
B = { d d | e e }
C = { e e | f f }
```

  The last bar checks in a sequence are not copied to the result in order to facilitate ending the last entry at non-bar boundaries.

\parenthesize *arg* (symbol list or music) ⇒ music
: Tag *arg* to be parenthesized.

  arg may be either a music event or a grob path.

\partCombine [*chord-range* (pair of numbers)] *part1* (music) *part2* (music) ⇒ music
: Combine two parts into a single staff.

  This takes the music in *part1* and *part2* and returns a music expression containing simultaneous `Voice` contexts (called `one` for the upper and `two` for the lower voice). Where appropriate, *part1* and *part2* are combined into a single voice (called `shared` or `solo`, depending on context).

  Optional argument *chord-range* is a pair `(start . stop)` that defines the range in which the two voices are printed as chords (or unison); the default value is `(0 . 8)`, which means that intervals up to and including a ninth are unified.

\partCombineDown [*chord-range* (pair of numbers)] *part1* (music) *part2* (music) ⇒ music
: Combine two parts into a single staff with all stems downwards.

  For details, see \partCombine (page 954).

\partCombineForce [*type* (symbol)] ⇒ music
: Override the part-combiner mode with *type*.

  The following table gives the possible values for *type*, together with the corresponding shorthand functions.

`apart`	`\partCombineApart`
`chords`	`\partCombineChords`
`unisono`	`\partCombineUnisono`
`solo1`	`\partCombineSoloI`
`solo2`	`\partCombineSoloII`
`\default`	`\partCombineAutomatic`

\partCombineUp [*chord-range* (pair of numbers)] *part1* (music) *part2* (music) ⇒ music
: Combine two parts into a single staff with all stems upwards.

  For details, see \partCombine (page 954).

\partial *dur* (duration-or-music) ⇒ music
: End the current measure after *dur*.

  dur can be specified as music; this is intended for specifying compound durations like `{ 2 4. }` and does not typeset the music; use \premeasure (page 955) for that purpose.

  Adjust `Timing.measurePosition` to end the current measure at *dur* past the point of use. As a special case, when used at the start, create an anacrusis before the first measure.

  Notational elements that communicate metric structure (e.g., beam breaks) will be engraved appropriately for the final *dur* part of a measure. Compare to \setMeasureLengthFromHere (page 958).

  The calculation requires `Timing.measureLength` to be finite, so `\partial` cannot be used in a senza-misura passage.

\phrasingSlurDashPattern *dash-fraction* (number) *dash-period* (number) ⇒ music
: Set up a custom dash pattern style for phrasing slurs.

  dash-fraction gives the size of one dash relative to *dash-period*; *dash-period* is the length of one dash plus one space. LilyPond adjusts *dash-period* to produce symmetrical output.

More complex patterns can be achieved by directly manipulating the PhrasingSlur .dash-definition property.

\pitchedTrill *main-note* (music) *secondary-note* (music) ⇒ music
Print a pitched trill.

main-note is the main note of the trill; *secondary-note* gets printed as a stemless note head in parentheses.

\pointAndClickOff ⇒ void
Suppress links to LilyPond source code in music output.

\pointAndClickOn ⇒ void
Generate links to LilyPond source code in music output.

This enables the creation of code in a PDF or SVG output file to reference the originating LilyPond source code (i.e., file name, line number, and column). This is helpful when developing a score; however, the output file becomes much larger.

\pointAndClickTypes *types* (symbol list or symbol) ⇒ void
Generate point-and-click info for music of type *types* only.

types is a single music expression (such as #'note-event) or a list of music expressions.

\polymetric [*time-signature-command* (music)] ⇒ music
Apply *time-signature-command* to the local context rather than to Timing.

Example:

```
\context Staff \polymetric \time 6/8
```

The local measure must align with the reference measure defined in Timing. So that explicit changes to Timing.measureLength (e.g., for ad-hoc irregular measures) remain visible, this command does not set the measureLength property in the local context.

A nominally incompatible local measure can be fitted to the controlling Timing measure by applying \scaleDurations (page 957) to this command. That adjusts the value of the internal property meterScalingFactor in the local context.

To unset the local properties and resume using the values from the Timing context, use \polymetric \default.

\popContextProperty *path* (list of indices or symbols) ⇒ music
Pop value of context property *path* from stack and set it.

This is the opposite to function \pushContextProperty (page 956).

\preBend *mus* (music) ⇒ post-event
Set BendSpanner.style to 'pre-bend for *mus*.

\preBendHold *mus* (music) ⇒ post-event
Set BendSpanner.style to 'pre-bend-hold for *mus*.

\premeasure *music* (music) ⇒ music
Create a lead-in partial measure from *music*.

The effect is equivalent to this:

```
{
  \initialContextFrom %{ copy of music %}
  \partial %{ duration of music %}
  music |
}
```

Compare to \measureRemainder (page 951).

\propertyOverride *grob-property-path* (list of indices or symbols) *value* (any type) ⇒ music
Set the grob property specified by *grob-property-path* to *value*.

grob-property-path is a symbol list of the form *Context.GrobName.property* or *GrobName.property*, possibly with subproperties given as well. This music function is mostly intended for use from Scheme as a substitute for the built-in \override command.

\propertyRevert *grob-property-path* (list of indices or symbols) ⇒ music
Revert the grob property specified by *grob-property-path* to its previous value.

grob-property-path is a symbol list of the form *Context.GrobName.property* or *GrobName.property*, possibly with subproperties given as well. This music function is mostly intended for use from Scheme as a substitute for the built-in \revert command.

\propertySet *property-path* (symbol list or symbol) *value* (any type) ⇒ music
Set the context property specified by *property-path* to *value*.

This music function is mostly intended for use from Scheme as a substitute for the built-in \set command.

\propertyTweak *prop* (key list or symbol) *value* (any type) *item* (key list or music) ⇒ music
Add a tweak to *item*, usually music.

This function sets the value of property *prop* to *value*; it generally behaves like \tweak but will turn into an \override when *item* is a symbol list. In that case, *item* specifies the grob path to override. This is mainly useful when using \propertyTweak as as a component for building other functions like \omit. It is not the default behavior for \tweak since many input strings in \lyricmode can serve equally as music or as symbols, which causes surprising behavior when tweaking lyrics using the less specific semantics of \propertyTweak.

prop can contain additional elements in which case a nested property (inside of an alist) is tweaked.

\propertyUnset *property-path* (symbol list or symbol) ⇒ music
Unset the context property specified by *property-path*.

This music function is mostly intended for use from Scheme as a substitute for the built-in \unset command.

\pushContextProperty *path* (list of indices or symbols) ⇒ music
Push the current value of context property *path* to a stack.

The property can later be restored to the saved value with function \popContextProperty (page 955).

\pushToTag *tag* (symbol) *more* (music) *music* (music) ⇒ music
Add *more* to the front of *music* tagged with *tag*.

A post-event can be added to the articulations of rhythmic events or chords; other expressions may be added to chords, sequential or simultaneous music.

\pushToTagMarkup *tag* (symbol) *more* (markup) *music* (music) ⇒ music
Prepend *more* to every markup in *music* tagged with *tag*.

\quoteDuring *what* (string) *main-music* (music) ⇒ music
Indicate a section of music to be quoted.

what indicates the name of the quoted voice specified with \addQuote (page 943). *main-music* is used to indicate the length of music to be quoted; it usually contains spacers or multi-measure rests.

`\raiseNote` *num* (integer) *music* (music) ⇒ music
'Raise' the *num*-th note in each chord of *music*.

This function moves the affected notes up (usually by an octave) to be higher than the other notes of the chord. The position in a chord is counted upwards from the bottom.

The opposite function is `\dropNote` (page 948).

`\reduceChords` *music* (music) ⇒ music
Reduce chords contained in *music* to single notes.

This is intended mainly for reusing music in a `RhythmicStaff` context. It does not reduce simultaneous music.

`\relative` [*pitch* (pitch)] *music* (music) ⇒ music
Make *music* relative to *pitch*.

If *pitch* is omitted, the first note in *music* is given in absolute pitch.

`\removeFromTagGroup` *tag-group* (symbol list) *tags* (symbol list) ⇒ void
Remove *tags* from the tag group identified by *tag-group*.

`\removeWithTag` *tags* (symbol list or symbol) *music* (music) ⇒ music
Remove elements of *music* that are tagged with one of the tags in *tags*.

tags may be either a single symbol or a list of symbols.

`\replaceWithTag` *tag* (symbol) *replacement* (music) *music* (music) ⇒ music
Replace tagged elements in *music*.

Everything tagged with *tag* (including the tagging itself) gets replaced with *replacement*.

`\replaceWithTagMarkup` *tag* (symbol) *replacement* (markup) *music* (music) ⇒ music
Replace tagged markup in *music*.

All markup tagged with *tag* (including the tagging itself) gets replaced with *replacement*.

`\resetRelativeOctave` *pitch* (pitch) ⇒ music
Set the octave inside a `\relative` section to *pitch*.

`\resetTagGroup` *tag-group* (symbol list) ⇒ void
Reset the tag group equal to *tag-group*.

`\resetTagGroups` ⇒ void
Reset all tag groups previously created with `\tagGroup`.

`\responsum` *music* (music) ⇒ music
Prepend character U+211F (RESPONSE) to the lyrics represented by *music*.

`\retrograde` *music* (music) ⇒ music
Return *music* in reverse order.

`\revertTimeSignatureSettings` *time-signature* (pair) ⇒ music
Revert `timeSignatureSettings` for time signatures equal to *time-signature*.

`\rightHandFinger` *finger* (index or markup) ⇒ post-event
Apply *finger* as a right-hand fingering indication.

`\rigid` *music* (music) ⇒ music
Render *music* without its elements moving before grace elements in other music

`\scaleDurations` *fraction* (non-negative rational, fraction, or moment) *music* (music) ⇒ music
Multiply the duration of events in *music* by *fraction*.

`\sectionLabel` *text* (markup) ⇒ music
: Mark the beginning of a named passage with *text*, e.g., “Coda”.

  This is well suited for use at a section division created with `\section`, but it does not imply `\section` and may be used alone.

`\segnoMark` [*num* (non-negative, exact integer)] ⇒ music
: Create a segno mark (or bar line).

  num may be 1 for the first segno, 2 for the second, etc., or it may be `\default` to use the next number in sequence automatically.

  If the `segnoStyle` context property is `'bar-line`, a segno bar line is created instead of a segno mark.

`\senzaMisura` ⇒ music
: Cancel the time signature; begin free time.

  This sets `Timing.timeSignature` to `#f`.

  This sets `Timing.measureLength` to `+inf.0`, allowing `Timing.measurePosition` to advance indefinitely without starting a new measure.

  Beyond this point, ending the current measure requires setting `Timing.measureLength` back to a finite value *before* (not *at*) the transition to the next measure. See `\measure` (page 951) and related functions for assistance.

  Compare to `\cadenzaOn` (page 946).

`\setDefaultMeasureLength` ⇒ music
: Set `Timing.measureLength` from `Timing.timeSignature`.

`\setMeasureLengthFromHere` *dur* (duration-or-music) ⇒ music
: End the current measure after *dur*.

  Adjust `Timing.measureLength` to end at *dur* past the point of use. The resulting measure length remains in effect until it is changed by another command.

  dur can be specified as music; this is intended for specifying compound durations like `{ 2 4. }` and does not typeset the music; use `\measureRemainder` (page 951) for that purpose.

  Notational elements that communicate metric structure (e.g., beam breaks) will be engraved appropriately to continue the measure. Compare to `\partial` (page 954).

  See also `\measure` (page 951) and `\measureRemainder` (page 951).

`\setupTableOfContents` ⇒ void
: Set up a table of contents in the currently active paper block.

  While a table of contents is established on the first call to `\tocItem` that cannot find one, using this function can establish a fresh table of contents if one happens to already be accessible, or it can be used when the first `\tocItem` would establish one at the wrong layer.

  Placing `\setupTableOfContents` right before

  ```
  \markuplist \table-of-contents
  ```

  will be an obvious choice when the table of contents precedes all referenced items.

`\shape` *offsets* (list) *item* (key list or music) ⇒ music
: Offset control points of *item* by *offsets*.

  offsets is a list of number pairs `(x . y)` or a list of such lists. Each pair represents an offset to a control point. The ‘y’ value of each pair is scaled by staff space.

  If *item* is a string, the result is `\once\override` for the specified grob type. If *item* is a music expression, the result is the same music expression with an appropriate tweak applied.

`\shiftDurations` *dur* (integer) *dots* (integer) *arg* (music) ⇒ music
Change duration of *arg*.

This function walks over all durations and dot counts in *arg*, adding *dur* to the durations and *dots* to the dot counts.

`\single` *overrides* (music) *music* (music) ⇒ music
Convert *overrides* to tweaks and apply them to *music*.

This does not convert `\revert`, `\set` or `\unset`.

`\skip` *arg* (duration-or-music) ⇒ music
Skip over *arg*, which may be music or a duration.

`\slashedGrace` *music* (music) ⇒ music
Create slashed graces from *music*.

This produces slashes through stems, but no slur.

`\slurDashPattern` *dash-fraction* (number) *dash-period* (number) ⇒ music
Set up a custom dash pattern style for slurs.

dash-fraction gives the size of one dash relative to *dash-period*; *dash-period* is the length of one dash plus one space. LilyPond adjusts *dash-period* to produce symmetrical output.

More complex patterns can be achieved by directly manipulating the `Slur .dash-definition` property.

`\staffHighlight` *color* (color) ⇒ music
Start a highlight with color *color*.

`\stanza` *text* (markup) ⇒ music
Set and display the stanza number *text* in lyrics.

`\startGradualTempoChange` [*text* (markup)] ⇒ music
Start an accelerando or ritardando.

Begin a gradual departure from the tempo most recently set with `\tempo`. The gradual change extends to the next `\stopGradualTempoChange` or `\tempo` command. When `\stopGradualTempoChange` is used, it defines the target tempo; otherwise, `\tempo` defines it.

The *text* argument is a placeholder. Only `\default` is accepted. It is intended eventually to serve the same purpose as the text argument to `\tempo`.

`\stopGradualTempoChange` [*text* (markup)] *tempo-unit* (duration) *metronome-count* (number) ⇒ music
End an accelerando or ritardando.

End a gradual tempo change at the tempo defined by *tempo-unit* and *metronome-count*. Unless there is a simultaneous `\tempo` command setting a new tempo, resume the tempo most recently set with `\tempo`.

The *text* argument is a placeholder. Only `\default` is accepted. It is intended eventually to serve the same purpose as the text argument to `\tempo`.

`\storePredefinedDiagram` *fretboard-table* (hash table) *chord* (music) *tuning* (pair) *diagram-definition* (string or pair) ⇒ void
Add a predefined fret diagram to *fretboard-table*.

It is defined by *diagram-definition* for the chord pitches *chord* and the string tuning *tuning*.

`\stringTuning` *chord* (music) ⇒ any type
: Convert *chord* to a string tuning.

  chord must be in absolute pitches and should have the highest string number (generally the lowest pitch) first.

`\styledNoteHeads` *style* (symbol) *heads* (symbol list or symbol) *music* (music) ⇒ music
: Set *heads* in *music* to *style*.

`\tabChordRepeats` [*event-types* (list)] *music* (music) ⇒ music
: Extend 'q' to also repeat string and fingering information.

  This function walks through *music* putting the notes, fingerings and string numbers of the previous chord into repeat chords, as well as an optional list of *event-types* such as `#'(articulation-event)`.

`\tabChordRepetition` ⇒ void
: Include the string and fingering information in a chord repetition.

  This function is deprecated; use `\tabChordRepeats` instead.

`\tag` *tags* (symbol list or symbol) *music* (music) ⇒ music
: Tag *music* with *tags*.

  This function adds the single symbol or symbol list *tags* to the `tags` property of *music* and returns the result.

`\tagGroup` *tags* (symbol list) ⇒ void
: Define a tag group comprising the symbols in the symbol list *tags*.

  Tag groups must not overlap.

`\tagGroupRef` *tags* (symbol list) ⇒ any type
: Define a tag group comprising the symbols in the symbol list *tags*.

  It returns the created tag group for further reference.

`\temporary` *music* (music) ⇒ music
: Make `\override` reversible with `\revert`.

  This function makes any `\override` in *music* replace an existing grob property value only temporarily, restoring the old value when a corresponding `\revert` is executed. This is achieved by clearing the `pop-first` property normally set on `\overrides`.

  An `\override`/`\revert` sequence created by using `\temporary` and `\undo` on the same music containing overrides will cancel out perfectly or cause a warning.

  Non-property-related music is ignored, warnings are generated for any property-changing music that isn't an `\override`.

`\textEndMark` *text* (markup) ⇒ music
: Create a right-aligned text mark using *text*.

`\textMark` *text* (markup) ⇒ music
: Create a (left-aligned) text mark using *text*.

`\tieDashPattern` *dash-fraction* (number) *dash-period* (number) ⇒ music
: Set up a custom dash pattern style for ties.

  dash-fraction gives the size of one dash relative to *dash-period*; *dash-period* is the length of one dash plus one space. LilyPond adjusts *dash-period* to produce symmetrical output.

  More complex patterns can be achieved by directly manipulating the `Tie .dash-definition` property.

`\time` [*structure* (a number list or a list of them)] *time-sig* (time signature) ⇒ music
: Set the time signature to *time-sig*.

  When *structure* is a plain list of numbers, it is used as-is for `beatStructure`, and `submeasureStructure` is left at the default.

  When *structure* is a list of lists, `beatStructure` is derived by discarding the grouping, and `submeasureStructure` is derived by summing each group. For example, a *structure* of `'((1 2) (4 8))` yields a `beatStructure` of `'(1 2 4 8)` and a `submeasureStructure` of `'(3 12)`.

  time-sig may be a fraction, e.g., `3/4`.

  time-sig may also describe a complex time signature as a Scheme expression. Fractions are represented as pairs, (*`numerator . denominator`*), where the denominator is always a number. The numerator is one number or a list of two or more numbers. A list represents concatenation.

  For example, a time signature of (3+1)/8 + 2/4 can be created with `\time #'(((3 1) . 8) (2 . 4))`

`\timeAbbrev` *time-sig* (pair) ⇒ music
: Set the time signature to *time-sig*.

  This is like `\time` *time-sig*, except that it allows abbreviating fractions as lists. For example, a time signature of (3+1)/8 + 2/4 can be created with `\timeAbbrev #'((3 1 8) (2 4))`, and a time signature of (3+2)/8 with either `\timeAbbrev #'((3 2 8))` or the shorter version `\timeAbbrev 3,2,8`.

  This is for backward compatibility. Using `\time` (page 961) instead is recommended.

`\times` *fraction* (fraction, as pair) *music* (music) ⇒ music
: Scale *music* in time by *fraction*.

`\tocItem` [*label* (symbol list or symbol)] *text* (markup) ⇒ music
: Add *text* as an entry to the table of contents.

  This uses the `tocItemMarkup` paper variable markup for formatting and assigns it to *label* if one is provided. If a hierarchy of labels is given, make the current item a child of the corresponding objects.

`\transpose` *from* (pitch) *to* (pitch) *music* (music) ⇒ music
: Transpose *music* from pitch *from* to pitch *to*.

`\transposedCueDuring` *what* (string) *dir* (direction) *pitch* (pitch) *main-music* (music) ⇒ music
: Create a transposed cue.

  This function inserts notes from the part *what* into a `CueVoice` context called `cue`, using the transposition defined by *pitch*. 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* (pitch) ⇒ music
: Set instrument transposition to *pitch*.

`\tuplet` *ratio* (fraction, as pair) [*tuplet-span* (duration)] *music* (music) ⇒ music
: Scale the given *music* to tuplets.

  ratio is a fraction that specifies how many notes are played in place of the nominal value: it will be 3/2 for triplets, namely three notes being played in place of two.

  If the optional duration *tuplet-span* is specified, it is used instead of `tupletSpannerDuration` for grouping the tuplets. For example,

  ```
  \tuplet 3/2 4 { c8 c c c c c }
  ```

results in two groups of three tuplets, each group lasting for a quarter note.

`\tupletSpan` [*tuplet-span* (duration)] ⇒ music
: Set `tupletSpannerDuration` to the duration *tuplet-span*.

  This context property is the length into which `\tuplet` without an explicit tuplet span argument of its own will group its tuplets. To revert to the default of not subdividing the contents of a `\tuplet` command without an explicit tuplet span argument, use

  ```
  \tupletSpan \default
  ```

`\tweak` *prop* (key list or symbol) *value* (any type) *music* (music) ⇒ music
: Add a tweak to *music*.

  Layout objects created by *music* get their property *prop* set to *value*. If *prop* has the form *`Grob.property`*, like with

  ```
  \tweak Accidental.color #red cis'
  ```

  an indirectly created grob (`Accidental` is caused by `NoteHead`) can be tweaked; otherwise only directly created grobs are affected.

  prop can contain additional elements in which case a nested property (inside of an alist) is tweaked.

  If *music* is an `event-chord`, every contained `rhythmic-event` is tweaked instead.

`\undo` *music* (music) ⇒ music
: Convert `\override` and `\set` in *music* to `\revert` and `\unset`, respectively.

  Any reverts and unsets already in *music* cause a warning. Non-property-related music is ignored.

`\unfolded` *music* (music) ⇒ music
: Mask *music* until the innermost enclosing repeat is unfolded.

`\unfoldRepeats` [*types* (symbol list or symbol)] *music* (music) ⇒ music
: Unfold `\repeat`.

  This forces `\repeat volta`, `\repeat tremolo` or `\repeat percent` commands in *music* to be interpreted as `\repeat unfold`, if specified in the optional symbol-list *types*. The default for *types* is an empty list, which forces any of those commands in *music* to be interpreted as `\repeat unfold`. Possible entries are `volta`, `tremolo` or `percent`. Multiple entries are possible.

`\verboseBarNumbers` ⇒ music
: Print bar numbers at every bar line and line break.

  This function is a concise way to add detail to documentation examples. The output is subject to change without notice.

  See Section “Bar_number_engraver” in *Internals Reference*.

`\versus` *music* (music) ⇒ music
: Prepend character U+2123 (VERSICLE) to the lyrics represented by *music*.

`\voices` *ids* (list of indices or symbols) *music* (music) ⇒ music
: Specify voice order in simultaneous music.

  This takes the key list *ids* of numbers (indicating the use of ‘`\voiceOne`’...) or symbols (indicating voice names, typically converted from strings by argument list processing) and assign the following \\-separated music in *music* to contexts according to that list. Named rather than numbered contexts can be used for continuing one voice (for the sake of spanners and lyrics), usually requiring a `\voiceOne`-style override at the beginning of the passage and a `\oneVoice` override at its end.

The default

```
<< ... \\ ... \\ ... >>
```

construct would correspond to

```
\voices 1,2,3 << ... \\ ... \\ ... >>
```

\void *arg* (any type) ⇒ void
: Accept a Scheme argument *arg* and return a void expression.

  Use this if you want to have a Scheme expression evaluated because of its side effects but its return value being ignored.

\volta *volta-numbers* (number list) *music* (music) ⇒ music
: Mark *music* as being limited to the volte given in *volta-numbers*.

  This gets used when the innermost enclosing repeat is unfolded. Volta number begins at 1 and increases by 1 with each repetition.

\vshape *offsets* (list) *item* (key list or music) ⇒ music
: Like \shape, but additionally show control points for ease of tweaking.

\withMusicProperty *sym* (symbol) *val* (any type) *music* (music) ⇒ music
: Set music property *sym* to *val* in *music*.

\withRelativeDir *file-name* (string) ⇒ any type
: Prepend directory of current input file to string *file-name*.

  Use this for markup commands that include files, and where such files should be found relative to the input file. Example:

```
\markup { \image #X #3 \withRelativeDir "test.png" }
```
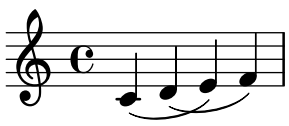
\xNote *note* (music) ⇒ music
: Print *note* with a cross-shaped note head.

\= *id* (index or symbol) *event* (post-event) ⇒ post-event
: Assign an ID to a spanner or an item.

  This sets the `spanner-id` or `id` property of *event* to the given *id*, which is a non-negative integer or a symbol.
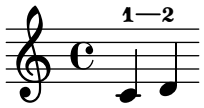
  For spanners this can be used to tell LilyPond how to connect overlapping or parallel slurs or phrasing slurs within a single `Voice` context.

```
\fixed c' { c\=1( d\=2( e\=1) f\=2) }
```



  For items this can be used, for example, to tell LilyPond how to connect a `FingerGlideSpanner` with non-matching fingers.

```
\fixed c' { c\glide \= #'foo -1 d\= #'foo -2 }
```



\% *count* (number) *music* (music) ⇒ music
: This is the same as \repeat `percent` *count music*.

  See \repeat.

* *count* (number) *music* (music) ⇒ music
: This is the same as \repeat unfold *count music*.

  See \repeat.

\~ *music* (music) ⇒ music
: Insert pes or flexa between previous and next note.

B.22 Context modification identifiers

The following commands are defined for use as context modifications within a \layout or \with block.

\EnableGregorianDivisiones
: Configure division commands such as \section to create Divisio grobs rather than BarLine grobs. This does not affect measure bar lines or the properties of the grobs themselves.

  - Sets translator property caesuraTypeTransform to caesura-to-divisio.
  - Sets translator property doubleRepeatBarType to '().
  - Sets translator property endRepeatBarType to '().
  - Sets translator property fineBarType to "".
  - Sets translator property sectionBarType to "".
  - Sets translator property startRepeatBarType to '().
  - Sets translator property underlyingRepeatBarType to "".
  - Sets translator property doubleRepeatSegnoBarType to "S-||".
  - Sets translator property endRepeatSegnoBarType to "S-||".
  - Sets translator property fineSegnoBarType to "S-||".
  - Sets translator property fineStartRepeatSegnoBarType to "S-||".
  - Sets translator property segnoBarType to "S-||".
  - Sets translator property startRepeatSegnoBarType to "S-||".

\RemoveAllEmptyStaves
: Remove staves which are considered to be empty according to the list of interfaces set by keepAliveInterfaces, including those in the first system.

  - Sets grob property remove-empty in Section "VerticalAxisGroup" in *Internals Reference* to #t.
  - Sets grob property remove-first in Section "VerticalAxisGroup" in *Internals Reference* to #t.

\RemoveEmptyStaves
: Remove staves which are considered to be empty according to the list of interfaces set by keepAliveInterfaces.

  - Sets grob property remove-empty in Section "VerticalAxisGroup" in *Internals Reference* to #t.

B.23 Paper variables

The following paper variables are predefined by LilyPond. Default values that are scaled to paper size are given for the default paper size, A4. The mentioned *xxx*-default variables are paper variables, too.

Some values like mm or pt are constants and thus essentially read-only; changing them would confuse LilyPond thoroughly.

Most paper variables can be either set in \paper or in \layout, see Section 27.1 [The \layout block], page 683.

annotate-spacing (boolean)
: If this value is set to #t, dimensions of vertical layout variables that may be altered for page formatting are graphically displayed. Default: #f.

auto-first-page-number (boolean)
: The page-breaking algorithm is affected by the first page number being odd or even. If set to #t, the algorithm decides whether to start with an odd or even number. This results in the first page number remaining as is or being increased by one. Default: #f.

binding-offset (number)
: This amount gets added to inner-margin so that nothing is hidden by the binding. Has only an effect if two-sided is set to #t. Default: binding-offset-default (5 mm, scaled to paper size).

blank-after-score-page-penalty (number)
: The penalty for having a blank page after the end of one score and before the next. By default, this is smaller than blank-page-penalty so that LilyPond prefers blank pages after scores to blank pages within a score. Default: 2.

blank-last-page-penalty (number)
: The penalty for ending the score on an odd-numbered page. Default: 0.

blank-page-penalty (number)
: The penalty for having a blank page in the middle of a score. Note that this is not used by ly:optimal-breaking, which never considers blank pages in the middle of a score. Default: 5.

blot-diameter (non-negative number)
: This value globally defines the smallest 'round' feature LilyPond uses while constructing almost all non-glyph elements like beams or stems. Essentially, it sets up how round the corners and line ends are. It only makes sense to change the value if you use a different music glyph font with crisper corners, say. Default: 0.4 pt.

book-title (procedure)
: Internal. This is what LilyPond actually uses for handling bookTitleMarkup.

book-title-properties (association list (list of pairs))
: Internal.

bookpart-level-page-numbering (boolean)
: If set to #t, restart page numbering for each \bookpart block. Otherwise all pages in the document are enumerated continuously. Default: #f.

bookTitleMarkup (markup)
: The titling markup within a \book, using standard fields from the \header block. The default value is defined in file titling-init.ly.

bottom-margin (non-negative number)
: The margin between the bottom of the printable area and the bottom of the page. Default: bottom-margin-default (10 mm, scaled to paper size).

bp (positive number)
: The big point unit, also called *desktop publishing point* (DTP point). It is 1/72 of an inch, approx. 0.353 mm (0.0138 in).

`check-consistency` (boolean)
: If set to `#t`, print a warning if the left margin, line width, and right margin do not exactly add up to `paper-width`, and replace each of these (except `paper-width`) with its default value (scaled to the paper size if necessary). If set to `#f`, ignore any inconsistencies and allow systems to run off the edge of the page. Default: `#t`.

`clip-regions` (association list (list of pairs))
: A list of rhythm location pairs to output fragments of a score.

```
clip-regions
= #(list (cons (make-rhythmic-location 2 0 1)
               (make-rhythmic-location 4 0 1))
         (cons (make-rhythmic-location 5 1 2)
               (make-rhythmic-location 7 3 4)))
```

The above example defines two regions: the first is from the beginning of bar 2 to the beginning of bar 4, the second starts after a half note duration in bar 5 and ends after the third quarter in bar 7.

Has only an effect if LilyPond's command-line option `-dclip-systems` is active. Default: unset.

`cm` (positive number)
: The centimeter unit, approx. 0.39 in.

`debug-beam-scoring` (boolean)
: If set to `#t`, print demerits together with their cause, followed by the number of configurations that have been scored before concluding. Default: `#f`.

Example:

`L 18.95 C 655.12 c19/625` → demerits for stem lengths ('`L`') and collisions ('`C`'), scored 19 out of 625 initially considered configurations.

Possible demerit causes: collision ('`C`'), inappropriate stem length ('`L`'), beam direction different from damping direction ('`Sd`'), difference between beam slope and musical slope ('`Sm`'), deviation from ideal slope ('`Si`'), horizontal inter-quants ('`H`'), forbidden quants ('`Fl`'/'`Fs`').

Demerits are configurable, see Section "beam-interface" in *Internals Reference* for a list of tunable parameters.

`debug-slur-scoring` (boolean)
: If set to `#t`, print demerits together with their cause, followed by the sum of all demerits and the index of the slur configuration finally chosen. Default: `#f`.

Example:

`slope=2.00, R edge=10.51, variance=0.03 TOTAL=12.54 idx=4` → demerits for slope, distance of the right edge to the attachment point, variance of distance between note heads and slur. Total demerits: 12.54, index of the chosen configuration: 4.

Possible demerit causes: distance of the left/right slur edge to the attachment points ('`L edge`'/'`R edge`'), inappropriate slope ('`slope`'), distance variations between note heads and slur ('`variance`'), distances for heads that are between the slur and an imaginary line between the attachment points ('`encompass`'), too small distance between slur and tie extrema ('`extra`').

Demerits are configurable, see Section "slur-interface" in *Internals Reference* for a list of tunable parameters.

`debug-tie-scoring` (boolean)
: If set to `#t`, print the basic configuration of ties, followed by demerits and their corresponding causes and the total sum of demerits. Default: unset.

  Example:

  ```
  0 (0.23) u: vdist=1.08 lhdist=1.79 tie/stem dir=8.00 TOTAL=10.87 →
  ```
  offset from the center of the staff according tie specification: 0 staff-spaces, vertical distance of the tie's center in y-direction to the bottom (or top) of the tie: 0.23, direction: up. Demerits for vertical and horizontal distance to note head, same direction of stem and tie. Total demerits: 10.87.

  Possible demerit causes: wrong tie direction ('`wrong dir`'), vertical distance to note heads ('`vdist`'), horizontal distance to left or right note head ('`lhdist`'/'`rhdist`'), same direction of stem and tie ('`tie/stem dir`'), position and direction of tie not matching, e.g., tie is in the upper half of the staff but has direction `DOWN` ('`tie/pos dir`'), tie is too short ('`minlength`'), tip of tie collides with staff line ('`tipline`'), collision with dot ('`dot collision`'), center of tie is too close to a staff line ('`line center`'), y-position (edge or center) of currently considered tie is less than the y-position of the previous tie ('`monotone edge`'/ '`monotone cent`'), edge or center of tie is too close to the one considered previously ('`tietie center`'/'`tietie edge`'), unsymmetrical horizontal positioning with respect to the note heads ('`length symm`'), unsymmetrical vertical positioning with respect to the note heads ('`pos symmetry`').

  Demerits are configurable, see Section "tie-interface" in *Internals Reference* for a list of tunable parameters.

`dimension-variables` (list)
: Internal.

`evenFooterMarkup` (markup)
: The footer markup used for even-numbered pages. If not set, `oddFooterMarkup` is used instead. The default value is defined in file `titling-init.ly`.

`evenHeaderMarkup` (markup)
: The header markup used for even-numbered pages. If not set, `oddHeaderMarkup` is used instead. The default value is defined in file `titling-init.ly`.

`first-page-number` (integer)
: The value of the page number on the first page. Default: 1.

`footnote-footer-padding` (number)
: The padding between the footer and the bottom-most footnote. Default: 0.5 mm.

`footnote-number-raise` (number)
: This controls how high the annotation numbers of both footnotes and in-notes are raised relative to the footnote or in-note text. Default: 0.5 mm.

`footnote-numbering-function` (procedure)
: This variable holds the name of the function that formats both footnote and in-note numbers (without positioning it). LilyPond provides two predefined functions (in file `output-lib.scm`): `numbered-footnotes` ('1', '2', etc.) and `symbol-footnotes` ('*', '†', etc.). Default: `numbered-footnotes`.

  The function takes an integer as an argument (starting with value 0) and returns the appropriate markup. You can create your own function, for example,

  ```
  footnote-numbering-function =
  #(lambda (x)
    #{ \markup \concat { "[" #(number->string (1+ x)) "]" } #})
  ```

`footnote-padding` (number)
: The padding between two footnotes. Default: 0.5 mm.

`footnote-separator-markup` (markup)
: Markup to separate the music from the footnotes, usually a horizontal line. The default value is defined in file `paper-defaults-init.ly`.

`horizontal-shift` (number)
: If set, the main content block of all pages (but not headers and footers) is shifted horizontally. Positive values shift to the right (on both even and odd pages). Default: 0 mm.

`in` (positive number)
: The inch unit, equal to 2.54 cm.

`in-note-padding` (number)
: The padding between two in-notes. Default: 0.5 mm.

`in-note-system-padding` (number)
: The padding between an in-note and the associated music system. Default: 0.5 mm.

`incipit-width` (positive number)
: The width of an incipit as created by the `\incipit` command. Must be smaller than or equal to `indent`. Default: `(indent / 2)`.

`indent` (non-negative number)
: The indentation of the first system in a score. The space within the `line-width` available for the first system is reduced by this amount. Default: `indent-default` (15 mm, scaled to paper size).

`inner-margin` (non-negative number)
: The margin all pages have at the inner side if they are part of a book. Has only an effect if `two-sided` is set to `#t`. If unset, `inner-margin-default` (15 mm, scaled to paper size) is used in computations. Default: unset.

`is-last-bookpart` (boolean)
: To be documented.

`label-alist-table` (list)
: Internal. LilyPond uses this to construct a table of contents.

`label-page-table` (association list (list of pairs))
: Internal. LilyPond uses this to implement the `\with-link` functionality and to construct a table of contents.

`landscape` (boolean)
: Internal. Use `set-default-paper-size` or `set-paper-size` to set the paper orientation.

`last-bottom-spacing` (association list (list of pairs))
: The distance from the last system or top-level markup on a page to the bottom of the printable area (i.e., the top of the bottom margin). The default value is defined in file `paper-defaults-init.ly`.

`left-margin` (non-negative number)
: The margin between the left edge of the page and the start of the staff lines in unindented systems. If `left-margin` is not set, and both `line-width` and `right-margin` are set, then `left-margin` is set to `(paper-width - line-width - right-margin)`. If only `line-width` is set, then both margins are set to `((paper-width - line-width) / 2)`, and the systems are consequently centered on

the page. If unset, `left-margin-default` (15 mm, scaled to paper size) is used in computations. Default: unset.

If `two-sided` is set to `#t`, this value is ignored. Also see `check-consistency`.

`line-thickness` (positive number)
: This value globally defines the default line thickness LilyPond uses while constructing many non-glyph elements like lines or boxes. The actual line thickness is usually specified as a factor of this base value, either as a default value, or overridden by the user. The default depends on the staff space and is computed as `(0.328571 + 0.0342857 * staff-space)` (see function `calc-line-thickness` in file `paper.scm`), which is synchronized with the Emmentaler font.

`line-width` (positive number)
: The horizontal extent of the staff lines in unindented, non-ragged systems, equal to `(paper-width - left-margin - right-margin)` if not set. If `line-width` is set, and both `left-margin` and `right-margin` are not set, then the margins are updated to center the systems on the page automatically. Also see `check-consistency`. Default: unset.

  As a feature of LilyPond, if the command-line option `-dcrop` is set, the dimensions of the cropped output file are always rounded up to integer (PostScript) big points; this might cause a little bit of whitespace at the right margin. To avoid that, set `line-width` to an integer big point value like `450\bp`.

`make-footer` (procedure)
: Internal. This is what LilyPond actually uses for for handling `evenFooterMarkup` and `oddFooterMarkup`.

`make-header` (procedure)
: Internal. This is what LilyPond actually uses for for handling `evenHeaderMarkup` and `oddHeaderMarkup`.

`markup-markup-spacing` (association list (list of pairs))
: The distance between two (title or top-level) markups. The default value is defined in file `paper-defaults-init.ly`.

`markup-system-spacing` (association list (list of pairs))
: The distance between a (title or top-level) markup and the system that follows it. The default value is defined in file `paper-defaults-init.ly`.

`max-systems-per-page` (non-negative, exact integer)
: The maximum number of systems that are placed on a page. This is currently supported only by the `ly:optimal-breaking` algorithm. Default: unset.

`min-systems-per-page` (non-negative, exact integer)
: The minimum number of systems that are 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.

`mm` (positive number)
: The millimeter unit, approx. 0.039 in.

`number-footnote-table` (list)
: Internal. LilyPond uses this to manage footnotes and footnote numbers.

`oddFooterMarkup` (markup)
: The footer markup used for odd-numbered pages. The default value is defined in file `titling-init.ly`.

`oddHeaderMarkup` (markup)
: The header markup used for odd-numbered pages. The default value is defined in file `titling-init.ly`.

`orphan-penalty` (number)
: To be documented. See `lily/page-breaking.cc`. Default is 100000.

`outer-margin` (non-negative number)
: The margin all pages have at the outer side if they are part of a book. Has only an effect if `two-sided` is set to `#t`. If unset, `outer-margin-default` (15 mm, scaled to paper size) is used in computations. Default: unset.

`output-filename` (string)
: If set, use this value as the output file name for LilyPond output. See also `output-suffix`. The default is the input file name (without the file extension).

`output-scale` (positive number)
: Internal. This value globally defines the output scale LilyPond uses while creating output. Use `set-global-staff-size` or `layout-set-staff-size` to change the scaling. Default: 1.7573, which corresponds to a 20 pt staff size.

`output-suffix` (string)
: If set, use this value as the output file name suffix for LilyPond output, appended to the output name with a dash in between. See also `output-filename`. The default is no output suffix if there is only a single book block (either implicit or explicit), and a running integer starting with value 1 otherwise.

`page-breaking` (procedure)
: The page-breaking algorithm to use. Choices are `ly:minimal-breaking`, `ly:page-turn-breaking`, `ly:one-page-breaking`, `ly:one-line-breaking`, `ly:one-line-auto-height-breaking`, and `ly:optimal-breaking`. Default: `ly:optimal-breaking`.

`page-breaking-system-system-spacing` (association list (list of pairs))
: This tricks the page breaker into thinking that `system-system-spacing` is set to something different than it really is. For example, if `page-breaking-system-system-spacing.padding` is set to something substantially larger than `system-system-spacing.padding`, then the page breaker puts fewer systems on each page. Default: unset.

`page-count` (non-negative, exact integer)
: The number of pages to be used for a score. Default: unset.

`page-number-type` (symbol)
: The type of numerals used for page numbers. Choices include `arabic`, `roman-ij-lower`, `roman-ij-upper`, `roman-lower`, and `roman-upper`. Default: `arabic`.

`page-post-process` (procedure)
: If this function is defined (within the \paper block), LilyPond uses it to post-process pages, for example, to extract the table of contents and writing the information to an auxiliary file. The call happens after page breaking has been performed.

  The syntax is

  ```
  #(define (page-post-process layout pages) ... )
  ```

`page-spacing-weight` (number)
: When using the `ly:optimal-breaking` algorithm for page breaking, LilyPond has to make trade-offs between horizontal and vertical stretching so that the overall spacing

is more acceptable. This parameter controls the relative importance of (vertical) page spacing and (horizontal) line spacing. High values makes page spacing more important. Default: 10.

`paper-height` (positive number)
: The height of the page. Note that the automatic scaling of some vertical dimensions is not affected if you set this value directly. Default: 297 mm (A4 paper height).

`paper-width` (positive number)
: The width of the page. While setting `paper-width` directly has no effect on the automatic scaling of some horizontal dimensions, it does influence the `line-width` variable. If both `paper-width` and `line-width` are set, then `left-margin` and `right-margin` also get updated. Also see `check-consistency`. Default: 210 mm (A4 paper width).

`papersizename` (string)
: Internal. Use `set-default-paper-size` or `set-paper-size` to set the paper size.

`print-all-headers` (boolean)
: If set to `#t`, print all headers for each `\score` in the output. If set to `#f`, only the `piece` and `opus` header variables are printed. Default: `#f`.

`print-first-page-number` (boolean)
: If set to `#t`, a page number is printed on the first page. Default: `#f`.

`print-page-number` (boolean)
: If set to `#f`, page numbers are not printed. Default: `#t`.

`property-defaults` (association list (list of pairs))
: This variable holds some settings that are used for top-level markups and as fallback values if nothing else is specified, for example, the standard fonts or the baseline skip. The default value is defined in file `paper-defaults-init.ly`.

`pt` (positive number)
: The point unit, equal to approx. 0.351 mm (0.0139 in).

`ragged-bottom` (boolean)
: If set to `#t`, systems are set at at their natural spacing, neither compressed nor stretched vertically to fit the page. Default: `#f`.

`ragged-last` (boolean)
: If set to `#t`, the last system in the score does not fill the line width. Instead, the last system ends at its natural horizontal length. Default: `#f`.

`ragged-last-bottom` (boolean)
: If set to `#f`, then the last page, and the last page in each section created with a `\bookpart` block, is vertically justified in the same way as the earlier pages. Default: `#t`.

`ragged-right` (boolean)
: If set to `#t`, systems don't fill the line width. Instead, systems end at their natural horizontal length. Default: `#t` for scores with only one system, and `#f` for scores with two or more systems.

`reset-footnotes-on-new-page` (boolean)
: If set to `#t`, footnote and in-note numbers are reset on each page break. For footnotes and in-notes numbered consecutively across page breaks, set to `#f`. Default: `#t`.

`right-margin` (non-negative number)
: The margin between the right edge of the page and the end of the staff lines in non-ragged systems. If `right-margin` is not set, and both `line-width` and

left-margin are set, then right-margin is set to (paper-width - line-width - left-margin). If only line-width is set, then both margins are set to ((paper-width - line-width) / 2), and the systems are consequently centered on the page. If unset, right-margin-default (15 mm, scaled to paper size) is used in computations. Default: unset.

If two-sided is set to #t, this value is ignored. Also see check-consistency.

score-markup-spacing (association list (list of pairs))
: The distance between the last system of a score and the (title or top-level) markup that follows it. The default value is defined in file paper-defaults-init.ly.

score-system-spacing (association list (list of pairs))
: The distance between the last system of a score and the first system of the score that follows it, if no (title or top-level) markup exists between them. The default value is defined in file paper-defaults-init.ly.

score-title (procedure)
: Internal. This is what LilyPond actually uses for handling scoreTitleMarkup.

score-title-properties (association list (list of pairs))
: Internal.

scoreTitleMarkup (markup)
: The titling markup of a score within a \book, using standard fields from the \header block. The default value is defined in file titling-init.ly.

short-indent (non-negative number)
: The level of indentation for all systems in a score besides the first system. The space within the line-width available for systems other than the first one is reduced by this amount. Default: short-indent-default (0 mm).

staff-height (positive number)
: Internal. Use set-global-staff-size or layout-set-staff-size to set the staff height. Default: 20 pt.

staff-space (positive number)
: Internal. One staff height contains four staff spaces. Use set-global-staff-size or layout-set-staff-size to set this value. Default: 5 pt.

system-count (non-negative, exact integer)
: The number of systems to be used for a score. Default: unset.

system-separator-markup (markup)
: A markup object that is inserted between systems, often used for orchestral scores. The \slashSeparator markup, defined in file titling-init.ly, is provided as a sensible standard value. Default: unset.

system-system-spacing (association list (list of pairs))
: The distance between two systems in the same score. The default value is defined in file paper-defaults-init.ly.

systems-per-page (non-negative, exact integer)
: The number of systems that should be placed on a page. This is currently supported only by the ly:optimal-breaking algorithm. Default: unset.

tagline (markup)
: This markup gets placed at the bottom of the last page (via oddFooterMarkup). The default value is defined in file titling-init.ly.

`text-font-size` (positive number)
: This value globally defines the standard text size for markups. The default depends on the staff height and is computed as `(staff-height / 20 * 11)`.

`tocFormatMarkup` (procedure)
: How the top-level entries of the table of contents are formatted (if there are several hierarchical levels). This variable holds a function like `make-bold-markup`. The default value is defined in file `toc-init.ly`.

`tocIndentMarkup` (markup)
: How the outline's hierarchy in the table of contents is made apparent. This markup is printed zero, one, or several times depending on the level of each entry. The default value is defined in file `toc-init.ly`.

`tocItemMarkup` (markup)
: How an item in the table of contents is formatted. The default value of this markup is defined in file `toc-init.ly`.

`tocTitleMarkup` (markup)
: How the title of the table of contents is formatted. The default value of this markup is defined in file `toc-init.ly`.

`top-margin` (non-negative number)
: The margin between the top of the page and the top of the printable area. Default: `top-margin-default` (10 mm, scaled to paper size).

`top-markup-spacing` (association list (list of pairs))
: The distance from the top of the printable area (i.e., the bottom of the top margin) to the first (title or top-level) markup on a page, when there is no system between the two. The default value is defined in file `paper-defaults-init.ly`.

`top-system-spacing` (association list (list of pairs))
: The distance from the top of the printable area (i.e., the bottom of the top margin) to the first system on a page, when there is no (title or top-level) markup between the two. The default value is defined in file `paper-defaults-init.ly`.

`two-sided` (boolean)
: If set to `#t`, use `inner-margin`, `outer-margin` and `binding-offset` to determine margins depending on whether the page number is odd or even. `left-margin` and `right-margin` are then ignored. Default: `#f`.

B.24 Naming conventions

Naming conventions are used so that it is easier to keep the nature of an object in mind.

Object/property type	**Naming convention**	**Examples**
Keywords	`aaaa`	`\score`, `\addlyrics`
Music functions and shortcuts	`aaaa` or `aaaaAaaaAaaa`	`\time`, `\stemUp`
Markup functions	`aaaa` or `aaaa-aaaa-aaaa`	`\bold`, `\with-dimensions`
Paper variables	`aaaa` or `aaaa-aaaa-aaaa`	`indent`, `page-breaking`
Music expressions	`Aaaa` or `AaaaAaaaAaaa`	`NoteEvent`, `SequentialMusic`
Music classes	`aaaa` or `aaaa-aaaa-aaaa`	`note-event`, `rhythmic-event`
Music properties	`aaaa` or `aaaa-aaaa-aaaa`	`force-accidental`, `iterator-ctor`
Contexts	`Aaaa` or `AaaaAaaaAaaa`	`Staff`, `GrandStaff`

Context properties	aaaa or aaaaAaaaAaaa	stanza, instrumentName
Engravers	Aaaa_engraver or Aaaa_aaaa_aaaa_engraver	Clef_engraver, Note_heads_engraver
Layout Objects	Aaaa or AaaaAaaaAaaa	Slur, NoteHead
Layout Object Properties	aaa or aaa-aaa-aaa	direction, beam-thickness
Interfaces	aaa-aaa-interface	grob-interface, break-aligned-interface
Scheme functions defined in Guile	aaaa or aaaa-aaaa-aaaa	map, for-each
Scheme functions defined in LilyPond	ly:aaaa or ly:aaaa-aaaa-aaaa or aaaa or aaaa-aaaa-aaaa[1]	ly:round-filled-box, music-map

B.25 Predefined type predicates

Predicates return #t (true) if their argument is of the named type and #f (false) if it isn't.

B.25.1 R5RS primary predicates

Primary predicates can be applied to any expression. They can be used on their own as predicates for LilyPond functions. The predicates here are part of the Scheme standard R5RS.

Type predicate	Description
boolean?	boolean
char?	character
complex?	complex number
eof-object?	end-of-file object
input-port?	input port
integer?	integer
list?	list (use cheap-list? for faster processing)
null?	null
number?	number
output-port?	output port
pair?	pair
port?	port
procedure?	procedure
rational?	rational number
real?	real number
string?	string
symbol?	symbol
vector?	vector

B.25.2 R5RS secondary predicates

Secondary predicates are only applicable to specific expressions (for example, to numbers). They will throw a type error when applied to expressions they are not intended for. The predicates here are part of the Scheme standard R5RS.

Type predicate	Description
char-alphabetic?	alphabetic character
char-lower-case?	lower-case character
char-numeric?	numeric character

[1] For historical reasons, some but not all of the functions defined by LilyPond start with the ly: prefix.

char-upper-case?	upper-case character
char-whitespace?	whitespace character
even?	even number
exact?	exact number
inexact?	inexact number
negative?	negative number
odd?	odd number
positive?	positive number
zero?	zero

B.25.3 Guile predicates

These predicates are defined by Guile but are not part of a Scheme standard.

Type predicate	Description
hash-table?	hash table

B.25.4 LilyPond scheme predicates

These predicates are only available within LilyPond and defined in Scheme.

Type predicate	Description
alist?	association list (list of pairs)
boolean-or-symbol?	boolean or symbol
cheap-list?	list (use this instead of list? for faster processing)
color?	color
exact-rational?	an exact rational number
fraction?	fraction, as pair
grob-list?	list of grobs
grouped-number-list?	list of non-empty lists of numbers
index?	non-negative, exact integer
index-or-markup?	index or markup
key?	index or symbol
key-list?	list of indices or symbols
key-list-or-music?	key list or music
key-list-or-symbol?	key list or symbol
ly:skyline-pair?	pair of skylines
markup?	markup
markup-command-list?	markup command list
markup-list?	markup list
moment-pair?	pair of moment objects
musical-length?	non-negative exact rational, fraction (as pair), moment, or +inf.0
musical-length-as-moment?	non-negative moment with no grace part
musical-length-as-number?	non-negative exact rational or +inf.0
non-negative-number?	non-negative number
number-list?	number list
number-or-grob?	number or grob
number-or-number-pair?	number or pair of numbers
number-or-pair?	number or pair
number-or-string?	number or string
number-pair?	pair of numbers
number-pair-list?	list of number pairs
optionally-grouped-beat-structure?	a number list or a list of them

positive-exact-rational?	a positive, exact rational number
positive-fraction?	positive, finite fraction, as pair
positive-musical-length?	positive exact rational, fraction (as pair), moment, or +inf.0
positive-musical-length-as-moment?	positive moment with no grace part
positive-musical-length-as-number?	positive exact rational or +inf.0
positive-number?	positive number
rational-or-procedure?	an exact rational or procedure
rhythmic-location?	rhythmic location
sane-simple-time-signature?	simple time signature
sane-time-signature?	time signature
scale?	non-negative rational, fraction, or moment
scheme?	any type
string-or-music?	string or music
string-or-pair?	string or pair
string-or-symbol?	string or symbol
symbol-key-alist?	alist, with symbols as keys
symbol-list?	symbol list
symbol-list-or-music?	symbol list or music
symbol-list-or-symbol?	symbol list or symbol
time-signature?	time signature
void?	void
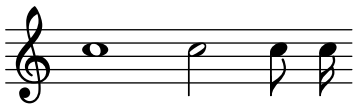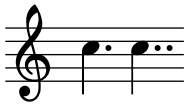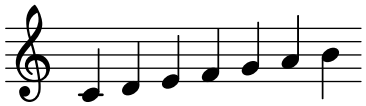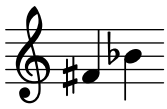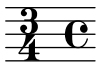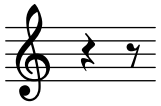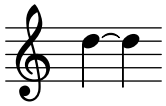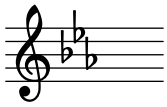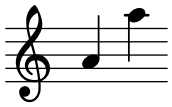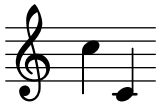
B.25.5 LilyPond exported predicates
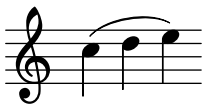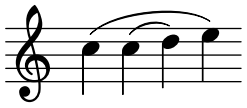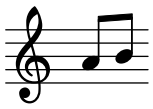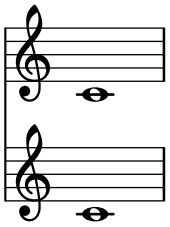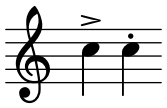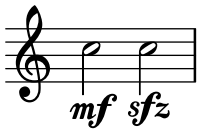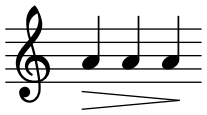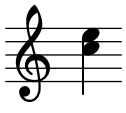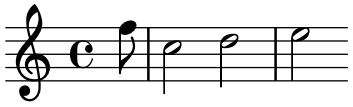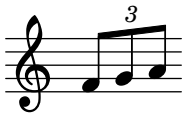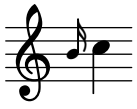
These predicates are only available within LilyPond and usually defined in C++.

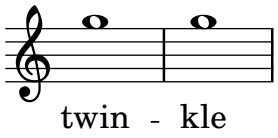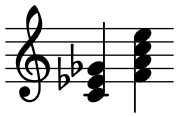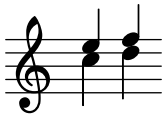
Type predicate	**Description**
ly:book?	book
ly:book-or-bookpart?	book or bookpart
ly:bookpart?	bookpart
ly:context?	context
ly:context-def?	context definition
ly:context-mod?	context modification
ly:dimension?	dimension, in staff space
ly:dir?	direction
ly:dispatcher?	dispatcher
ly:duration?	duration
ly:event?	post-event
ly:font-metric?	font metric
ly:grob?	graphical (layout) object
ly:grob-array?	array of grobs
ly:grob-properties?	grob properties
ly:input-location?	input location
ly:item?	item
ly:iterator?	iterator
ly:lily-lexer?	lily-lexer
ly:lily-parser?	lily-parser
ly:listener?	listener
ly:moment?	moment
ly:music?	music
ly:music-function?	music function
ly:music-list?	list of music objects

`ly:music-output?`	music output
`ly:no-callback?`	procedure, no property callback
`ly:note-scale?`	note scale
`ly:otf-font?`	OpenType font
`ly:output-def?`	output definition
`ly:page-marker?`	page marker
`ly:pango-font?`	Pango font
`ly:paper-book?`	paper book
`ly:paper-system?`	paper-system Prob
`ly:pitch?`	pitch
`ly:prob?`	property object
`ly:score?`	score
`ly:skyline?`	skyline
`ly:source-file?`	source file
`ly:spanner?`	spanner
`ly:spring?`	spring
`ly:stencil?`	stencil
`ly:stream-event?`	stream event
`ly:transform?`	coordinate transform
`ly:translator?`	translator
`ly:translator-group?`	translator group
`ly:unpure-pure-container?`	unpure/pure container

Appendix C Cheat sheet

Syntax	Description	Example
`1 2 8 16`	durations	
`c4. c4..`	augmentation dots	
`c d e f g a b`	scale	
`fis bes`	alteration	
`\clef treble \clef bass`	clefs	
`\time 3/4 \time 4/4`	time signature	
`r4 r8`	rest	
`d ~ d`	tie	
`\key es \major`	key signature	
note`'`	raise octave	
note`,`	lower octave	

`c( d e)`	slur	
`c\( c( d) e\)`	phrasing slur	
`a8[ b]`	beam	
`<< \new Staff ... >>`	more staves	
`c-> c-.`	articulations	
`c2\mf c\sfz`	dynamics	
`a\< a a\!`	crescendo	
`a\> a a\!`	decrescendo	
`< >`	chord	
`\partial 8`	pickup / upbeat	
`\tuplet 3/2 {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	
`\new ChordNames`	printing chord names	C° F△
`<<{e f} \\ {c d}>>`	polyphony	
`s4 s8 s16`	spacer rests	

Appendix D GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
`https://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. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, 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 `https://www.gnu.org/licenses/`.

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 Index

In addition to all the LilyPond commands and keywords, this index lists musical terms and words that relate to each of them, with links to those sections of the manual that describe or discuss that topic.

!

! 8, 529
\! 160, 530

"

"|" 141

#

#f (false) 974
#t (true) 974

%

% 592, 598
\% 209, 963
%{ ... %} 592, 598

’

' 3

(

(..... 171
\(..... 174

)

) 171
\) 174

*

* 61
* 189, 964

+

+ 529
\+ 529

,

, ... 3

–

- 156, 529, 774
\- 535
-! 928
-+ 930
-- 357, 928
-. 928
-> 928
-^ 928
-_ 928

.

. 53

/

/ 61, 512, 529
/+ 512

:

: 213, 509

<

< 216
\< 160
<...> 216
<< 220
<< ... \\ ... >> 224
<< ... >> 220
<> 162, 217, 414

=

= 12
\= 171, 963

>

> 216
\> 160
>> 220

?

? 8

[

[..... 115
\[..... 540

]

] 115
\] 540

^

^ 511, 774
\^ 440

_

_ 353, 774
__ 357

\

\\ 224, 529

|

| 141

~

~ 63, 353
\~ 558, 563, 964

1

15ma 26
15mb 26

8

8va 26
8vb 26

A

a due part 233
\abs-fontsize 324, 809
\abs-hspace 822
\abs-vspace 822
absolute 3
\absolute 943
absolute dynamics 160
absolute octave entry 3
absolute octave specification 3
\accent 156, 928
'accent' articulation 156, 928
\accentus 557, 930
'accentus' Gregorian articulation 557, 930
\accepts 751, 752, 755
acciaccatura 148
\acciaccatura 148, 943
acciaccatura, multi-note 153
accidental 8, 31
\accidental 851
accidental glyph set 922
accidental style 31
accidental style, choral 35
accidental style, choral-cautionary 35
accidental style, default 31, 33
accidental style, dodecaphonic 36
accidental style, dodecaphonic-first 37
accidental style, dodecaphonic-no-repeat 37
accidental style, forget 37
accidental style, modern 33, 34
accidental style, modern-cautionary 33, 34
accidental style, modern-voice-cautionary 34
accidental style, neo-modern 35
accidental style, neo-modern-cautionary 36
accidental style, neo-modern-voice 36
accidental style, neo-modern-voice-cautionary 36
accidental style, no-reset 37
accidental style, piano 35
accidental style, piano-cautionary 35
accidental style, teaching 37
accidental style, voice 33
accidental, alternate glyphs 39
accidental, and simultaneous notes 38
accidental, automatic 31
accidental, cautionary 8
accidental, Gregorian 556
accidental, hiding, on tied notes at start of system . 9
accidental, in cadenzas 93
accidental, in chords 38
accidental, in unmetered music 93
accidental, Kievan 565
accidental, mensural 546
accidental, multi-voice 34
accidental, musica ficta 547
accidental, on tied note 8
accidental, parenthesized 8
accidental, quarter tone 10
accidental, reminder 8
\accidentalStyle 31, 943
AccidentalSuggestion 159
accordion 424
accordion discant symbol 424
accordion register symbol 424
accordion shift symbol 424
acoustic bass 931
acoustic snare 931
add-grace-property 151
add-stem-support 284
add-toc-item! 626
\addChordShape 470, 943
adding custom fret diagram 469
adding white background, to text 884
\addInstrumentDefinition 943
addition, in chords 511
additional voices in polyphonic music 230
additionalPitchPrefix 517
additive time signature 81
\addlyrics 347, 349, 350
\addQuote 266, 943
\addToTagGroup 636, 943
adjusting start and end, slur 172
Aeolian 24
\aeolian 24
\after 161, 210, 943
\afterGrace 149, 944
afterGraceFraction 941
agogo 931
Aiken shape note head 46
Aiken shape note head, thin variant 48
\aikenHeads 46

\aikenHeadsMinor ... 47
\aikenThinHeads ... 46
\aikenThinHeadsMinor ... 47
al coda ... 197
al fine ... 196
al niente, hairpin ... 165
\alias ... 751, 752
align to object ... 797
\align-on-other ... 822
alignAboveContext ... 256, 360, 756
alignBelowContext ... 256, 360, 369, 756
aligning markup ... 327
aligning markup text ... 327
aligning objects ... 791
aligning text ... 327
aligning to cadenza ... 153
alignment point ... 792
alignment, bar numbers ... 139
alignment, breaking vertical, dynamics ... 167
alignment, breaking vertical, text scripts ... 167
alignment, horizontal, lyrics ... 364
alignment, horizontal, melisma ... 365
alignment, text, commands ... 332
alignment, vertical, dynamics ... 166
alignment, vertical, text ... 328
alignment, vertical, text scripts ... 166
alist ... 940
alla breve ... 81, 82
alla coda ... 197
\allowBreak ... 121, 689
\allowPageTurn ... 696, 944
\allowVoltaHook ... 944
alpha transparency ... 290
alteration, figured bass, glyphs ... 530
alteration, figured bass, position ... 530
\alterBroken ... 777, 944
altered chord ... 510
alternate accidental glyph ... 39
alternate fretboard table ... 475
alternating time signature ... 81
\alternative ... 191, 193
alternative chord name ... 516
alternative ending ... 191
alternative ending, and lyrics ... 370
alternative ending, with ties ... 64
alternative melody, switching to ... 384
alternative style, of breve notes ... 54
alto clef ... 19, 923
alto varC clef ... 923
'Amazing Grace', bagpipe example ... 498
ambitus ... 40
ambitus, line gap ... 42
ambitus, multiple voices ... 41
ambitus, per voice ... 41
ambitus, placement ... 43
ambitusAfter ... 43, 944
AmbitusLine ... 42
anacrusis ... 92
anacrusis, in a repeat ... 191
analysis bracket, musicological ... 304
analysis bracket, with label ... 305
ancient clef ... 19
ancient music clef ... 543, 924
angled hairpin ... 791
annotate moving by spacing, in text ... 874
\annotate-moving ... 874
annotate-spacing ... 729, 965
anthem ... 389
\append-to-tag ... 639, 875
\appendToTag ... 637, 944
\appendToTagMarkup ... 639, 944
\applyContext ... 742, 944
\applyMusic ... 944
\applyOutput ... 944
\applySwing ... 666
\applySwingWithOffset ... 666
appoggiatura ... 148
\appoggiatura ... 148, 944
approximate pitch ... 10
\approximatePitch ... 10, 944
Arabic improvisation ... 580
Arabic key signatures ... 577
Arabic maqam ... 575
Arabic music ... 575
Arabic music example ... 580
Arabic music template ... 580
Arabic note name ... 576
Arabic semi-flat symbol ... 577
Arabic time signatures ... 579
\arabicStringNumbers ... 432
arpeggio ... 182
\arpeggio ... 182
arpeggio symbol, special ... 183
arpeggio, and ties ... 65
arpeggio, cross-staff parenthesis-style ... 186
arpeggio, cross-voice ... 185
arpeggio-direction ... 182
\arpeggioArrowDown ... 182, 944
\arpeggioArrowUp ... 182, 944
\arpeggioBracket ... 183, 944
\arpeggioNormal ... 182, 945
\arpeggioParenthesis ... 183, 945
\arpeggioParenthesisDashed ... 183, 945
arranger ... 603
Arrastre ... 287
\arrow-head ... 333, 840
\articulate ... 665
articulate.ly script ... 665
articulation ... 928
articulation, accent ... 156, 928
articulation, bachschleifer ... 156
articulation, default values, modifying ... 158
articulation, espressivo ... 156, 928
articulation, Gregorian ... 557
articulation, Gregorian, accentus ... 557, 930
articulation, Gregorian, circulus ... 557, 930
articulation, Gregorian, episem finis ... 557
articulation, Gregorian, episem initium ... 557
articulation, Gregorian, ictus ... 557, 930
articulation, Gregorian, semicirculus ... 557, 930
articulation, half-open ... 156, 930
articulation, marcato ... 156, 928
articulation, open ... 156, 496, 930
articulation, portato ... 156, 928
articulation, staccatissimo ... 156, 928
articulation, staccato ... 156, 928
articulation, stopped ... 156, 496, 930
articulation, tenuto ... 156, 928
articulation-event ... 268
artificial harmonics ... 429

\ascendens ... 558, 563
\assertBeamQuant ... 945
\assertBeamSlope ... 945
associatedVoice ... 347, 349, 384
association list ... 940
\atLeft ... 157, 945
\atRight ... 157, 945
\auctum ... 558, 563
aug ... 509
augmentation dot, changing number ... 54
\augmentum ... 563, 945
auto-beaming, properties for time signatures ... 83
auto-first-page-number ... 680, 965
\auto-footnote ... 875
autoBeaming ... 106, 736
\autoBeamOff ... 103, 420
\autoBeamOn ... 103
\autoBreaksOff ... 691
\autoBreaksOn ... 691
\autoChange ... 416, 945
\autoChange, and relative music ... 417
\autoLineBreaksOff ... 691
\autoLineBreaksOn ... 691
automatic accidental ... 31
automatic bar line ... 131
automatic chord diagram ... 477
automatic fret diagram ... 477
automatic part combining ... 233
automatic replacement, of strings ... 649
automatic staff change ... 416
\autoPageBreaksOff ... 693
\autoPageBreaksOn ... 693

B

Bézier curve, control points ... 801
bachschleifer ... 157
\bachschleifer ... 156
'bachschleifer' articulation ... 156
backslashed digit ... 875
\backslashed-digit ... 875
bagpipe ... 498
bagpipe, example ... 498
balance, in MIDI ... 664
balloon ... 300
balloon help ... 300
Balloon_engraver ... 300
\balloonGrobText ... 300, 945
\balloonLengthOff ... 300
\balloonLengthOn ... 300
\balloonText ... 300, 945
banjo tablature ... 431, 484
banjo tuning ... 485
banjo-c-tuning ... 485
banjo-double-c-tuning ... 485
banjo-double-d-tuning ... 485
banjo-modal-tuning ... 485
banjo-open-d-tuning ... 485
banjo-open-dm-tuning ... 485
banjo-open-g-tuning ... 485
\bar ... 121, 130, 945
bar check, with repeats ... 191
bar line ... 121
bar line, automatic ... 131
bar line, avoided by lyrics ... 366
bar line, between staves ... 126, 247
bar line, cadenzas ... 93
bar line, caesura ... 131, 132
bar line, default ... 132
bar line, default, changing ... 130
bar line, defining ... 126
bar line, dotted ... 131
bar line, double ... 121, 131, 132, 147
bar line, final ... 121, 132, 147
bar line, for repeat at start of piece ... 190
bar line, hairpin, stopping at ... 164
bar line, in ChordNames ... 520
bar line, in Gregorian chant ... 556
bar line, in hymns ... 409
bar line, in Kievan chant ... 566
bar line, in markup ... 851
bar line, invisible ... 121
bar line, list ... 933
bar line, manual ... 121
bar line, measure ... 132
bar line, phrase ... 131, 132
bar line, repeat ... 132
bar line, section ... 132, 147
bar line, segno ... 123, 132, 199
bar line, short ... 122
bar line, single ... 132
bar line, submeasure ... 131
bar line, symbols on ... 314
bar line, text over ... 314
bar line, tick ... 122
bar line, underlying repeat ... 131
bar line, unmetered music ... 93
bar number ... 78, 133
bar number, alignment ... 139
bar number, alternative, in repeat ... 203
bar number, broken measure ... 135
bar number, cadenzas ... 93
bar number, centered ... 140
bar number, check ... 141
bar number, collision ... 141
bar number, modulo bar number ... 136
bar number, printed at regular intervals ... 134
bar number, printed in first measure ... 134
bar number, regular spacing ... 133
bar number, removal ... 139
bar number, style ... 136
bar number, unmetered music ... 93
bar number, with letter, in repeat ... 203
bar, check ... 141
\bar-line ... 851
baritone clef ... 19, 923
baritone varC clef ... 923
baritone varF clef ... 923
BarNumber ... 133
\barNumberCheck ... 142, 945
barNumberVisibility ... 133
barré indication ... 456
barré line ... 481
Bartók pizzicato ... 429
bartype ... 130
base-shortest-duration ... 713
bass ... 931
bass clef ... 19, 923
bass note, for chords ... 512

bass, figured ... 526
bass, thorough ... 526
\bassFigureExtendersOff ... 529
\bassFigureExtendersOn ... 529
\bassFigureStaffAlignmentDown ... 532
\bassFigureStaffAlignmentNeutral ... 532
\bassFigureStaffAlignmentUp ... 532
basso continuo ... 526
\beam ... 840
beam, cadenzas ... 93
beam, cross-staff ... 414
beam, customizing rules ... 102
beam, endings, in a score ... 113
beam, endings, with multiple voices ... 113
beam, feathered ... 119
beam, horizontal ... 444
beam, in tablature ... 444
beam, line breaks ... 103
beam, manual ... 102, 115
beam, nibs ... 117
beam, \partCombine with \autoBeamOff ... 104
beam, slashed ... 120
beam, subdividing ... 110
beam, unmetered music ... 93
beam, with knee gap, changing ... 104
beam, with lyrics ... 106
beam, with melisma ... 103
beam, with polymetric meters ... 95
beam-like flags ... 118
\beamBreak ... 115
beamed tuplet, line break within ... 60
beamExceptions ... 83, 106, 945
beaming, strict-beat ... 111
beaming, time signature default properties ... 83
beamlet, orientation ... 111
beat repeat ... 209
beat, grouping ... 112
beatBase ... 83, 106, 110
beats per minute ... 87
beatStructure ... 83, 106, 110
beginners' music ... 45
\bendAfter ... 177, 945
\bendHold ... 440, 946
bending, string, in tablature ... 440
\bendStartLevel ... 440, 946
big point (bp) ... 775
binding gutter ... 678
binding-offset ... 678, 965
bisbigliando ... 425
bison program ... 941
bitmap fonts ... 338
black mensural clef ... 543, 924
\blackTriangleMarkup ... 517
blank-after-score-page-penalty ... 680, 965
blank-last-page-penalty ... 680, 965
blank-page-penalty ... 680, 965
block chord ... 184
block comment ... 592, 598
blot-diameter ... 965
BNF ... 941
\bold ... 324, 809
bongo ... 931
\book ... 593, 597
\book, and \layout ... 595
\book, and variables ... 594
book-title ... 965
book-title-properties ... 965
bookmarks ... 623, 625
\bookOutputName ... 946
\bookOutputSuffix ... 946
\bookpart ... 594, 597, 693
\bookpart, and \layout ... 595
\bookpart, and variables ... 594
bookpart-level-page-numbering ... 681, 965
bookTitleMarkup ... 608, 965
Bottom ... 739
bottom-margin ... 674, 965
bounding box ... 776
bounding box, of glyph ... 885
bowing indication ... 428
bowing, down ... 156, 930
bowing, up ... 156, 930
\box ... 332, 809
\bp ... 775, 965
brace, at start of single staff ... 246
brace, in markup ... 878
brace, nested ... 248
brace, vertical ... 244
bracket ... 296, 304
\bracket ... 169, 332, 840
bracket, angle ... 216
bracket, at start of single staff ... 246
bracket, cross-staff ... 420
bracket, horizontal ... 304
bracket, nested ... 248
bracket, non-arpeggiato ... 184
bracket, phrasing ... 304
bracket, phrasing, with label ... 305
bracket, square, at start of staff group ... 246
bracket, vertical ... 244
bracket, vertical, marking divided voices ... 391
bracket, volta ... 205
\break ... 689
break, in unmetered music ... 94
break-align symbol ... 718
break-align-orders ... 719
break-align-symbol ... 797, 939
break-align-symbols ... 797
break-alignable-interface ... 797
break-aligned-interface ... 797
break-visibility ... 786
breakable ... 103
breakable glissando ... 180
\breakAlignInsert ... 946
breakbefore ... 605
\breakDynamicSpan ... 167
breaking lines ... 689
breaking pages ... 722
breaking vertical alignment of dynamics ... 167
breath mark ... 176
breath mark symbol, changing ... 176
breath mark symbol, list ... 931
\breathe ... 176, 946
\breve ... 52, 67
breve note, alternative style ... 54
breve rest ... 67
broken chord ... 182
broken spanner, modifying ... 776

C

C clef 19, 923
cabasa 931
cadenza 93, 153
cadenza, accidentals 93
cadenza, aligning to 153
cadenza, bar lines 93
cadenza, bar numbers 93
cadenza, beams 93
cadenza, line breaks 94
cadenza, page breaks 94
\cadenzaOff 93, 946
\cadenzaOn 93, 946
caesura 76
\caesura 76, 77, 131, 132, 409, 557, 566, 946
caesura bar line 131, 132
caesura bar line, in hymns 409
caesura bar line, in Kievan chant 566
callback 940
canticle 400, 405
capo 461
\caps 810
cautionary accidental 8
\cavum 558, 563
\center-align 327, 823
\center-column 330, 823
CENTER 794, 796
centered dynamics in piano music 414
centering column of text 823
centering text on page 331
centimeter (cm) 775
CFF2, font format 337
\change 414
changing breath mark symbol 176
changing chord separator 521
changing direction of text column 824
changing font 324
changing instrument name 264
changing number of augmentation dots 54
changing properties 760, 766
changing staff automatically 416
changing staff manually 414
changing tempo without metronome mark 91
changing tuplet number 57
chant 400, 405
\char 876
character name 393
chart, fingering 496
check mark 176
check-consistency 677, 966
Chinese cymbal 931
choir staff 244
ChoirStaff 737
choral 35
'choral' accidental style 35
choral tenor clef 20
choral-cautionary 35
'choral-cautionary' accidental style 35
chord 216, 508, 513
chord bracket 184
chord bracket, cross-staff 420
chord diagram 455, 466
chord diagram, automatic 477
chord grid 522
chord mode 508
chord name 508, 513
chord name, exceptions 519
chord name, with fret diagrams 467
chord separator, changing 521
chord shape, for fretted instrument 470
chord slur 184
chord step, altering 511
chord, accidentals in 38
chord, alternative name 516
chord, and relative octave entry 7
chord, and tie 63
chord, block 184
chord, broken 182
chord, cross-staff 420
chord, empty 151, 162, 217, 414
chord, fingering 283
chord, glissando, in tablatures 449
chord, inversion 513
chord, jazz 516
chord, major 7, layout 520
chord, modifying a single note 765
chord, power 483
chord, quality 509
chord, relative pitch 217
chord, repetition 218, 435
chord, rolled 184
chord, rotating 513
chord, splitting across staves with \autoChange .. 418
chord, suppressing repeated 474, 515
chord, voicing 513
chord, volta repeat, below 520
chordChanges 474, 515
ChordGrid 522, 738
ChordGridScore 737
\chordmode 6, 16, 467, 589
chordNameExceptions 518, 519
chordNameLowercaseMinor 517
ChordNames 467, 739
ChordNames, with bar lines 520
chordNameSeparator 518, 521
chordNoteNamer 518
chordPrefixSpacer 519
\chordRepeats 435, 946
chordRootNamer 517
\chords 515, 589
\chordSlur 184
chorus level, in MIDI 664
Christian Harmony note head 46
church mode 24
church rest 276
\circle 332, 841
circling text 841
\circulus 557, 930
'circulus' Gregorian articulation 557, 930
claves 931
clef 8, 19, 555, 565
\clef 19, 946
clef, alto 19, 923
clef, alto varC 923
clef, ancient 19
clef, ancient music 543, 924
clef, baritone 19
clef, baritone varC 923
clef, baritone varF 923

clef, bass 19, 923
clef, black mensural 543, 924
clef, C 19, 923
clef, F 19, 923
clef, french 19, 923
clef, G 19, 923
clef, G2 923
clef, GG 923
clef, Kievan 543, 924
clef, mensural 543, 924
clef, mezzo-soprano 19, 923
clef, moderntab 454
clef, percussion 487, 924
clef, Petrucci 543, 924
clef, soprano 19, 923
clef, style 543, 923
clef, subbass 19, 923
clef, tab 454, 924
clef, tenor 19, 923
clef, tenor G 923
clef, tenor varC 923
clef, transposing 20
clef, treble 19, 923
clef, tweaking properties 22
clef, varbaritone 19, 923
clef, varC 923
clef, violin 19, 923
clef, visibility following explicit change 788
clef, visibility of transposition 790
clef, with cue notes 20
clip-regions 650, 966
closure 940
cluster 221
\cm 775, 966
coda 197
\coda 156, 853, 930
'coda' sign 156, 930
'coda' sign, manual 207
'coda' sign, variant 156, 201, 930
'coda' sign, with repeats 196
\codaMark 131, 199, 207, 946
collision 227
collision, bar number 141
collision, cross-staff voices 415
collision, ignoring 220, 232
color 289
color 289
color blind 291
color fonts 338
color, CSS-like codes 290
color, in chord 291
color, list 902
color, RGB 291
color, suitable for dichromatism 906
coloring note 289
coloring note, in chord 291
coloring object 289, 785
coloring text 885
coloring voice 227
column 717
\column 330, 823
column, text 319, 330
\column-lines 886
\combine 333, 824
combining parts 233
combining parts, changing text 237
comma interval 581
comment 592, 598
common time 81
common-shortest-duration 713
Completion_heads_engraver 98, 572
Completion_rest_engraver 98, 572
complex time signature 81
composer 603
\compound-meter 853
\compressEmptyMeasures 274
compressing music 62
\compressMMRests 71, 73, 275, 947
\concat 824
concatenating text 824
condensing rests 75
conducting signs 112
conga 931
\consists 745, 751, 752
constante hairpin 166
contemporary glissando 179
\context 739, 746
context definitions, with MIDI 659
context property, changing default 746
context, adding alias 752
context, creating and referencing 739
context, defining in MIDI 753
context, defining new 751
context, implicit 756
\context, in \layout block 746
context, keeping alive 742
context, layout order 755
context, lifetime 742
context, overlaying multiple 245
context, properties 760
context, Score-level 754
context-spec-music 230
\contextPropertyCheck 947
continuation, of notes 535
control pitch 12
control point, Bézier curve 801
control point, tweaking 766
controlling general text alignment 826
controlpitch 12
copyright 603
copyright sign 648
count visibility, of percent repeats 210
counter, percent repeat 210
countPercentRepeats 210
cowbell 931
\cr 160
crash cymbal 931
creating a table 887
creating empty text object 879
creating horizontal space, in text 822, 828
creating text fraction 877
creating vertical space, in text 822, 837, 883
\cresc 162
crescendo 160
crescendo-event 268
crescendoSpanner 168
crescendoText 168
\crescHairpin 162
\crescTextCresc 162
cross 43

cross note head ... 43
cross-staff beam ... 414
cross-staff chord ... 420
cross-staff chord bracket ... 420
cross-staff collision ... 415
cross-staff line ... 418
cross-staff note ... 414, 420
cross-staff parenthesis-style arpeggio ... 186
cross-staff stem ... 420
cross-staff tremolo ... 213
cross-voice arpeggio ... 185
\crossStaff ... 420, 947
CSS color codes ... 290
cue clef ... 20
cue note ... 266, 269
cue note, formatting ... 269
cue note, removing ... 273
cue, in vocal score ... 397
cue, musical ... 395
\cueClef ... 269, 947
\cueClefUnset ... 269, 947
\cueDuring ... 270, 947
\cueDuring, and tags ... 632
\cueDuringWithClef ... 270, 947
CueVoice ... 269, 739
cuica ... 931
currentBarNumber ... 78, 133
custodes ... 541
custom fret diagram ... 455, 464
custom fret diagram, adding ... 469
custom fretboard fret diagram ... 471
custom metronome mark markup ... 91
custom rehearsal mark ... 143
custom string tuning ... 452
customized fret diagram ... 463
customizing chord name ... 516
\customTabClef ... 855
custos ... 541
cut time ... 81
cymbal, various ... 931

D

D.C. ... 197
D.C., manual ... 208
D.S. ... 198
D.S., forced at start ... 199
D.S., manual ... 208
da capo ... 197
da capo, manual ... 208
dal segno ... 198
dal segno, forced at start ... 199
dal segno, manual ... 208
dampened note, on fretted instrument ... 482
dash patterns, slur ... 172, 173
\dashBang ... 158
\dashDash ... 158
\dashDot ... 158
dashed phrasing slur ... 175
dashed slur ... 171
dashed tie ... 64
\dashHat ... 158
\dashLarger ... 158
\dashPlus ... 158
\dashUnderscore ... 158
dead note, percussion ... 494
\deadNote ... 44, 947
\deadNotesOff ... 44
\deadNotesOn ... 44
debug-beam-scoring ... 682, 966
debug-slur-scoring ... 682, 966
debug-tie-scoring ... 682, 967
debugging, vertical spacing ... 729
decorating text ... 332
\decr ... 160
\decresc ... 162
decrescendo ... 160
decrescendoSpanner ... 168
decrescendoText ... 168
dedication ... 603
default ... 33
\default ... 142, 614
'default' accidental style ... 31, 33
default bar line ... 132
default bar line, changing ... 130
default context property, changing ... 746
default note duration ... 53
default note name ... 8
default pitch ... 53
default vertical direction (-) ... 774
default, offsetting ... 766
default-staff-staff-spacing ... 697
\defaultchild ... 738, 756
\defaultTimeSignature ... 81
\defineBarLine ... 126, 947
defining bar line ... 126
defining predefined fretboard ... 472
delayed 'turn' ornament ... 159
\deminutum ... 558, 563
\denies ... 751, 753, 755
\descendens ... 558, 563
desk parts ... 262
Devnull ... 738, 739
diagram, chord for fretted instruments ... 455
diagram, fret ... 455
diagram, fret, customized ... 463
diagram, fret, transposing ... 468
diamond note head ... 43
diamond-shaped note head ... 428
dichromatism, suitable colors ... 906
dim ... 509
\dim ... 162
dimension ... 776
dimension units ... 775
dimension, of bounding box ... 885
dimension-variables ... 967
\dimHairpin ... 162
diminuendo ... 160
\dimTextDecr ... 162
\dimTextDecresc ... 162
\dimTextDim ... 162
\dir-column ... 824
direction ... 793
direction, automatic, of stem, on center line ... 298
direction, default (-) ... 774
direction, default, of stem, on center line ... 297
direction, down (_) ... 774
direction, up (^) ... 774
\discant ... 869

discant symbol, accordion 424
display vertical layout 729
\displayLilyMusic 667, 947
\displayMusic 947
\displayScheme 947
distance, absolute 775
distance, between staves 697
distance, scaled 775
divided lyrics 374
divided voices 391
divisi staves 260
divisio 556
\divisioMaior 557
\divisioMaxima 557
\divisioMinima 557
divisiones 556
dodecaphonic 36
'dodecaphonic' accidental style 36
dodecaphonic-first 37
'dodecaphonic-first' accidental style 37
dodecaphonic-no-repeat 37
'dodecaphonic-no-repeat' accidental style 37
doit 177
Dorian 24
\dorian 24
\dotsDown 54
\dotsNeutral 54
\dotsUp 54
dotted bar line 131
dotted note 53
dotted note, changing number of dots 54
dotted note, moving horizontally 231
dotted phrasing slur 175
dotted slur 171
dotted tie 64
double bar line 121, 131, 132, 147
double flat 8
double sharp 8
double slur, for legato chords 173
double-dotted note 53
double-time signature 95
\doubleflat 855
\doublesharp 855
doubleSlurs 173
down direction (_) 774
'down pralltriller' ornament 156, 928
'down-bow' bowing indication 156, 428, 930
DOWN 793, 796
\downbow 156, 428, 930
\downmordent 156, 928
\downprall 156, 928
\draw-circle 333, 841
\draw-dashed-line 842
\draw-dotted-line 842
\draw-hline 842
\draw-line 333, 843
\draw-squiggle-line 843
drawing beam, within text 840
drawing box, with rounded corners 844
drawing box, with rounded corners, around text . 849
drawing circle, within text 841
drawing dashed line, within text 842
drawing dotted line, within text 842
drawing ellipse, around text 844
drawing graphic object 332
drawing line, across a page 842
drawing line, within text 843
drawing oval, around text 845
drawing path 846
drawing polygon 847
drawing solid box, within text 844
drawing squiggled line, within text 843
drawing triangle, within text 850
\dropNote 513, 948
drum 487, 489
drum staff 242
drum, various 931
\drummode 242, 487, 589
drumPitchNames 492
drumPitchTable 492
\drums 487, 589
DrumStaff 242, 737
drumStyleTable 491
DrumVoice 739
duration line 535
duration, default 53
duration, isolated 53
duration, of note 52
duration, scaling 61
\dwn 577
\dynamic 169, 810
dynamic mark, multiple on one note 161
dynamic mark, new 168
dynamic-event 268
\dynamicDown 163
DynamicLineSpanner 163, 166
\dynamicNeutral 163
dynamics 160
Dynamics 739
dynamics, absolute 160
dynamics, breaking vertical alignment 167
dynamics, centered in keyboard music 414
dynamics, editorial 169
dynamics, extender line, hiding 167
dynamics, in MIDI 656
dynamics, in MIDI, custom 656
dynamics, parenthesis 169
dynamics, text spanner, customize 313
dynamics, text, style 168
dynamics, vertical alignment 166
dynamics, vertical positioning 163
\dynamicUp 163

E

easy notation 45
easy notation, using numbers 45
easy play note head 45
\easyHeadsOff 45
\easyHeadsOn 45
editorial dynamics 169
effects, in MIDI 664
electric snare 931
\ellipse 844
embedded graphics 334
embedding files, in PDF 652
embedding graphic object 332
Emmentaler font 907
Emoji fonts 338

empty chord 151, 162, 414
empty staff 257
\EnableGregorianDivisiones 964
\enablePerStaffTiming 948
encapsulated postscript output 651
enclosing text, in box 809
enclosing text, in box with rounded corners 849
encoding, of text 647
end repeat 205
\endcr 160
\enddecr 160
\endSpanners 784, 948
\englishChords 517
engraver, including in context 751
Engraver_group 752
ensemble music 393
entering lyrics 346
entering multiple tuplets 57
'episem finis' Gregorian articulation 557
'episem initium' Gregorian articulation 557
\episemFinis 557
\episemInitium 557
EPS image 845
EPS output 651
\epsfile 334, 844
equalization, instrument, replacing MIDI default . 658
\espressivo 156, 162, 928
'espressivo' articulation 156, 162, 928
\etc 323, 643
evenFooterMarkup 967
evenHeaderMarkup 967
\eventChords 948
exact positioning of staves 704
exception, chord names 519
\expandEmptyMeasures 274
expanding music 62
explicitClefVisibility 788
explicitKeySignatureVisibility 788
expression markup 320
expression, in MIDI 664
extended chord 510
extender 357
extender line, dynamics, hiding 167
extent, of actual inking 885
extent, of bounding box 885
extent, of grobs 791
extra natural sign, preventing 9
extra voice, for handling breaks 692
extra-offset 697
extra-spacing-height 720
extra-spacing-width 720
extreme pitch 10
\eyeglasses 876
Ez_numbers_engraver 45

F

\f 160
F clef 19, 923
fall 177
false (#f) 974
false note, percussion 494
\featherDurations 119, 948
feathered beam 119
feature, OpenType font 340, 815
fermata 156, 929
\fermata 77, 156, 855, 929
fermata, Henze 156, 929
fermata, long 156, 929
fermata, on bar line 77
fermata, on multi-measure rest 72
fermata, short 156, 929
Ferneyhough hairpin 166
Feta font 907
\ff 160
\fff 160
\ffff 160
\fffff 160
figured bass 526
figured bass, alignment 533
figured bass, alteration, glyphs 530
figured bass, alteration, position 530
figured bass, extender lines 530
figured bass, vertical distance of elements 532
\figured-bass 810
FiguredBass 739
figuredBassAlterationDirection 530
figuredBassPlusDirection 530
figuredBassPlusStrokedAlist 530
\figuremode 527, 589
\figures 527, 589
files, embedding in PDF 652
\fill-line 331, 825
\fill-with-pattern 627, 826
\filled-box 333, 844
final bar line 121, 132, 147
final bar line, in Gregorian chant 556
final bar line, in Kievan chant 566
finalis 556
\finalis 557
fine 196
\fine 132, 147, 195, 196, 566
\finger 282, 811, 948
finger change 282
fingering 282
fingering chart 496
fingering number, vs. string 432
fingering, adding to fret diagram 478
fingering, and multi-measure rest 75
fingering, glide 285
fingering, inside of staff 284
fingering, line between 285
fingering, orientation 283
fingering, right-hand, for fretted instruments 479
fingering, right-hand, placement 480
fingering, stem support 284
fingering, symbols, for wind instruments 496
fingering, thumb 156, 283, 930
fingeringOrientations 283
first-page-number 680, 967
\first-visible 876
\fixed 4, 948
flag, beam-like 118
flag, mensural 545
flag, modern straight 118
flag, old straight 118
flag, stacked 118
\flageolet 156, 930
'flageolet' harmonics 156, 930

flags, flat ... 117
\flagStyleDefault ... 118
\flagStyleStacked ... 118
flared hairpin ... 166
flat ... 8
\flat ... 855
flat flags ... 117
flat, double ... 8
flex program ... 941
\flexa ... 558, 563
flexa, in mensural ligatures ... 549
floor tom tom ... 931
follow voice ... 418
followVoice ... 418
font ... 940
font family ... 326
font family, setting ... 341
font feature, OpenType ... 340, 815
font format, bitmap ... 338
font format, CFF2 ... 337
font format, Font Variations (VF) ... 338
font format, OTC ... 338
font size (markup) ... 324
font size (notation) ... 278
font size (notation), scaling ... 278
font size (notation), standard ... 282
font size, setting ... 685
font, changing for entire document ... 341
font, Emmentaler ... 907
font, external files ... 338
font, Feta ... 907
font, finding ... 338
font, Parmesan ... 907
font, switching ... 324
font-interface ... 280
\font-select ... 811
font-size ... 279, 282
\fontCaps ... 811
FontConfig ... 337
fonts ... 341
fonts, color ... 338
fonts, for Emojis ... 338
\fontsize ... 324, 811
fontSize ... 280
foot mark ... 156, 930
footer ... 600
footer, page ... 607
footnote ... 613
\footnote ... 613, 876, 948
footnote, event-based ... 614
footnote, in music expressions ... 613
footnote, in stand-alone text ... 619
footnote, time-based ... 616
footnote-footer-padding ... 967
footnote-number-raise ... 967
footnote-numbering-function ... 967
footnote-padding ... 968
footnote-separator-markup ... 682, 968
Forbid_line_break_engraver ... 60
forced vertical direction of grobs ... 774
forget ... 37
'forget' accidental style ... 37
formatting, cue notes ... 269
formatting, in lyrics ... 346
formatting, rehearsal mark ... 143
formatting, text spanner ... 312
four-bar music ... 691
four-string-banjo ... 485
\fp ... 160
\fraction ... 877
fragment ... 269
fragment, music ... 650
fragment, quoting ... 266
framing text ... 332
free time ... 82
\freeBass ... 870
French clef ... 19, 923
\frenchChords ... 517
Frenched score ... 257
Frenched staff ... 254, 257
fret ... 434
fret diagram ... 455, 466
fret diagram, adding custom ... 469
fret diagram, adding fingering ... 478
fret diagram, automatic ... 477
fret diagram, changing orientation ... 464
fret diagram, custom ... 455
fret diagram, customized ... 463, 464
fret diagram, left-handed ... 459
fret diagram, mandolin ... 466
fret diagram, markup ... 457
fret diagram, transposing ... 468
fret diagram, ukulele ... 466
fret diagram, with chord names ... 467
\fret-diagram ... 457, 864
fret-diagram-interface ... 463
\fret-diagram-terse ... 459, 865
\fret-diagram-terse markup ... 459
\fret-diagram-verbose ... 461, 866
\fret-diagram-verbose markup ... 461
fretboard fret diagram, customized ... 471
fretboard, alternate table ... 475
fretboard, define predefined ... 472
FretBoards ... 466, 738
fretted instrument, chord shape ... 470
fretted instrument, dampened note ... 482
fretted instrument, harmonics ... 482
fretted instrument, indicating position and barring ... 481
fretted instrument, predefined string tunings ... 452
fretted instrument, right-hand fingering ... 479
\fromproperty ... 877
full score ... 392, 393
full-measure rest ... 71
funk shape note head ... 46
\funkHeads ... 46
\funkHeadsMinor ... 47

G

G clef ... 19
\general-align ... 328, 826
\germanChords ... 517
ghost note ... 296
ghost note, percussion ... 494
\glide ... 285
gliding fingering ... 285
glissando ... 177
\glissando ... 177
glissando, across repeats ... 181

glissando, and repeats 205
glissando, breakable 180
glissando, chords in tablatures 449
glissando, contemporary 179
glissando, timing marks 179
\glissandoMap 178
Global 736
global replacement, automatic, of strings 649
global variable 941
glyph 940
glyph set, accidental 922
glyph, alternate accidentals 39
glyph, bounding box 885
glyph, for figured bass alteration 530
\grace 148, 948
grace note 148, 498
grace note, and lyrics 383
grace note, and strict spacing 152
grace note, changing layout settings 150, 151
grace note, following 149
grace note, synchronization 150
grace note, tweaking 150, 151
grammar, for LilyPond 941
grand staff 244
GrandStaff 737
graphic notation 333
graphic object, drawing 332
graphic object, embedding 332
graphical object 940
graphical object interface 941
graphics, embedding 332, 334
Gregorian accidental 556
Gregorian articulation 557
Gregorian articulation, accentus 557, 930
Gregorian articulation, circulus 557, 930
Gregorian articulation, episem finis 557
Gregorian articulation, episem initium 557
Gregorian articulation, ictus 557, 930
Gregorian articulation, semicirculus 557, 930
Gregorian key signature 556
Gregorian music, modern transcription 405
Gregorian square neumes ligature 558
Gregorian transcription staff 243
GregorianTranscriptionLyrics 738
GregorianTranscriptionStaff 243, 738
GregorianTranscriptionVoice 738
grid line 302
grid lines, changing appearance 302
Grid_line_span_engraver 302
Grid_point_engraver 302
gridInterval 302
grob 757, 940
grob, aligning 791
grob, and whiteout 65, 785
grob, overwriting 65, 785
grob, property 762
grob, vertical position 774
grob, visibility 784
grob-interface 941
\grobdescriptions 949
grouping beats 112
grow-direction 119
guiro 931
guitar note head 43
guitar tablature 431
guitar, chord chart 100
guitar, strumming rhythm, showing 100, 101
gutter 678

H

hairpin 160
hairpin, al niente 165
hairpin, aligning ends to NoteColumn directions .. 165
hairpin, angled 791
hairpin, constante 166
hairpin, Ferneyhough 166
hairpin, flared 166
hairpin, minimum length 164
hairpin, moving ends 165
hairpin, stopping at bar lines 164
'half-open' articulation 156, 930
half-open high hat 156, 491, 931
\halfopen 156, 491, 930
\halign 327, 827
hammer on 450
hand clap 931
Harmonia Sacra note head 46
\harmonic 44, 429, 438
harmonic indication, in tablature 438
harmonic note head 43
\harmonicByFret 438, 949
\harmonicByRatio 438, 949
\harmonicNote 949
harmonics, artificial 429
harmonics, flageolet 156, 930
harmonics, in tablatures 445
harmonics, natural 428
harmonics, on fretted instrument 482
\harmonicsOff 428
\harmonicsOn 428, 949
harp 425
harp, pedal 425
harp, pedal diagram 425
\harp-pedal 426, 867
'Haydn turn' ornament 156, 928
\haydnturn 156, 928
\hbracket 332, 845
\hcenter-in 828
header 600
\header 597
header, page 607
\heel 156, 423, 930
'heel' pedal mark 156, 423, 930
\heelcircle 156, 423, 930
Hel-arabic note name 576
help, balloon 300
Henze fermata 156, 929
\henzelongfermata 156, 929
\henzeshortfermata 156, 929
hidden note 289
\hide 785, 949
\hideKeySignature 498
\hideNotes 289
\hideSplitTiedTabNotes 437
\hideStaffSwitch 418
hiding accidentals, on tied notes at start of system . 9
hiding ancient staves 257
hiding dynamics extender line 167

hiding rhythmic staves 257
hiding staves 257
hiding Vaticana staves 257
high bongo 931
high conga 931
high hat 931
high hat, half-open 156, 491, 931
high hat, open 156, 931
high hat, pedal 156, 491, 931
high hat, pedal splash 491
high timbale 931
high tom tom 931
highest pitch 10
highlight, staff 292
horizontal alignment, lyrics 364
horizontal alignment, melisma 365
horizontal beams 444
horizontal bracket 304
horizontal extent, of grobs 791
horizontal movement, in markup 328
horizontal ornament 157
horizontal shift of notes 231
horizontal spacing 713
horizontal spacing, between columns 720
horizontal spacing, between non-musical items ... 718
horizontal text alignment 327
horizontal-shift 678, 968
Horizontal_bracket_engraver 304
HorizontalBracketText 305
horizontally centering text 823
\hspace 328, 828
Hufnagel 538, 539
\huge 281, 327, 812
hymn 400, 405
hymn, and partial measures 410
hyperlink, as QR code 881
hyphen 357

I

\ictus 557, 930
'ictus' Gregorian articulation 557, 930
\if 611, 864
\iij 559
\IIJ 559
\ij 559
\IJ 559
image 845
\image 845
image, embedding 334
immutable object 941
immutable property 941
implicit context 756
importing stencil, into text 883
improvisation 50
improvisation, Arabic 580
\improvisationOff 50, 100
\improvisationOn 50, 100
\in 775, 968
in-note 622
in-note-direction 622
in-note-padding 622, 968
in-note-system-padding 622, 968
inch (in) 775
\incipit 567, 949
incipit, adding 567
incipit-width 968
\inclinatum 558, 563
\include 598, 628
include settings 641
including files 628
indent 263, 678, 722, 968
indicating 'No Chord', in ChordNames 514
indicating position and barring for fretted instrument 481
individual parts 262
\inherit-acceptability 752, 949
\initialContextFrom 949
inner-margin 678, 968
input-language 941, 950
inserting music, into text 860
inserting PostScript directly, into text 848
inserting URL link, into text 851
\inStaffSegno 131, 132, 207, 949
instrument 603
instrument equalization, in MIDI, replacing default 658
instrument name 262, 660
instrument name, adding to other contexts 264
instrument name, centering 263
instrument name, changing 264
instrument name, complex 263
instrument name, short 262
instrument, transposing 14
\instrumentSwitch 950
interface 941
interface, layout 757
interleaved music 238
Internals Reference 735
inversion 16
\inversion 16, 950
inversion, modal 18
\invertChords 513, 950
invisible note 289
invisible rest 69
invisible stem 297
Ionian 24
\ionian 24
irregular measure 79, 92
is-last-bookpart 968
isolated duration 53
isolated percent repeat 211
isolated pitch 53
\italianChords 517
\italic 324, 812
item, musical 717
item, non-musical 717

J

jazz chord 516
\jump 208, 950
\justified-lines 322, 886
\justify 331, 829
\justify-field 829
\justify-line 830
\justify-string 831
justifying lines of text 886

justifying text 331, 829

K

\keep-with-tag 639, 878
keepAliveInterfaces 258
keeping tagged music 631
\keepWithTag 631, 950
\key 24, 47, 950
key cancellation, before bar line 719
key signature 8, 24
key signature, Gregorian 556
key signature, mensural 546
key signature, non-traditional 26
key signature, preventing natural signs 25
key signature, visibility following explicit change . 788
keyboard instrument staff 414
keyboard music, centering dynamics 414
keyed instrument staff 414
Kievan accidental 565
Kievan clef 543, 924
Kievan ligature 566
Kievan_ligature_engraver 566
\kievanOff 565
\kievanOn 565
KievanStaff 564, 738
KievanVoice 564, 738
\killCues 273, 950
Kirchenpause 276
knee gap, with beams, changing 104
koron 583

L

\label 625, 950
label-alist-table 968
label-page-table 968
laissez vibrer 64
\laissezVibrer 64
landscape 968
\language 950
language, note names in other 10
language, pitch names in other 10
\languageRestore 950
\languageSaveAndChange 950
\large 281, 327, 812
\larger 324, 327, 812
last-bottom-spacing 676, 709, 968
layer 785
\layout 596, 683, 736, 746
layout object 940
\layout, and \book 595
\layout, and \bookpart 595
\layout, and paper variables 683
layout, file 685
layout, interface 757
layout, major 7 chord 520
layout, vertical, debugging 729
layout-set-staff-size 685
lead sheet 515
ledger line 250
ledger line, internal 250
ledger line, modifying 250
\left-align 327, 831
left-aligning text 831
\left-brace 878
\left-column 832
left-handed fret diagram 459
left-margin 677, 968
LEFT 793, 794
length of multi-measure rest 73
length of note 52
lexer, for LilyPond 941
\lheel 423
\lheeltoe 423
ligature 540
\ligature 950
ligature, in text 824
ligature, Kievan 566
ligature, mensural 549
ligature, square neumes 558
Ligature_bracket_engraver 549, 566
LilyPond grammar 941
LilyPond lexer 941
\line 321, 322, 832
line break 121
line break, beams 103
line break, cadenzas 94
line break, managing with extra voice 692
line break, unmetered music 94
line comment 592, 598
line, barré 481
line, between fingerings 285
line, cross-staff 418
line, grid 302
line, indicating duration 535
line, staff-change 418
line, staff-change follower 418
line, vertical, between staves 302
line-break-system-details 704
line-thickness 969
line-width 331, 676, 722, 969
\linea 558, 563
\lineprall 156, 928
list of colors 902
list of keys, in woodwind diagrams 506
list of markups 321
list of woodwind diagrams 503
Locrian 24
\locrian 24
long fermata 156, 929
\longa 52, 67
longa rest 67
\longfermata 156, 929
\lookup 878
low bongo 931
low conga 931
low timbale 931
low tom tom 931
\lower 328, 832
'lower mordent' ornament 156, 928
lowering text 832
lowest pitch 10
\ltoe 423
\ltoeheel 423
lute tablature 485
lute tuning 486
ly:minimal-breaking 694
ly:one-line-auto-height-breaking 695

ly:one-line-breaking ... 695
ly:one-page-breaking ... 694
ly:optimal-breaking ... 694
ly:page-turn-breaking ... 695
Lydian ... 24
\lydian ... 24
\lyricmode ... 346, 347, 589
\lyrics ... 589
Lyrics ... 738
lyrics, aligning to a melody ... 347
lyrics, aligning with sporadic melody ... 743
lyrics, and markup ... 347
lyrics, and melodies ... 349
lyrics, and tied notes ... 372
lyrics, assigned to one voice ... 222
lyrics, avoid bar lines ... 366
lyrics, divided ... 374
lyrics, entering ... 346
lyrics, formatting ... 346
lyrics, horizontal alignment ... 364
lyrics, increasing space between ... 363
lyrics, keeping inside margin ... 311, 366
lyrics, on grace notes ... 383
lyrics, positioning ... 256, 360
lyrics, punctuation ... 346
lyrics, repeating ... 366
lyrics, repeats with alternative endings ... 370
lyrics, separate, on single line ... 362
lyrics, shared among voices ... 376
lyrics, skip ... 70
lyrics, skipping notes ... 371
lyrics, using variables ... 358
lyrics, with beam ... 106
\lyricsto ... 347, 349

M

m (in chord) ... 509
magnification->font-size ... 280, 685
\magnify ... 324, 813
magnifying text ... 813
\magnifyMusic ... 278, 950
\magnifyStaff ... 685, 951
magstep ... 280, 685, 776
maj ... 509
major ... 24
\major ... 24
major 7 chord, layout ... 520
major seven symbol ... 519
majorSevenSymbol ... 517, 520
makam ... 575, 581
makam, example ... 582
makamlar ... 575, 581
make-dynamic-script ... 169
make-footer ... 969
make-header ... 969
make-relative ... 646
\makeClusters ... 221, 951
\makeDefaultStringTuning ... 951
manual bar line ... 121
manual beam ... 102, 115
manual beam, direction shorthand for ... 115
manual beam, grace notes ... 115
manual engraving, of ties ... 66
manual line break ... 689
manual measure line ... 121
manual rehearsal mark ... 143
manual repeat mark ... 205
manual staff change ... 414
maqam ... 575
maracas ... 931
\marcato ... 156, 928
'marcato' articulation ... 156, 928
margin, text running over ... 311
\mark ... 142, 314, 951
mark, on every staff ... 144
mark, phrasing ... 174
mark, rehearsal ... 142
mark, rehearsal, below staff ... 90
mark, rehearsal, format ... 143
mark, rehearsal, manual ... 143
mark, rehearsal, style ... 143
mark, text ... 314
Mark_engraver ... 144
\markalphabet ... 878
\markLengthOff ... 89, 315
\markLengthOn ... 89, 315
\markletter ... 879
markup ... 320
\markup ... 308, 314, 318, 319, 320, 590
markup list ... 321
markup mode, quoted text ... 320
markup mode, special characters ... 320
markup object ... 308
markup text, aligning ... 327
markup text, alignment commands ... 332
markup text, decorating ... 332
markup text, framing ... 332
markup text, justified ... 331
markup text, line width ... 331
markup text, multi-page ... 322
markup text, padding ... 333
markup text, size ... 324
markup text, word-wrapped ... 331
markup, aligning ... 327
markup, centering on page ... 331
markup, conditional ... 611
markup, expression ... 320
markup, horizontal movement ... 328
markup, image ... 845
markup, in lyrics ... 347
markup, multi-line ... 330
markup, multi-measure rest ... 75
markup, multi-page ... 322
markup, music notation inside ... 335
markup, on multi-measure rest ... 72
markup, rhythm ... 859
markup, score inside ... 336
markup, syntax ... 320
markup, text ... 320
markup, text, inside slurs ... 173
markup, two-column ... 319
markup, vertical movement ... 328, 329
markup-markup-spacing ... 676, 969
markup-system-spacing ... 675, 969
\markuplist ... 319, 322
\markupMap ... 951
matter, optional ... 295
matter, prefatory ... 717, 718

max-systems-per-page 679, 969
\maxima 52, 67
maxima rest 67
maximum pitch 10
measure 78
\measure 79, 951
measure counter 144
measure line 121, 132
measure line, invisible 121
measure line, manual 121
measure number 78, 133
measure number, and repeats 205
measure number, check 141
measure number, style 136
measure position 78, 92
measure repeat 209
measure, check 141
measure, grouping 112
measure, irregular 79, 92
measure, partial 92
measure, partial, in hymns 410
measure, pickup 92
measure, subgrouping 112
Measure_grouping_engraver 112
measureBarType 130
measureLength 78, 106
measurePosition 78
\measureRemainder 951
measuring units 775
Medicaea, Editio 538, 539
medium interval 575
melisma 354, 357
\melisma 354
melisma, horizontal alignment 365
melisma, with beams 103
\melismaEnd 354
melismata 354
melody rhythm, showing 100
mensural 538, 539
mensural clef 543, 924
mensural flag 545
mensural ligature 549
mensural ligature, flexa 549
mensural music, transcription 247
mensural notation, signum congruentiae 930
Mensural_ligature_engraver 549
MensuralStaff 243, 542, 737
MensuralVoice 542, 738
mensuration sign 544
Mensurstriche 568
Mensurstriche, layout 247
\mergeDifferentlyDottedOff 227
\mergeDifferentlyDottedOn 227
\mergeDifferentlyHeadedOff 227
\mergeDifferentlyHeadedOn 227
merging notes 227
merging rests 232
merging text 824, 833
meter 80
meter 603
meter, additive 81
meter, alternating 81
meter, complex 81
meter, open 82
meter, polymetric 95
meter, strictly alternating 131
meter, style 81
metronome mark 87
metronome mark, below staff 90
metronome mark, custom markup 91
mezzo-soprano clef 19, 923
\mf 160
microtone 11
microtone, in tablature 454
mid tom tom 931
\midi 596, 736
MIDI 30, 654
MIDI, balance 664
MIDI, block 655
MIDI, channels 661
MIDI, chorus level 664
MIDI, context definitions 659
MIDI, custom dynamics 656
MIDI, dynamics 656
MIDI, effects 664
MIDI, equalization 657
MIDI, expression 664
MIDI, instrument 660
MIDI, metadata 612
MIDI, one channel per voice 663
MIDI, pan position 664
MIDI, replacing default instrument equalization 658
MIDI, reverb 664
MIDI, stereo balance 664
MIDI, supported notation 654
MIDI, tracks 661
MIDI, transposition 30
MIDI, unsupported notation 655
MIDI, using repeats 661
MIDI, volume 657
midiBalance 664
midiChannelMapping 661
midiChorusLevel 664
midiDrumPitches 492
midiExpression 664
midiInstrument 660
midiPanPosition 664
midiReverbLevel 664
millimeter (mm) 775
min-systems-per-page 679, 969
minimum length, hairpin 164
minimum pitch 10
minimum-Y-extent 697
minimumFret 434, 478
minor 24
\minor 24
minorChordModifier 519
mirroring markup 850
Mixolydian 24
\mixolydian 24
\mm 775, 969
modal inversion 18
modal transformation 17
modal transposition 17
\modalInversion 18, 951
\modalTranspose 17, 951
mode 24
modern 34
modern accidental 34
'modern' accidental style 33, 34

modern transcription of Gregorian music 405
modern-cautionary 34
'modern-cautionary' accidental style 33, 34
modern-voice 34
modern-voice-cautionary 34
'modern-voice-cautionary' accidental style 34
moderntab clef 454
modifier, in chord 509
modifying contexts 760
modifying properties 760
\mordent 156, 928
'mordent' ornament 156, 928
'mordent, lower' ornament 156, 928
'mordent, upper' ornament 156, 928
movement in markup, horizontal 328
movement in markup, vertical 328, 329
movement, multiple 593
\mp 160
multi-line comment 592, 598
multi-line markup 330
multi-line text 330
multi-measure rest 71
multi-measure rest, and fingerings 75
multi-measure rest, attaching fermata 72
multi-measure rest, attaching text 72
multi-measure rest, contracting multiple 274, 275
multi-measure rest, expanding multiple 274, 275
multi-measure rest, length 73
multi-measure rest, markup 75
multi-measure rest, numbering 276
multi-measure rest, positioning 74
multi-measure rest, script 72
multi-measure rest, style 276
multi-measure rest, with markup 72
multi-measure rest, within text, by duration 858
multi-measure rest, within text, by number of measures 856
\multi-measure-rest-by-number 856
multi-note acciaccatura 153
multi-page markup 322
multi-voice accidental 34
MultiMeasureRestScript 73
MultiMeasureRestText 73
multiple dynamic marks, on one note 161
multiple phrasing slurs 175
multiple slurs 171
multiple voices 227
music expression, copying 644
music expression, sharing 644
music fragment 650
music, beginners' 45
music, inside markup 335
music, unmetered 78
musica ficta 547
musical cue 395
musical item 717
musical theater 392
\musicglyph 856
\musicLength 952
\musicMap 952
musicological analysis 304
musicQuotes 941
mutable object 941
mutable property 941
mute bongo 931
mute conga 931
mute timbale 931
muted note, percussion 494

N

\n 160
N.C. symbol 514
N.C. symbol, customizing 516
\name 751, 752
name, character 393
name, of singer 382
\natural 856
natural harmonics 428
natural pitch 8
natural sign 8
natural sign, extra, preventing 9
natural sign, preventing in key signatures 25
neo-modern 35
'neo-modern' accidental style 35
neo-modern-cautionary 36
'neo-modern-cautionary' accidental style 36
neo-modern-voice 36
'neo-modern-voice' accidental style 36
neo-modern-voice-cautionary 36
'neo-modern-voice-cautionary' accidental style 36
neomensural 539
nesting repeat 205
nesting, staff bracket 248
nesting, staves 248
\new 739
new context 739
new dynamic mark 168
new spacing section 714
new staff 242
\newSpacingSection 714
\newTocItem 952
niente, al, hairpin 165
'No Chord' symbol 514
'No Chord' symbol, customizing 516
no-reset 37
'no-reset' accidental style 37
\noBeam 115
\noBeamBreak 115
\noBreak 691
noChordSymbol 516
non-arpeggiato 184
non-ASCII character 647
non-default tuplet numbers 57
non-empty text 310
non-musical item 717
non-musical item, horizontal spacing 718
non-musical symbol 333
\nonArpeggiato 184, 391, 420
NonMusicalPaperColumn 704, 717
nonstaff-nonstaff-spacing 697
nonstaff-relatedstaff-spacing 697
nonstaff-unrelatedstaff-spacing 697
\noPageBreak 693, 952
\noPageTurn 696, 952
\normal-size-sub 813
\normal-size-super 325, 813
\normal-text 813
\normal-weight 814

\normalsize 279, 281, 327, 814
\noStanza 378, 952
notation, explaining 300
notation, font size 278
notation, graphic 333
notation, inside markup 335
\note 857
note cluster 221
note collision 227
note continuation 535
note duration 52
note duration, default 53
note grouping bracket 304
note head 278
note head, Aiken 46
note head, Aiken, thin variant 48
note head, ancient 544, 565
note head, Christian Harmony 46
note head, cross 43
note head, diamond 43
note head, diamond-shaped 428
note head, easy notation 45
note head, easy play 45
note head, funk 46
note head, guitar 43
note head, Harmonia Sacra 46
note head, harmonic 43
note head, improvisation 50
note head, parlato 43
note head, practice 45
note head, Sacred Harp 46
note head, shape 46
note head, shape, merging 49
note head, slashed 50
note head, Southern Harmony 46
note head, special 43
note head, style 43, 922
note head, Walker 46
note length 52
note name, Arabic 576
note name, default 8
note name, Dutch 8
note name, Hel-arabic 576
note name, other languages 10
note name, printing 298
note pitch, default 53
note, colored 289
note, colored, in chords 291
note, cross-staff 414, 420
note, dotted 53
note, dotted, moving horizontally 231
note, double-dotted 53
note, ghost 296
note, hidden 289
note, horizontal shift 231
note, invisible 289
note, parenthesized 296
note, smaller 269
note, spacing horizontally 714
note, splitting 98
note, transparent 289
note, transposition 13
note, within text, by duration 857
note, within text, by log and dot-count 857
\note-by-number 857
note-event 268
Note_heads_engraver 98
Note_name_engraver 298
NoteColumn 717
\notemode 590
noteNameFunction 298
NoteNames 298, 739
noteNameSeparator 298
\null 879
NullVoice 376, 739
\number 814
number, bar 133
number, in easy notation 45
number, measure 133
number-footnote-table 969
\numericTimeSignature 81

O

object, colored 289, 785
object, markup 308
object, overwriting 785
object, rotating 791
object, visibility 784
objects, aligning 791
octavation 26
octave changing mark 3
octave check 12
octave correction 12
octave entry, absolute 3
octave entry, relative 4
octave specification, absolute 3
octave specification, relative 4
octave transposition 20
octave transposition, optional 20
\octaveCheck 12, 952
oddFooterMarkup 969
oddHeaderMarkup 970
\offset 766, 952
offsetting 766
\omit 785, 952
\on-the-fly 879
\once 763, 768, 777, 952
OneStaff 245, 737
\oneVoice 222
\open 156, 428, 491, 496, 930
'open' articulation 156, 496, 930
open bongo 931
open conga 931
open high hat 156, 931
open meter 82
open string indication 428
open timbale 931
OpenType collections (.otc) 338
OpenType font (.otf) 337
OpenType, font feature 340, 815
opera 392
operation, inversion 16
operation, modal 17
operation, modal inversion 18
operation, retrograde 17
operation, transposition 17
optional material 295
optional octave transposition 20

optional passage ... 295
opus ... 603
oratorio ... 389
orchestra, notation for ... 393
orchestral music ... 393
orchestral strings ... 427
ordering, vertical, of scripts ... 158
organ ... 423
organ pedal mark ... 156, 423, 930
orientation, of fingerings ... 283
orientation, of fret diagram, changing ... 464
orientation, of string numbers ... 283
orientation, of stroke finger ... 283
\oriscus ... 558, 563
ornament ... 148
ornament, down prall triller ... 156, 928
ornament, Haydn turn ... 156, 928
ornament, horizontally ... 157
ornament, lower mordent ... 156, 928
ornament, mordent ... 156, 928
ornament, mordent, lower ... 156, 928
ornament, mordent, upper ... 156, 928
ornament, prall triller ... 156, 928
ornament, prall triller, down ... 156, 928
ornament, prall triller, long ... 156, 928
ornament, prall triller, up ... 156, 928
ornament, reverse turn ... 156, 928
ornament, signum congruentiae ... 156
ornament, slash turn ... 156, 928
ornament, trill ... 156, 928
ornament, turn ... 156, 928
ornament, turn, delayed ... 159
ornament, up prall triller ... 156, 928
ornament, upper mordent ... 156, 928
orphan-penalty ... 970
ossia ... 254, 261
ossia, positioning ... 256
ottava ... 26
\ottava ... 26, 953
ottava spanner, modifying slope ... 29
ottava text ... 28
ottava, for single voice ... 28
ottavation ... 28
ottavation-numbers ... 27
ottavation-ordinals ... 27
ottavation-simple-ordinals ... 27
ottavationMarkups ... 27
Ottoman music ... 581
Ottoman, classical music ... 575
outer-margin ... 678, 970
output definition ... 736
output-count ... 941
output-def ... 941
output-filename ... 595, 970
output-scale ... 970
output-suffix ... 595, 941, 970
outside-staff-horizontal-padding ... 711
outside-staff-padding ... 711
outside-staff-priority ... 710, 937
\oval ... 845
\overlay ... 833
overlaying contexts ... 245
\override ... 762, 764, 879
override, reverting ... 762
\override-lines ... 887
\overrideProperty ... 764, 953
\overrideTimeSignatureSettings ... 84, 953
overriding property, within text markup ... 879
\overtie ... 816
overwriting grob ... 65
overwriting object ... 785

P

\p ... 160
\pad-around ... 333, 833
\pad-markup ... 333, 833
\pad-to-box ... 333, 834
\pad-x ... 333, 834
\pad-x-left ... 333, 834
\pad-x-right ... 333, 835
padding ... 759
padding around text ... 333
padding text ... 833
padding text horizontally ... 834, 835
page break ... 722
page break, cadenzas ... 94
page break, managing with extra voice ... 692
page break, manual ... 693
page break, unmetered music ... 94
page break, within markup ... 322
page footer ... 607
page header ... 607
page layout ... 722
page number, auto-numbering ... 680
page number, in roman numerals ... 680
page number, independent for introduction ... 681
page number, per book part ... 681
page number, referencing ... 623
page number, specify first ... 680
page number, suppress ... 680
page size ... 672
page, orientation ... 673
page-breaking ... 679, 970
page-breaking-system-system-spacing ... 679, 970
page-count ... 679, 970
\page-link ... 879
page-number-type ... 680, 970
page-post-process ... 970
\page-ref ... 625, 880
page-spacing-weight ... 679, 970
\pageBreak ... 693, 953
\pageTurn ... 696, 953
pageTurnMinimumRepeatLength ... 696
pageTurnMinimumRestLength ... 695
\palmMute ... 953
\palmMuteOn ... 953
pan position, in MIDI ... 664
Pango ... 337
\paper ... 596, 672
paper column ... 717
paper size ... 672
paper size, landscape ... 673
paper size, orientation ... 673
paper variable, predefined ... 672
paper variables, in \layout block ... 683
paper-height ... 674, 971
paper-width ... 676, 971
papersizename ... 971

parallel music ... 238
\parallelMusic ... 238, 953
parent-alignment-X ... 795
parent-alignment-Y ... 796
parenthesis ... 296
\parenthesize ... 296, 846, 954
parenthesized accidental ... 8
parlato ... 399
parlato note head ... 43
Parmesan font ... 907
parser ... 941
parser variable ... 941
part combiner ... 233
part combiner, changing text ... 237
part song ... 389
part, desk ... 262
part, individual ... 262
part, section ... 262
\partCombine ... 233, 376, 954
\partCombine, and lyrics ... 236, 376
\partCombineApart ... 235
\partCombineAutomatic ... 235
\partCombineChords ... 235
\partCombineDown ... 954
\partCombineForce ... 954
partCombineListener ... 941
\partCombineSoloI ... 235
\partCombineSoloII ... 235
\partCombineUnisono ... 235
\partCombineUp ... 954
\partial ... 78, 92, 191, 954
partial measure ... 92
partial measure, in hymns ... 410
\path ... 846
path, drawing ... 846
\pattern ... 880
pause mark ... 76, 176
PDF, embedding files ... 652
PDF, metadata ... 612
pedal diagram, harp ... 425
pedal high hat ... 156, 491, 931
pedal indication, bracket ... 422
pedal indication, mixed ... 422
pedal indication, style ... 422
pedal indication, text ... 422
pedal mark, heel ... 156, 423, 930
pedal mark, organ ... 156, 423, 930
pedal mark, toe ... 156, 423, 930
pedal splash high hat ... 491
pedal sustain, style ... 422
pedal, harp ... 425
pedal, piano ... 421
pedal, sostenuto ... 421
pedal, sustain ... 421
pedalSustainStyle ... 422
percent repeat ... 209
percent repeat, count visibility ... 210
percent repeat, counter ... 210
percent repeat, isolated ... 211
percussion ... 487, 489
percussion clef ... 487, 924
percussion staff ... 242
percussion, custom ... 491
percussion, dead note ... 494
percussion, false note ... 494
percussion, ghost note ... 494
percussion, muted note ... 494
percussion, silenced note ... 494
Performer_group ... 753
Persian accidentals (sori, koron) ... 583
Persian classical music ... 583
Persian makam ... 575
\pes ... 558, 563
Petrucci ... 538, 539
Petrucci clef ... 543, 924
PetrucciStaff ... 549, 738
PetrucciVoice ... 549, 738
phrase bar line ... 131, 132
phrase bar line, in Gregorian chant ... 556
phrase bar line, in hymns ... 409
phrase bar line, in Kievan chant ... 566
phrasing bracket ... 304
phrasing mark ... 174
phrasing slur ... 171, 174
phrasing slur, dashed ... 175
phrasing slur, defining dash patterns ... 175
phrasing slur, dotted ... 175
phrasing slur, half solid and half dashed ... 175
phrasing slur, multiple ... 175
phrasing slur, simultaneous ... 175
phrasing, in lyrics ... 354
\phrasingSlurDashed ... 175
\phrasingSlurDashPattern ... 175, 954
\phrasingSlurDotted ... 175
\phrasingSlurDown ... 174
\phrasingSlurHalfDashed ... 175
\phrasingSlurHalfSolid ... 175
\phrasingSlurNeutral ... 174
\phrasingSlurSolid ... 175
\phrasingSlurUp ... 174
Phrygian ... 24
\phrygian ... 24
piano ... 35
'piano' accidental style ... 35
piano music, centering dynamics ... 414
piano pedal ... 421
piano staff ... 245, 414
piano-cautionary ... 35
'piano-cautionary' accidental style ... 35
PianoStaff ... 414, 416, 737
pickup measure ... 92
pickup, in a repeat ... 191
piece ... 603
pitch ... 3
pitch name ... 3
pitch name, other languages ... 10
pitch range ... 40
pitch, approximate ... 10
pitch, default ... 53
pitch, extreme ... 10
pitch, isolated ... 53
pitch, transposition ... 13
pitch, unspecified ... 10
Pitch_squash_engraver ... 100
pitched trill ... 187
pitched trill, with accidental ... 187
\pitchedTrill ... 187, 955
pitches, 'smart' transposition ... 15
pitchnames ... 941, 950
pizzicato, Bartók ... 429

pizzicato, snap 429
place-fret 866
placeholder event 217
placement, lyrics 360
placement, right-hand fingering 480
placing horizontal brackets, around text 845
placing parentheses, around text 846
placing vertical brackets, around text 840
PNG image 845
poet 603
point (pt) 775
point, big (bp) 775
point, of alignment 792
point, of reference 791
\pointAndClickOff 955
\pointAndClickOn 955
\pointAndClickTypes 955
\polygon 847
\polymetric 95, 955
polymetric meter, with beams 95
polymetric signatures 95
polyphonic music 227
polyphony, additional voices 230
polyphony, in tablatures 444
polyphony, shared lyrics 376
polyphony, single-staff 222
\popContextProperty 761, 955
\portato 156, 928
'portato' articulation 156, 928
position, figured bass alteration 530
position, in measure 78, 92
position, lyrics 256
position, multi-measure rest 74
position, ossia 256
position, vertical, of grobs 774
positioning, exact, of staves 704
post-event 928
postscript 334
\postscript 334, 848
power chord 483
\pp 160
\ppp 160
\pppp 160
\ppppp 160
practice note head 45
\prall 156, 928
\pralldown 156, 928
\prallmordent 156, 928
\prallprall 156, 928
'pralltriller' ornament 156, 928
'pralltriller, down' ornament 156, 928
'pralltriller, long' ornament 156, 928
'pralltriller, up' ornament 156, 928
\prallup 156, 928
\preBend 440, 955
\preBendHold 440, 955
predefined paper variables 672
predefined string tuning, for fretted instruments . 452
predefinedDiagramTable 475
\predefinedFretboardsOff 477
\predefinedFretboardsOn 477
prefatory matter 717, 718
\premeasure 79, 92, 955
prima volta 191
print-all-headers 681, 971
print-first-page-number 680, 971
print-page-number 680, 971
printAccidentalNames 298
printing chord name 513
printing order 785
printing reserved character 320
printing special character 320
printNotesLanguage 298
printOctaveNames 298
prob 942
properties 766
property object 942
property, grob 762
property, modifying 760
property-defaults 971
\property-recursive 880
\propertyOverride 956
\propertyRevert 956
\propertySet 956
\propertyTweak 956
\propertyUnset 956
psalm 400, 405
\pt 775, 971
pull off 450
punctuation, in lyrics 346
\push-to-tag 639, 880
\pushContextProperty 761, 956
\pushToTag 637, 956
\pushToTagMarkup 639, 956
\put-adjacent 835
putting space around text 833

Q

q (chord repetition) 218, 435
\qr-code 881
QR code 881
quarter tone 8
quarter tone accidental 10
quarter tone, in tablature 454
\quilisma 558, 563
quote, in lyrics 346, 353
quote, voices 266
quoted text 310
quoted text, in markup mode 320
quotedCueEventTypes 268
quotedEventTypes 268
\quoteDuring 266, 269, 956
\quoteDuring, and tags 632

R

r (rest) 67
R (full-measure rest) 71
ragged-bottom 674, 971
ragged-last 677, 722, 971
ragged-last-bottom 674, 971
ragged-right 677, 722, 971
\raise 328, 835
\raiseNote 513, 957
raising text 835
range of pitches 40
Ratisbona, Editio 539
\reduceChords 101, 957

reference point ... 791
referencing context ... 739
referencing page label, in text ... 885
referencing page number, in text ... 879, 880
register symbol, accordion ... 424
regular line break ... 691
rehearsal mark ... 142
rehearsal mark, below staff ... 90
rehearsal mark, format ... 143
rehearsal mark, manual ... 143
rehearsal mark, style ... 143
relative ... 4
\relative ... 4, 6, 16, 417, 646, 957
relative music, and \autoChange ... 417
relative octave entry ... 4
relative octave entry, and chords ... 7
relative octave entry, and transposition ... 6
relative octave specification ... 4
relative pitch, chords ... 217
religious music ... 400
reminder accidental ... 8
removal, in chord ... 511
\remove ... 745
remove-empty ... 258
remove-first ... 258
remove-grace-property ... 151
remove-layer ... 260
\remove-with-tag ... 639, 881
\RemoveAllEmptyStaves ... 258, 964
\RemoveEmptyStaves ... 258, 964
\removeFromTagGroup ... 636, 957
\removeWithTag ... 631, 957
removing bar numbers ... 139
removing cue notes ... 273
removing tagged music ... 631
Renaissance music ... 247
repeat ... 124
\repeat ... 189, 190, 191
repeat bar ... 121
repeat bar line ... 132
repeat bar line, underlying ... 131
repeat number, changing ... 205
\repeat percent ... 209
\repeat segno ... 131, 132, 196
\repeat tremolo ... 212
\repeat unfold ... 189
\repeat volta ... 131, 132, 190, 191
repeat volta, changing ... 205
repeat, al fine ... 196
repeat, alla coda ... 197
repeat, alternative bar numbers ... 203
repeat, ambiguous ... 205
repeat, and glissandi ... 181, 205
repeat, and lyrics ... 366
repeat, and measure number ... 205
repeat, and slur ... 205
repeat, bar line at start of piece ... 190
repeat, bar numbers with letters ... 203
repeat, beat ... 209
repeat, D.C. ... 197
repeat, D.C., manual ... 208
repeat, D.S. ... 198
repeat, D.S., forced at start ... 199
repeat, D.S., manual ... 208
repeat, da capo ... 197
repeat, da capo, manual ... 208
repeat, dal segno ... 198
repeat, dal segno, forced at start ... 199
repeat, dal segno, manual ... 208
repeat, double, style for volta ... 133
repeat, e poi la coda ... 197
repeat, end ... 205
repeat, manual ... 205
repeat, measure ... 209
repeat, nested ... 205
repeat, percent ... 209
repeat, percent, count visibility ... 210
repeat, percent, counter ... 210
repeat, percent, isolated ... 211
repeat, short ... 209
repeat, simple ... 190
repeat, slash ... 209
repeat, start ... 205
repeat, timing information ... 205
repeat, tremolo ... 212
repeat, unfolded ... 189
repeat, with alternative endings ... 191
repeat, with anacrusis ... 191
repeat, with bar checks ... 191
repeat, with pickup ... 191
repeat, with segno ... 199
repeat, with ties ... 64
repeat, with upbeat ... 191
repeat, written-out ... 189
repeatCommands ... 205
repeatCountVisibility ... 210
repeated chords, suppressing ... 474, 515
repeating lyrics, with alternative endings ... 370
repeating tie ... 64
repeats, in MIDI ... 661
\repeatTie ... 64, 372
repetition, using q ... 218, 435
\replace ... 817
\replace-with-tag ... 639, 882
\replaceWithTag ... 638, 957
\replaceWithTagMarkup ... 639, 957
replacing strings, automatically ... 649
reserved character, printing ... 320
reset-footnotes-on-new-page ... 681, 971
\resetRelativeOctave ... 7, 957
\resetTagGroup ... 636, 957
\resetTagGroups ... 635, 957
resizing of staves ... 254
\responsum ... 559, 957
rest ... 67
\rest ... 68, 858
rest, ancient ... 546
rest, church ... 276
rest, collisions ... 75
rest, condensing ordinary ... 75
rest, entering durations ... 67
rest, full-measure ... 71
rest, invisible ... 69
rest, merging ... 232
rest, multi-measure ... 68, 71
rest, specifying vertical position ... 68
rest, splitting ... 98
rest, style ... 68
rest, whole, for a full measure ... 71
rest, whole-measure ... 68

rest, within text, by duration 858
rest, within text, by log and dot-count 859
\rest-by-number 859
rest-event 268
restNumberThreshold 276
restoring default properties for time signatures 84
restrainOpenStrings 434
\retrograde 17, 957
retrograde transformation 17
reverb, in MIDI 664
'reverse turn' ornament 156, 928
\reverseturn 156, 928
\revert 762
reverting override 762
\revertTimeSignatureSettings 84, 957
\rfz 160
rgb-color 291
RGB color 291
\rheel 423
\rheeltoe 423
\rhythm 737, 738, 739, 859
rhythm, in text 859
rhythm, showing melody 100
rhythmic staff 243
RhythmicStaff 243, 737
ride bell 931
ride cymbal 931
\right-align 327, 835
right-aligning text 835
\right-brace 882
\right-column 836
right-hand fingering, for fretted instruments 479
right-hand fingering, placement 480
right-margin 677, 971
RIGHT 793, 794
\rightHandFinger 479, 957
\rigid 150, 957
rolled chord 184
\romanStringNumbers 428, 432
root of chord 509
\rotate 836
rotating object 791
rotating text 836
\rounded-box 332, 849
\rtoe 423
\rtoeheel 423

S

s (invisible rest) 69
Sacred Harp note head 46
\sacredHarpHeads 46
\sacredHarpHeadsMinor 47
\sans 817
SATB 389
scalable vector graphics output 651
\scale 850
\scaleDurations 62, 95, 957
scaling duration 61
scaling markup 850
scaling text 837
Scheme object 942
Scheme variable 941
Score 736
\score 592, 597, 860
Score context, replacing 754
score, inside markup 336
score, vocal, adding cues 397
\score-lines 887
score-markup-spacing 676, 972
score-system-spacing 676, 972
score-title 972
score-title-properties 972
Score_engraver 755
Score_performer 755
scoreTitleMarkup 608, 972
Scottish highland bagpipe 498
script 928
script, on multi-measure rest 72
script, vertical ordering 158
script-priority 158, 938
seconda volta 191
\section 132, 147, 566
section bar line 132, 147
section bar line, in Gregorian chant 556
section bar line, in Kievan chant 566
section label 313
section parts 262
\sectionLabel 197, 313, 958
segno 198
\segno 156, 861, 930
segno bar line 123, 132, 199
'segno' sign 156, 930
'segno' sign, bar line 123, 199
'segno' sign, manual 207
'segno' sign, with repeats 196, 199
segno, forced at start 199, 207
\segnoMark 131, 132, 199, 207, 958
self-alignment-interface 794
self-alignment-X 697, 794
self-alignment-Y 796
semai form 579
semi-flat 8, 11
semi-flat symbol, appearance 577
semi-flat symbol, Persian (koron) 583
semi-sharp 8, 11
semi-sharp symbol, Persian (sori) 583
semi-transparent colors 290
\semicirculus 557, 930
'semicirculus' Gregorian articulation 557, 930
\semiflat 861
\semiGermanChords 517
\semisharp 861
senza misura 82
\senzaMisura 82, 958
\senzaMisuraTimeSignatureOff 82
\senzaMisuraTimeSignatureX 82
separate text 318
separator mark, system 249
\serif 817
sesqui-flat 11
sesqui-sharp 11
\sesquiflat 861
\sesquisharp 861
\set 106, 760, 764
set-global-staff-size 685
\setDefaultMeasureLength 79, 958
\setMeasureLengthFromHere 79, 958
setting extent of text object 885
setting horizontal text alignment 827

setting subscript, in standard font size 813
setting superscript, in standard font size 813
\setupTableOfContents 627, 958
seventh chord 509
\sf 160
\sff 160
\sfz 160
\shape 801, 958
shape note 46
shape note, merging 49
shaping slurs and ties 801
shared property 941
sharp 8
\sharp 861
sharp, double 8
shift note 227
shift rest, automatic 227
shift symbol, accordion 424
shift, fingering 285
\shiftDurations 959
shifting voice 227
\shiftOff 227
\shiftOn 227
\shiftOnn 227
\shiftOnnn 227
short bar line 122
short bar line, in Gregorian chant 556
short fermata 156, 929
short-indent 263, 678, 972
shortened volta brackets 202
\shortfermata 156, 929
show vertical layout 729
showFirstLength 650, 941
\showKeySignature 498
showLastLength 650, 941
\showStaffSwitch 418
side-axis 793
side-position-interface 793
sidestick 931
sign, coda 156, 930
sign, coda, manual 207
sign, coda, variant 156, 201, 930
sign, coda, with repeats 196
sign, conducting 112
sign, segno 156, 930
sign, segno, bar line 123, 199
sign, segno, manual 207
sign, segno, with repeats 196, 199
sign, snappizzicato 156, 930
sign, variant coda 156, 201, 930
signature, polymetric 95
'signum congruentiae' mensural notation 930
'signum congruentiae' ornament 156
\signumcongruentiae 156, 930
silenced note, percussion 494
\simple 818
simple repeat 190
simple text string, with tie characters 863
simultaneous notes and accidentals 38
simultaneous phrasing slurs 175
simultaneous slurs 171
singer name 382
\single 617, 764, 766, 769, 959
single bar line 132
single staff, with bracket or brace 246
single-line comment 592, 598
single-staff polyphony 222
skip 69
\skip 69, 371, 959
skip typesetting 650
skipBars 276
skipping notes, in lyrics 371
skipTypesetting 650
slash repeat 209
'slash turn' ornament 156, 928
slashChordSeparator 518
slashed beam 120
slashed digit 882
slashed note head 50
slashed stem 151
\slashed-digit 882
\slashedGrace 148, 959
\slashSeparator 681
\slashturn 156, 928
slide, in tablature 448
slope, ottava spanner, modifying 29
slur 171
slur, above notes 171
slur, adjusting start and end 172
slur, and repeats 205
slur, below notes 171
slur, dashed 171
slur, dashed phrasing 175
slur, defining dash patterns 172, 173
slur, defining dash patterns for phrasing 175
slur, dotted 171
slur, dotted phrasing 175
slur, double, for legato chords 173
slur, for tuplets 56
slur, half dashed and half solid 171
slur, half solid and half dashed phrasing 175
slur, manual placement 171
slur, modifying 801
slur, multiple 171
slur, multiple phrasing 175
slur, phrasing 171, 174
slur, phrasing, defining dash patterns 175
slur, simultaneous 171
slur, simultaneous phrasing 175
slur, solid 171
slur, style 171
slur, text markup inside 173
slur, vertical 184
slur-event 268
\slurDashed 171
\slurDashPattern 172, 959
\slurDotted 171
\slurDown 171
\slurHalfDashed 171
\slurHalfSolid 171
\slurNeutral 171
\slurSolid 171
\slurUp 172
\small 281, 327, 818
\smallCaps 818
\smaller 324, 327, 818
smaller note 269
smob 942
snap pizzicato 429
\snappizzicato 156, 930

'snappizzicato' sign 156, 930
snare 931
Solesmes 539
solid slur 171
solo part 233
soprano clef 19, 923
sori 583
sos. 421
sostenuto pedal 421
\sostenutoOff 421
\sostenutoOn 421
sound 654
\sourcefileline 598
\sourcefilename 598
Southern Harmony note head 46
\southernHarmonyHeads 46
\southernHarmonyHeadsMinor 47
\sp 160
space, between staves 697
space, in lyrics 346, 353
space, inside systems 697
space-alist 718
spacer note 69
spacer rest 69
spacing 713
spacing section, new 714
spacing, display of layout 729
spacing, horizontal 713
spacing, lyrics 363
spacing, vertical 697
span bar 126
Span_stem_engraver 420
spanner 776, 942
spanner, modifying 776
special arpeggio symbol 183
special character 647
special character, in markup mode 320
special note head 43
splash cymbal 931
splicing, into tagged music 637
splitting notes 98
splitting rests 98
\spp 160
Sprechgesang 399
square bracket, at start of staff group 246
square neumes ligature 558
\staccatissimo 156, 928
'staccatissimo' articulation 156, 928
\staccato 156, 928
'staccato' articulation 156, 928
stacked flags 118
stacking text, in a column 823
Staff 737
staff brace, in markup 878
staff change, automatic 416
staff change, manual 414
staff group 244
staff group, with square bracket at start 246
staff line, modifying 250
staff line, stopping and starting 250
staff symbol 250
staff, choir 244
staff, distance 697
staff, drum 242
staff, empty 257
staff, exact positioning 704
staff, Frenched 254
staff, grand 244
staff, hiding 257
staff, highlight 292
staff, initiation 242
staff, instantiation 242
staff, keyboard instruments 414
staff, keyed instruments 414
staff, metronome mark below 90
staff, multiple 244
staff, nested 248
staff, new 242
staff, percussion 242
staff, piano 245, 414
staff, resizing 254
staff, single 242
staff, single, with bracket or brace 246
staff, size 685
staff, switching 418
staff-affinity 697
staff-change line 418
staff-height 972
staff-padding 284
\staff-space 775, 972
staff-staff-spacing 697
Staff_collecting_engraver 144, 317
Staff_symbol_engraver 257
StaffGroup 737
staffgroup-staff-spacing 697
StaffGrouper 697
\staffHighlight 292, 959
stand-alone text 318
StandaloneRhythmScore 737
StandaloneRhythmStaff 738
StandaloneRhythmVoice 739
standard font size (notation) 282
\stanza 377, 959
stanza number 377
stanza reminder 378
stanzaReminders 378
stanzaReminderText 378
start of system 244
start repeat 205
start-repeat 205
startAcciaccaturaMusic 151
startAppoggiaturaMusic 151
startGraceMusic 151
\startGradualTempoChange 959
\startGroup 304
\startOptionalMaterial 295
\startStaff 250, 254, 737
\startTextSpan 312, 783, 784
\startTrillSpan 186
staves, divisi 260
\stdBass 870
\stdBassIV 871
\stdBassV 872
\stdBassVI 873
stem 297
stem, automatic direction on center line 298
stem, cross-staff 420
stem, default direction on center line 297
stem, direction 297
stem, down 297

stem, in tablature 444
stem, invisible 297
stem, neutral 297
stem, up 297
stem, with slash 151
stem-spacing-correction 713
\stemDown 297
stemLeftBeamCount 116, 117
\stemNeutral 297
stemRightBeamCount 116, 117
\stemUp 297
stencil 942
\stencil 883
stencil, removing 785
stereo balance, in MIDI 664
stopAcciaccaturaMusic 151
stopAppoggiaturaMusic 151
stopGraceMusic 151
\stopGradualTempoChange 959
\stopGroup 304
\stopOptionalMaterial 295
\stopped 156, 491, 496, 930
'stopped' articulation 156, 496, 930
\stopStaff 250, 254, 257, 737
\stopStaffHighlight 292
\stopTextSpan 312, 783, 784
\stopTrillSpan 186
\storePredefinedDiagram 470, 475, 959
straight flags, modern 118
straight flags, old 118
strict spacing, and grace notes 152
strict-beat beaming 111
strictBeatBeaming 111
strictly alternating meter 131
string bending, in tablature 440
string number 428, 432
string number, orientation 283
string number, vs. fingering 432
string, automatic replacement 649
string, indicating open 428
\string-lines 887
stringNumberOrientations 283
strings, orchestral 427
strings, writing for 427
\stringTuning 452, 960
stringTunings 452, 466
stroke finger, orientation 283
strokeFingerOrientations 283, 480
\stropha 558, 563
strumming rhythm, showing 100, 101
\strut 883
style, bar number 136
style, double repeat for volta 133
style, measure number 136
style, multi-measure rests 276
style, note heads 43
style, rehearsal mark 143
style, rests 68
style, slur 171
style, text dynamics 168
style, voice 227
\styledNoteHeads 960
\sub 325, 818
subbass clef 19, 923
subdivideBeams 110
subdividing beams 110
submeasure bar line 131
\submeasureBarsOn 131
submeasureStructure 83, 131
subscript 325
subscript text 818
substitution function, and relative octave entry .. 646
subsubtitle 603
subtitle 603
suggestAccidentals 159, 547
\super 325, 819
superscript 325
superscript text 819
suppressing repeated chords 474, 515
sus 512
sustain pedal 421
sustain pedal, style 422
\sustainOff 421
\sustainOn 421
SVG output 651
swing 859
swing.ly script 665
switching font 324
syllable duration, automatic 349
symbol, break-align 718
symbol, breath mark, changing 176
symbol, major seven 519
symbol, non-musical 333
syntax, markup 320
system 244
system separator mark 249
system start delimiter 244
system start delimiter, nested 248
system-count 679, 972
system-separator-markup 681, 972
system-system-spacing 676, 972
systems-per-page 679, 972

T

tab clef 454, 924
\tabChordRepeats 435, 960
\tabChordRepetition 960
\tabFullNotation 434
tablature 243, 431
tablature, and beams 444
tablature, and harmonic indications 438
tablature, and harmonics 445
tablature, and polyphony 444
tablature, and slides 448
tablature, and stems 444
tablature, and string bending 440
tablature, banjo 431, 452, 484
tablature, basic 433
tablature, bass 452
tablature, bass guitar 452
tablature, cello 452
tablature, chord glissando 449
tablature, custom 452
tablature, custom string tunings 452
tablature, default 433
tablature, double bass 452
tablature, guitar 431, 452
tablature, hammer on 450

tablature, lute ... 485
tablature, mandolin ... 452
tablature, microtones ... 454
tablature, predefined string tunings ... 452
tablature, pull off ... 450
tablature, quarter tones ... 454
tablature, ukulele ... 452
tablature, viola ... 452
tablature, violin ... 452
\table ... 887
table of contents ... 625
table of contents, customized functions ... 626
\table-of-contents ... 627, 888
TabStaff ... 243, 433, 737
TabVoice ... 433, 739
tag ... 631
\tag ... 631, 883, 960
tag groups ... 634
\tag, and quoting music ... 632
\tag-list ... 640, 888
\tagGroup ... 634, 960
\tagGroupRef ... 635, 960
tagline ... 603, 972
tam tam ... 931
tambourine ... 931
\taor ... 498
taqasim ... 579
teaching ... 37
'teaching' accidental style ... 37
\teeny ... 281, 327, 819
template, Arabic music ... 580
tempo ... 87
\tempo ... 87
tempo, change, without metronome mark ... 91
tempo, with rhythm ... 859
tempoCountPrecision ... 88
\temporary ... 764, 768, 777, 960
tenor clef ... 19, 923
tenor clef, choral ... 20
tenor G clef ... 923
tenor varC clef ... 923
\tenuto ... 156, 928
'tenuto' articulation ... 156, 928
text alignment, commands ... 332
text column, left-aligned ... 832
text column, right-aligned ... 836
text dynamics, style ... 168
text encoding ... 647
text item, non-empty ... 310
text mark ... 314
text mark, alignment ... 316
text mark, below staff ... 316
text mark, on every staff ... 317
text mark, vertical stacking ... 316
text markup ... 320
text markup, inside slurs ... 173
text object ... 308
text script ... 310
text script, breaking vertical alignment ... 167
text script, vertical alignment ... 166
text size ... 324
text spanner ... 312
text spanner, dynamics, customize ... 313
text spanner, formatting ... 312
text, aligning ... 327
text, at beginning of line ... 314
text, at end of score ... 314
text, between notes ... 314
text, centering on page ... 331
text, decorating ... 332
text, framing ... 332
text, horizontal alignment ... 327
text, in columns ... 319, 330
text, in volta bracket ... 207
text, justified ... 331
text, keeping inside margin ... 311
text, line width ... 331
text, multi-line ... 330
text, on bar line ... 313, 314
text, on multi-measure rest ... 72
text, other languages ... 308
text, ottavation ... 28
text, outside margin ... 311
text, padding ... 333
text, separate ... 318
text, spread over multiple pages ... 322
text, stand-alone ... 318
text, top-level ... 318
text, vertical alignment ... 328
text, with a tie above ... 816, 819
text, with a tie below ... 819, 820
text, word-wrapped ... 331
\text-accidental ... 862
\text-common-time ... 862
\text-cut-time ... 862
\text-doubleflat ... 862
\text-doublesharp ... 862
\text-flat ... 863
text-font-size ... 324, 973
\text-natural ... 863
\text-sharp ... 863
Text_mark_engraver ... 317
\textEndMark ... 314, 960
\textLengthOff ... 73, 75, 311
\textLengthOn ... 73, 75, 166, 311
\textMark ... 314, 960
\textSpannerDown ... 312
\textSpannerNeutral ... 312
\textSpannerUp ... 312
thorough bass ... 526
\thumb ... 156, 283
'thumb' fingering ... 156, 283, 930
tick bar line ... 122
tick bar line, in Gregorian chant ... 556
tick mark ... 176
tie ... 63
\tie ... 819
tie, above text ... 816, 819
tie, alternative endings ... 64
tie, and volta bracket ... 64
tie, appearance ... 64
tie, below text ... 819, 820
tie, dashed ... 64
tie, dotted ... 64
tie, from nothing ... 64
tie, in chord ... 63
tie, in lyrics ... 353
tie, in repeats ... 64
tie, laissez vibrer ... 64
tie, manual engraving ... 66

tie, modifying ... 801
tie, placement ... 64
tie, repeating ... 64
tie, to nothing ... 64
tie, with arpeggios ... 65
TieColumn ... 66
tied note, accidental ... 8
\tied-lyric ... 863
\tieDashed ... 64
\tieDashPattern ... 64, 960
\tieDotted ... 64
\tieDown ... 64
\tieHalfDashed ... 64
\tieHalfSolid ... 64
\tieNeutral ... 64
\tieSolid ... 64
\tieUp ... 64
tieWaitForNote ... 65
timbale ... 931
\time ... 80, 106, 961
time administration ... 78
time signature ... 80
time signature, additive ... 81
time signature, alternating ... 81
time signature, and volta repeats ... 87
time signature, complex ... 81
time signature, default settings ... 83
time signature, double ... 95
time signature, mensural ... 544
time signature, mid-measure ... 92
time signature, polymetric ... 95
time signature, printing only numerator ... 85
time signature, properties, restoring default values ... 84
time signature, style ... 81, 544
time signature, visibility ... 80
time, common ... 81
time, cut ... 81
time, free ... 82
\timeAbbrev ... 961
\times ... 961
timeSignature ... 95
Timing ... 78, 736
timing ... 78
timing information, and repeats ... 205
timing mark, for glissando ... 179
timing, within score ... 78
\tiny ... 281, 327, 819
title ... 600
title ... 603
toc ... 625
tocFormatMarkup ... 627, 973
tocIndentMarkup ... 627, 973
\tocItem ... 627, 961
tocItemMarkup ... 627, 973
\tocItemWithDotsMarkup ... 626
tocTitleMarkup ... 627, 973
\toe ... 156, 423, 930
'toe' pedal mark ... 156, 423, 930
toeHeelStyle ... 423
tom tom ... 931
top-level text ... 318
top-margin ... 674, 973
top-markup-spacing ... 676, 973
top-system-spacing ... 676, 973
toplevel-bookparts ... 941
toplevel-scores ... 941
transcription, mensural to modern ... 572
transcription, modern of Gregorian music ... 405
transcription, of mensural music ... 247
transformation, modal ... 17
transformation, retrograde ... 17
\translate ... 328, 836
\translate-scaled ... 328, 837
translating text ... 836, 837
transparency, semi ... 290
\transparent ... 883
transparent note ... 289
transparent objects ... 785
\transpose ... 6, 13, 16, 961
transposed clef, visibility ... 790
\transposedCueDuring ... 272, 961
transposing ... 13
transposing clef ... 20
transposing fret diagram ... 468
transposing instrument ... 14, 30
transposition ... 13
\transposition ... 30, 266, 961
transposition, and relative octave entry ... 6
transposition, instrument ... 30
transposition, MIDI ... 30
transposition, modal ... 17
transposition, of notes ... 13
transposition, of pitches ... 13
transposition, pitches, 'smart' ... 15
tre corde ... 421
treble clef ... 19, 923
\treCorde ... 421
tremolo ... 212
tremolo beam ... 212
tremolo mark ... 213
tremolo, cross-staff ... 213
triad ... 509
triangle ... 931
\triangle ... 333, 850
trill ... 186
\trill ... 156, 186, 928
'trill' ornament ... 156, 928
trill, pitched ... 187
trill, with accidental ... 187
triplet ... 55
triplet, formatting ... 57
\tripletFeel ... 666
true (#t) ... 974
tuning, banjo ... 485
tuning, lute ... 486
tuning, non-Western ... 575
tuplet ... 55
\tuplet ... 55, 95, 961
tuplet bracket, note head side ... 59
tuplet bracket, placement ... 56
tuplet bracket, visibility ... 56, 58
tuplet number, changing ... 57
tuplet number, non-default ... 57
tuplet number, visibility ... 56
tuplet slur ... 56
tuplet, beamed, line break within ... 60
tuplet, entering multiple ... 57
tuplet, formatting ... 57
tuplet, grouping ... 55
tuplet-slur ... 56

\tupletDown 56
\tupletNeutral 56
TupletNumber 57
\tupletSpan 57, 962
tupletSpannerDuration 57
\tupletUp 56
Turkish makam 575
Turkish makam, example 582
Turkish music 581
Turkish note name 581
Turkish, classical music 575
\turn 156, 928
'turn' ornament 156, 928
'turn' ornament, delayed 159
\tweak 764, 962
\tweak, relation to \override 764
tweaking 764
tweaking control point 766
tweaking grace note 150, 151
two-column text 319
two-sided 678, 973
\type 751, 752
typeface 940
typesetting text 320
typesetting, skip 650
\typewriter 820

U

U.C. 421
ukulele 457
ultima volta 191
una corda 421
\unaCorda 421
\underline 324, 820
underlining text 820
underlying repeat bar line 131
\undertie 820
\undo 764, 769, 962
\unfolded 193, 962
unfolding repeat 189
\unfoldRepeats 193, 661, 962
\unHideNotes 289
Unicode 648
units, of measuring 775
universal-color 291, 906
\unless 611, 864
unmetered music 78, 93
unmetered music, accidentals 93
unmetered music, bar lines 93
unmetered music, bar numbers 93
unmetered music, beams 93
unmetered music, line breaks 94
unmetered music, page breaks 94
\unset 760
unspecified pitch 10
up direction (^) 774
'up pralltriller' ornament 156, 928
'up-bow' bowing indication 156, 428, 930
UP 793, 796
upbeat 92
upbeat, in a repeat 191
\upbow 156, 428, 930
\upmordent 156, 928
'upper mordent' ornament 156, 928
\upprall 156, 928
\upright 821
URL link, as QR code 881
UTF-8 647

V

varbaritone clef 19
varC clef 923
\varcoda 156, 863, 930
\varheel 156, 423, 930
variable 597
variable, and \book 594
variable, and \bookpart 594
variable, using 629
variant 'coda' sign 156, 201, 930
Variation Font format (VF) 338
\vartoe 156, 423, 930
Vaticana, Editio 538, 539
VaticanaLyrics 555, 738
VaticanaScore 555, 737
VaticanaStaff 243, 555, 737
VaticanaVoice 555, 738
\vcenter 837
\verbatim-file 884
\verboseBarNumbers 962
\version 598
\versus 559, 962
vertical alignment, dynamics 166
vertical alignment, dynamics, breaking 167
vertical alignment, text 328
vertical alignment, text scripts 166
vertical direction, default, of grobs 774
vertical direction, forced, of grobs 774
vertical distance, figured bass elements 532
vertical extent, of grobs 791
vertical line, between staves 302
vertical movement, in markup 328, 329
vertical ordering, of scripts 158
vertical positioning, of dynamics 163
vertical slur 184
vertical spacing 697, 722
vertical spacing, debugging 729
VerticalAxisGroup 697
vertically centering text 837
\verylongfermata 156, 929
\veryshortfermata 156, 929
vibraslap 931
violin clef 19, 923
\virga 558, 563
\virgula 557
visibility of object 784
visibility of transposed clef 790
visibility of tuplet brackets 58
visibility of tuplets 56
vocal score 392
vocal score, adding cues 397
voice 222
Voice 222, 738
voice 33
'voice' accidental style 33
voice style 227
voice, additional, in polyphonic music 230
voice, ambitus 41

voice, divided ... 391
voice, extra, for handling breaks ... 692
voice, following ... 418
voice, multiple ... 227
voice, \partCombine with \autoBeamOff ... 104
voice, quoting ... 266, 269
voice, shifting ... 227
voice, with ottavation ... 28
\voiceFour ... 222
\voiceFourStyle ... 227
\voiceNeutralStyle ... 227
\voiceOne ... 222
\voiceOneStyle ... 227
\voices ... 225, 962
\voiceThree ... 222
\voiceThreeStyle ... 227
\voiceTwo ... 222
\voiceTwoStyle ... 227
\void ... 667, 963
volta ... 190, 191
\volta ... 191, 193, 963
volta bracket ... 205
volta bracket, and tie ... 64
volta bracket, in additional staves ... 202
volta bracket, shortened ... 202
volta bracket, with text ... 207
volta repeat, and time signatures ... 87
volta repeat, bar line at start of piece ... 190
volta repeat, below chords ... 520
volta, double repeat style ... 133
volta, prima ... 191
volta, seconda ... 191
volta, ultima ... 191
\volta-number ... 821
Volta_engraver ... 202
vowel transition ... 357
\vowelTransition ... 357
\vshape ... 802, 963
\vspace ... 329, 837

W

Walker shape note head ... 46
\walkerHeads ... 46
\walkerHeadsMinor ... 47
whistle ... 931
white mensural ligature ... 549
whiteout ... 65, 785
\whiteout ... 884
whitespace ... 599
\whiteTriangleMarkup ... 517
whole rest, for a full measure ... 71
wind instrument ... 495
wind instrument, fingering symbols ... 496
\with ... 745, 749
\with-color ... 289, 885
\with-dimension ... 885
\with-dimension-from ... 885
\with-dimensions ... 885
\with-dimensions-from ... 885
\with-link ... 885
\with-outline ... 885
\with-string-transformer ... 821
\with-true-dimension ... 885
\with-true-dimensions ... 886
\with-url ... 851
\withMusicProperty ... 963
\withRelativeDir ... 963
woodblock ... 931
woodwind diagram, key lists ... 506
woodwind diagram, list ... 503
woodwind diagram, modifying ... 505
\woodwind-diagram ... 868
word-wrapping text ... 331
\wordwrap ... 331, 838
\wordwrap-field ... 838
\wordwrap-lines ... 322, 889
\wordwrap-string ... 839
writing music in parallel ... 238
written-out repeat ... 189

X

x11-color ... 290, 291, 903
X ... 793
X-offset ... 697, 793
X11 color ... 290, 291
\xNote ... 44, 963
\xNotesOff ... 44
\xNotesOn ... 44

Y

Y ... 793
Y-offset ... 793