LilyPond

The music typesetter

Internals Reference

The LilyPond development team

Copyright © 2000–2021 by the authors

For LilyPond version 2.23.5

Table of Contents

1 Music definitions 2

1.1 Music expressions 2

1.1.1 AbsoluteDynamicEvent 2
1.1.2 AdHocMarkEvent 2
1.1.3 AlternativeEvent 2
1.1.4 AnnotateOutputEvent 3
1.1.5 ApplyContext 3
1.1.6 ApplyOutputEvent 3
1.1.7 ArpeggioEvent 4
1.1.8 ArticulationEvent 4
1.1.9 BarCheck 5
1.1.10 BassFigureEvent 5
1.1.11 BeamEvent 5
1.1.12 BeamForbidEvent 6
1.1.13 BendAfterEvent 6
1.1.14 BendSpanEvent 6
1.1.15 BreakDynamicSpanEvent 7
1.1.16 BreathingEvent 7
1.1.17 ClusterNoteEvent 7
1.1.18 CompletizeExtenderEvent 8
1.1.19 ContextChange 8
1.1.20 ContextSpeccedMusic 8
1.1.21 CrescendoEvent 9
1.1.22 DecrescendoEvent 9
1.1.23 DoublePercentEvent 10
1.1.24 DurationLineEvent 10
1.1.25 EpisemaEvent 10
1.1.26 Event 11
1.1.27 EventChord 11
1.1.28 ExtenderEvent 11
1.1.29 FineEvent 12
1.1.30 FingerGlideEvent 12
1.1.31 FingeringEvent 13
1.1.32 FootnoteEvent 13
1.1.33 GlissandoEvent 13
1.1.34 GraceMusic 14
1.1.35 HarmonicEvent 14
1.1.36 HyphenEvent 14
1.1.37 KeyChangeEvent 15
1.1.38 LabelEvent 15
1.1.39 LaissezVibrerEvent 15
1.1.40 LigatureEvent 16
1.1.41 LineBreakEvent 16
1.1.42 LyricCombineMusic 16
1.1.43 LyricEvent 17
1.1.44 MeasureCounterEvent 17
1.1.45 MeasureSpannerEvent 18
1.1.46 MultiMeasureArticulationEvent 18

1.1.47 MultiMeasureRestEvent 18
1.1.48 MultiMeasureRestMusic 19
1.1.49 MultiMeasureTextEvent 19
1.1.50 Music 19
1.1.51 NoteEvent 20
1.1.52 NoteGroupingEvent 20
1.1.53 OttavaEvent 20
1.1.54 OverrideProperty 21
1.1.55 PageBreakEvent 21
1.1.56 PageTurnEvent 22
1.1.57 PartCombineMusic 22
1.1.58 PartialSet 22
1.1.59 PercentEvent 23
1.1.60 PercentRepeatedMusic 23
1.1.61 PesOrFlexaEvent 24
1.1.62 PhrasingSlurEvent 24
1.1.63 PostEvents 24
1.1.64 PropertySet 25
1.1.65 PropertyUnset 25
1.1.66 QuoteMusic 25
1.1.67 RehearsalMarkEvent 26
1.1.68 RelativeOctaveCheck 26
1.1.69 RelativeOctaveMusic 27
1.1.70 RepeatSlashEvent 27
1.1.71 RepeatTieEvent 27
1.1.72 RestEvent 28
1.1.73 RevertProperty 28
1.1.74 ScriptEvent 29
1.1.75 SectionEvent 29
1.1.76 SegnoEvent 29
1.1.77 SequentialAlternativeMusic 30
1.1.78 SequentialMusic 30
1.1.79 SimultaneousMusic 31
1.1.80 SkipEvent 31
1.1.81 SkipMusic 32
1.1.82 SlurEvent 32
1.1.83 SoloOneEvent 32
1.1.84 SoloTwoEvent 33
1.1.85 SostenutoEvent 33
1.1.86 SpacingSectionEvent 34
1.1.87 SpanEvent 34
1.1.88 StaffSpanEvent 34
1.1.89 StringNumberEvent 35
1.1.90 StrokeFingerEvent 35
1.1.91 SustainEvent 35
1.1.92 TempoChangeEvent 36
1.1.93 TextScriptEvent 36
1.1.94 TextSpanEvent 36
1.1.95 TieEvent 37
1.1.96 TimeScaledMusic 37
1.1.97 TimeSignatureEvent 37
1.1.98 TimeSignatureMusic 38
1.1.99 TransposedMusic 38
1.1.100 TremoloEvent 39

1.1.101 TremoloRepeatedMusic 39
1.1.102 TremoloSpanEvent 40
1.1.103 TrillSpanEvent 40
1.1.104 TupletSpanEvent 40
1.1.105 UnaCordaEvent 41
1.1.106 UnfoldedRepeatedMusic 41
1.1.107 UnfoldedSpeccedMusic 41
1.1.108 UnisonoEvent 42
1.1.109 UnrelativableMusic 42
1.1.110 VoiceSeparator 43
1.1.111 VoltaRepeatedMusic 43
1.1.112 VoltaSpanEvent 44
1.1.113 VoltaSpeccedMusic 44
1.1.114 VowelTransitionEvent 45
1.2 Music classes 45
1.2.1 absolute-dynamic-event 45
1.2.2 ad-hoc-mark-event 45
1.2.3 alternative-event 45
1.2.4 annotate-output-event 45
1.2.5 apply-output-event 45
1.2.6 arpeggio-event 45
1.2.7 articulation-event 45
1.2.8 bass-figure-event 46
1.2.9 beam-event 46
1.2.10 beam-forbid-event 46
1.2.11 bend-after-event 46
1.2.12 bend-span-event 46
1.2.13 break-dynamic-span-event 46
1.2.14 break-event 46
1.2.15 break-span-event 46
1.2.16 breathing-event 46
1.2.17 cluster-note-event 46
1.2.18 completize-extender-event 47
1.2.19 crescendo-event 47
1.2.20 decrescendo-event 47
1.2.21 double-percent-event 47
1.2.22 duration-line-event 47
1.2.23 dynamic-event 47
1.2.24 episema-event 47
1.2.25 extender-event 47
1.2.26 fine-event 47
1.2.27 finger-glide-event 47
1.2.28 fingering-event 47
1.2.29 footnote-event 48
1.2.30 glissando-event 48
1.2.31 harmonic-event 48
1.2.32 hyphen-event 48
1.2.33 key-change-event 48
1.2.34 label-event 48
1.2.35 laissez-vibrer-event 48
1.2.36 layout-instruction-event 48
1.2.37 ligature-event 48
1.2.38 line-break-event 48
1.2.39 lyric-event 48

1.2.40 mark-event 49
1.2.41 measure-counter-event 49
1.2.42 measure-spanner-event 49
1.2.43 melodic-event 49
1.2.44 multi-measure-articulation-event 49
1.2.45 multi-measure-rest-event 49
1.2.46 multi-measure-text-event 49
1.2.47 music-event 49
1.2.48 note-event 50
1.2.49 note-grouping-event 50
1.2.50 ottava-event 50
1.2.51 page-break-event 50
1.2.52 page-turn-event 50
1.2.53 part-combine-event 50
1.2.54 pedal-event 51
1.2.55 percent-event 51
1.2.56 pes-or-flexa-event 51
1.2.57 phrasing-slur-event 51
1.2.58 rehearsal-mark-event 51
1.2.59 repeat-slash-event 51
1.2.60 repeat-tie-event 51
1.2.61 rest-event 51
1.2.62 rhythmic-event 51
1.2.63 script-event 51
1.2.64 section-event 52
1.2.65 segno-event 52
1.2.66 skip-event 52
1.2.67 slur-event 52
1.2.68 solo-one-event 52
1.2.69 solo-two-event 52
1.2.70 sostenuto-event 52
1.2.71 spacing-section-event 52
1.2.72 span-dynamic-event 52
1.2.73 span-event 52
1.2.74 staff-span-event 53
1.2.75 StreamEvent 53
1.2.76 string-number-event 53
1.2.77 stroke-finger-event 53
1.2.78 sustain-event 53
1.2.79 tempo-change-event 54
1.2.80 text-script-event 54
1.2.81 text-span-event 54
1.2.82 tie-event 54
1.2.83 time-signature-event 54
1.2.84 tremolo-event 54
1.2.85 tremolo-span-event 54
1.2.86 trill-span-event 54
1.2.87 tuplet-span-event 54
1.2.88 una-corda-event 54
1.2.89 unisono-event 54
1.2.90 volta-span-event 55
1.2.91 vowel-transition-event 55
1.3 Music properties 55

2 Translation 61
2.1 Contexts 61
2.1.1 ChoirStaff 61
2.1.2 ChordNames 63
2.1.3 CueVoice 65
2.1.4 Devnull 76
2.1.5 DrumStaff 76
2.1.6 DrumVoice 82
2.1.7 Dynamics 92
2.1.8 FiguredBass 95
2.1.9 FretBoards 97
2.1.10 Global 99
2.1.11 GrandStaff 99
2.1.12 GregorianTranscriptionStaff 101
2.1.13 GregorianTranscriptionVoice 111
2.1.14 KievanStaff 122
2.1.15 KievanVoice 132
2.1.16 Lyrics 143
2.1.17 MensuralStaff 145
2.1.18 MensuralVoice 155
2.1.19 NoteNames 166
2.1.20 NullVoice 168
2.1.21 OneStaff 170
2.1.22 PetrucciStaff 171
2.1.23 PetrucciVoice 181
2.1.24 PianoStaff 192
2.1.25 RhythmicStaff 194
2.1.26 Score 197
2.1.27 Staff 220
2.1.28 StaffGroup 229
2.1.29 TabStaff 231
2.1.30 TabVoice 240
2.1.31 VaticanaStaff 251
2.1.32 VaticanaVoice 261
2.1.33 Voice 271
2.2 Engravers and Performers 282
2.2.1 Accidental_engraver 282
2.2.2 Alteration_glyph_engraver 283
2.2.3 Ambitus_engraver 283
2.2.4 Arpeggio_engraver 284
2.2.5 Auto_beam_engraver 284
2.2.6 Axis_group_engraver 285
2.2.7 Balloon_engraver 285
2.2.8 Bar_engraver 285
2.2.9 Bar_number_engraver 286
2.2.10 Beam_collision_engraver 287
2.2.11 Beam_engraver 287
2.2.12 Beam_performer 288
2.2.13 Beat_engraver 288
2.2.14 Beat_performer 288
2.2.15 Bend_engraver 289
2.2.16 Bend_spanner_engraver 289
2.2.17 Break_align_engraver 290
2.2.18 Breathing_sign_engraver 290

2.2.19 Centered_bar_number_align_engraver 290
2.2.20 Chord_name_engraver 290
2.2.21 Chord_tremolo_engraver 291
2.2.22 Clef_engraver 291
2.2.23 Cluster_spanner_engraver 292
2.2.24 Collision_engraver 292
2.2.25 Completion_heads_engraver 292
2.2.26 Completion_rest_engraver 293
2.2.27 Concurrent_hairpin_engraver 294
2.2.28 Control_track_performer 294
2.2.29 Cue_clef_engraver 294
2.2.30 Custos_engraver 295
2.2.31 Default_bar_line_engraver 295
2.2.32 Dot_column_engraver 295
2.2.33 Dots_engraver 296
2.2.34 Double_percent_repeat_engraver 296
2.2.35 Drum_note_performer 296
2.2.36 Drum_notes_engraver 296
2.2.37 Duration_line_engraver 297
2.2.38 Dynamic_align_engraver 297
2.2.39 Dynamic_engraver 297
2.2.40 Dynamic_performer 298
2.2.41 Episema_engraver 298
2.2.42 Extender_engraver 299
2.2.43 Figured_bass_engraver 299
2.2.44 Figured_bass_position_engraver 299
2.2.45 Finger_glide_engraver 300
2.2.46 Fingering_column_engraver 300
2.2.47 Fingering_engraver 300
2.2.48 Font_size_engraver 300
2.2.49 Footnote_engraver 300
2.2.50 Forbid_line_break_engraver 301
2.2.51 Fretboard_engraver 301
2.2.52 Glissando_engraver 302
2.2.53 Grace_auto_beam_engraver 302
2.2.54 Grace_beam_engraver 302
2.2.55 Grace_engraver 303
2.2.56 Grace_spacing_engraver 303
2.2.57 Grid_line_span_engraver 303
2.2.58 Grid_point_engraver 303
2.2.59 Grob_pq_engraver 303
2.2.60 Horizontal_bracket_engraver 304
2.2.61 Hyphen_engraver 304
2.2.62 Instrument_name_engraver 304
2.2.63 Instrument_switch_engraver 305
2.2.64 Jump_engraver 305
2.2.65 Keep_alive_together_engraver 305
2.2.66 Key_engraver 305
2.2.67 Key_performer 306
2.2.68 Kievan_ligature_engraver 307
2.2.69 Laissez_vibrer_engraver 307
2.2.70 Ledger_line_engraver 307
2.2.71 Ligature_bracket_engraver 307
2.2.72 Lyric_engraver 307

2.2.73 Lyric_performer 308
2.2.74 Mark_engraver 308
2.2.75 Mark_tracking_translator 308
2.2.76 Measure_counter_engraver 309
2.2.77 Measure_grouping_engraver 309
2.2.78 Measure_spanner_engraver 309
2.2.79 Melody_engraver 310
2.2.80 Mensural_ligature_engraver 310
2.2.81 Merge_mmrest_numbers_engraver 310
2.2.82 Merge_rests_engraver 310
2.2.83 Metronome_mark_engraver 310
2.2.84 Midi_control_change_performer 311
2.2.85 Multi_measure_rest_engraver 311
2.2.86 New_fingering_engraver 312
2.2.87 Note_head_line_engraver 312
2.2.88 Note_heads_engraver 313
2.2.89 Note_name_engraver 313
2.2.90 Note_performer 313
2.2.91 Note_spacing_engraver 313
2.2.92 Ottava_spanner_engraver 314
2.2.93 Output_property_engraver 314
2.2.94 Page_turn_engraver 314
2.2.95 Paper_column_engraver 315
2.2.96 Parenthesis_engraver 315
2.2.97 Part_combine_engraver 315
2.2.98 Percent_repeat_engraver 316
2.2.99 Phrasing_slur_engraver 316
2.2.100 Piano_pedal_align_engraver 316
2.2.101 Piano_pedal_engraver 317
2.2.102 Piano_pedal_performer 317
2.2.103 Pitch_squash_engraver 317
2.2.104 Pitched_trill_engraver 318
2.2.105 Pure_from_neighbor_engraver 318
2.2.106 Repeat_acknowledge_engraver 318
2.2.107 Repeat_tie_engraver 319
2.2.108 Rest_collision_engraver 320
2.2.109 Rest_engraver 320
2.2.110 Rhythmic_column_engraver 320
2.2.111 Script_column_engraver 320
2.2.112 Script_engraver 320
2.2.113 Script_row_engraver 321
2.2.114 Separating_line_group_engraver 321
2.2.115 Show_control_points_engraver 321
2.2.116 Slash_repeat_engraver 321
2.2.117 Slur_engraver 322
2.2.118 Slur_performer 322
2.2.119 Spacing_engraver 322
2.2.120 Span_arpeggio_engraver 322
2.2.121 Span_bar_engraver 323
2.2.122 Span_bar_stub_engraver 323
2.2.123 Span_stem_engraver 323
2.2.124 Spanner_break_forbid_engraver 323
2.2.125 Spanner_tracking_engraver 323
2.2.126 Staff_collecting_engraver 323

2.2.127 Staff_performer ... 324
2.2.128 Staff_symbol_engraver ... 324
2.2.129 Stanza_number_align_engraver ... 324
2.2.130 Stanza_number_engraver ... 324
2.2.131 Stem_engraver ... 324
2.2.132 System_start_delimiter_engraver ... 325
2.2.133 Tab_note_heads_engraver ... 325
2.2.134 Tab_staff_symbol_engraver ... 326
2.2.135 Tab_tie_follow_engraver ... 326
2.2.136 Tempo_performer ... 327
2.2.137 Text_engraver ... 327
2.2.138 Text_spanner_engraver ... 327
2.2.139 Tie_engraver ... 327
2.2.140 Tie_performer ... 328
2.2.141 Time_signature_engraver ... 328
2.2.142 Time_signature_performer ... 328
2.2.143 Timing_translator ... 329
2.2.144 Trill_spanner_engraver ... 330
2.2.145 Tuplet_engraver ... 330
2.2.146 Tweak_engraver ... 330
2.2.147 Vaticana_ligature_engraver ... 331
2.2.148 Vertical_align_engraver ... 331
2.2.149 Volta_engraver ... 331
2.3 Tunable context properties ... 331
2.4 Internal context properties ... 345

3 Backend ... 348

3.1 All layout objects ... 348
3.1.1 Accidental ... 348
3.1.2 AccidentalCautionary ... 349
3.1.3 AccidentalPlacement ... 350
3.1.4 AccidentalSuggestion ... 350
3.1.5 Ambitus ... 352
3.1.6 AmbitusAccidental ... 354
3.1.7 AmbitusLine ... 354
3.1.8 AmbitusNoteHead ... 355
3.1.9 Arpeggio ... 355
3.1.10 BalloonText ... 357
3.1.11 BarLine ... 358
3.1.12 BarNumber ... 361
3.1.13 BassFigure ... 363
3.1.14 BassFigureAlignment ... 363
3.1.15 BassFigureAlignmentPositioning ... 364
3.1.16 BassFigureBracket ... 365
3.1.17 BassFigureContinuation ... 365
3.1.18 BassFigureLine ... 366
3.1.19 Beam ... 366
3.1.20 BendAfter ... 368
3.1.21 BendSpanner ... 369
3.1.22 BreakAlignGroup ... 371
3.1.23 BreakAlignment ... 372
3.1.24 BreathingSign ... 373
3.1.25 CenteredBarNumber ... 375
3.1.26 CenteredBarNumberLineSpanner ... 376

3.1.27 ChordName 377
3.1.28 Clef 378
3.1.29 ClefModifier 381
3.1.30 ClusterSpanner 382
3.1.31 ClusterSpannerBeacon 383
3.1.32 CombineTextScript 383
3.1.33 ControlPoint 385
3.1.34 ControlPolygon 386
3.1.35 CueClef 387
3.1.36 CueEndClef 389
3.1.37 Custos 392
3.1.38 DotColumn 394
3.1.39 Dots 394
3.1.40 DoublePercentRepeat 395
3.1.41 DoublePercentRepeatCounter 396
3.1.42 DoubleRepeatSlash 398
3.1.43 DurationLine 399
3.1.44 DynamicLineSpanner 401
3.1.45 DynamicText 402
3.1.46 DynamicTextSpanner 403
3.1.47 Episema 405
3.1.48 FingerGlideSpanner 406
3.1.49 Fingering 408
3.1.50 FingeringColumn 410
3.1.51 Flag 410
3.1.52 Footnote 411
3.1.53 FretBoard 412
3.1.54 Glissando 414
3.1.55 GraceSpacing 415
3.1.56 GridLine 416
3.1.57 GridPoint 416
3.1.58 Hairpin 417
3.1.59 HorizontalBracket 418
3.1.60 HorizontalBracketText 419
3.1.61 InstrumentName 420
3.1.62 InstrumentSwitch 421
3.1.63 JumpScript 422
3.1.64 KeyCancellation 424
3.1.65 KeySignature 427
3.1.66 KievanLigature 430
3.1.67 LaissezVibrerTie 430
3.1.68 LaissezVibrerTieColumn 431
3.1.69 LedgerLineSpanner 432
3.1.70 LeftEdge 432
3.1.71 LigatureBracket 434
3.1.72 LyricExtender 436
3.1.73 LyricHyphen 436
3.1.74 LyricSpace 437
3.1.75 LyricText 438
3.1.76 MeasureCounter 439
3.1.77 MeasureGrouping 441
3.1.78 MeasureSpanner 442
3.1.79 MelodyItem 443
3.1.80 MensuralLigature 444

3.1.81 MetronomeMark 444
3.1.82 MultiMeasureRest 446
3.1.83 MultiMeasureRestNumber 447
3.1.84 MultiMeasureRestScript 449
3.1.85 MultiMeasureRestText 450
3.1.86 NonMusicalPaperColumn 452
3.1.87 NoteCollision 453
3.1.88 NoteColumn 454
3.1.89 NoteHead 455
3.1.90 NoteName 456
3.1.91 NoteSpacing 456
3.1.92 OttavaBracket 457
3.1.93 PaperColumn 458
3.1.94 Parentheses 459
3.1.95 PercentRepeat 460
3.1.96 PercentRepeatCounter 461
3.1.97 PhrasingSlur 462
3.1.98 PianoPedalBracket 464
3.1.99 RehearsalMark 465
3.1.100 RepeatSlash 467
3.1.101 RepeatTie 467
3.1.102 RepeatTieColumn 469
3.1.103 Rest 469
3.1.104 RestCollision 470
3.1.105 Script 470
3.1.106 ScriptColumn 471
3.1.107 ScriptRow 472
3.1.108 Slur 472
3.1.109 SostenutoPedal 474
3.1.110 SostenutoPedalLineSpanner 475
3.1.111 SpacingSpanner 476
3.1.112 SpanBar 477
3.1.113 SpanBarStub 478
3.1.114 StaffGrouper 479
3.1.115 StaffSpacing 479
3.1.116 StaffSymbol 480
3.1.117 StanzaNumber 481
3.1.118 Stem 481
3.1.119 StemStub 483
3.1.120 StemTremolo 484
3.1.121 StringNumber 485
3.1.122 StrokeFinger 487
3.1.123 SustainPedal 488
3.1.124 SustainPedalLineSpanner 489
3.1.125 System 490
3.1.126 SystemStartBar 491
3.1.127 SystemStartBrace 492
3.1.128 SystemStartBracket 493
3.1.129 SystemStartSquare 494
3.1.130 TabNoteHead 494
3.1.131 TextScript 496
3.1.132 TextSpanner 498
3.1.133 Tie 500
3.1.134 TieColumn 501

3.1.135 TimeSignature 502
3.1.136 TrillPitchAccidental 504
3.1.137 TrillPitchGroup 505
3.1.138 TrillPitchHead 506
3.1.139 TrillPitchParentheses 507
3.1.140 TrillSpanner 507
3.1.141 TupletBracket 509
3.1.142 TupletNumber 510
3.1.143 UnaCordaPedal 511
3.1.144 UnaCordaPedalLineSpanner 512
3.1.145 VaticanaLigature 513
3.1.146 VerticalAlignment 514
3.1.147 VerticalAxisGroup 514
3.1.148 VoiceFollower 516
3.1.149 VoltaBracket 517
3.1.150 VoltaBracketSpanner 518
3.1.151 VowelTransition 520
3.2 Graphical Object Interfaces 521
3.2.1 accidental-interface 521
3.2.2 accidental-placement-interface 521
3.2.3 accidental-suggestion-interface 522
3.2.4 accidental-switch-interface 522
3.2.5 align-interface 522
3.2.6 ambitus-interface 523
3.2.7 arpeggio-interface 523
3.2.8 axis-group-interface 524
3.2.9 balloon-interface 526
3.2.10 bar-line-interface 527
3.2.11 bar-number-interface 528
3.2.12 bass-figure-alignment-interface 528
3.2.13 bass-figure-interface 528
3.2.14 beam-interface 528
3.2.15 bend-after-interface 531
3.2.16 bend-interface 531
3.2.17 bezier-curve-interface 532
3.2.18 break-alignable-interface 533
3.2.19 break-aligned-interface 533
3.2.20 break-alignment-interface 534
3.2.21 breathing-sign-interface 535
3.2.22 centered-bar-number-interface 535
3.2.23 centered-bar-number-line-spanner-interface 536
3.2.24 centered-text-interface 536
3.2.25 chord-name-interface 536
3.2.26 clef-interface 536
3.2.27 clef-modifier-interface 537
3.2.28 cluster-beacon-interface 537
3.2.29 cluster-interface 537
3.2.30 control-point-interface 538
3.2.31 control-polygon-interface 538
3.2.32 custos-interface 538
3.2.33 dot-column-interface 538
3.2.34 dots-interface 539
3.2.35 duration-line-interface 539
3.2.36 dynamic-interface 540

3.2.37 dynamic-line-spanner-interface 540
3.2.38 dynamic-text-interface 540
3.2.39 dynamic-text-spanner-interface 540
3.2.40 enclosing-bracket-interface 540
3.2.41 episema-interface 541
3.2.42 figured-bass-continuation-interface 541
3.2.43 finger-glide-interface 542
3.2.44 finger-interface 543
3.2.45 fingering-column-interface 543
3.2.46 flag-interface 543
3.2.47 font-interface 543
3.2.48 footnote-interface 545
3.2.49 fret-diagram-interface 545
3.2.50 glissando-interface 547
3.2.51 grace-spacing-interface 547
3.2.52 gregorian-ligature-interface 547
3.2.53 grid-line-interface 548
3.2.54 grid-point-interface 548
3.2.55 grob-interface 548
3.2.56 hairpin-interface 552
3.2.57 hara-kiri-group-spanner-interface 553
3.2.58 horizontal-bracket-interface 554
3.2.59 horizontal-bracket-text-interface 554
3.2.60 inline-accidental-interface 555
3.2.61 instrument-specific-markup-interface 555
3.2.62 item-interface 557
3.2.63 jump-script-interface 558
3.2.64 key-cancellation-interface 558
3.2.65 key-signature-interface 558
3.2.66 kievan-ligature-interface 558
3.2.67 ledger-line-spanner-interface 559
3.2.68 ledgered-interface 559
3.2.69 ligature-bracket-interface 559
3.2.70 ligature-head-interface 560
3.2.71 ligature-interface 560
3.2.72 line-interface 560
3.2.73 line-spanner-interface 561
3.2.74 lyric-extender-interface 562
3.2.75 lyric-hyphen-interface 562
3.2.76 lyric-interface 563
3.2.77 lyric-space-interface 563
3.2.78 lyric-syllable-interface 563
3.2.79 mark-interface 563
3.2.80 measure-counter-interface 563
3.2.81 measure-grouping-interface 564
3.2.82 measure-spanner-interface 564
3.2.83 melody-spanner-interface 565
3.2.84 mensural-ligature-interface 566
3.2.85 metronome-mark-interface 566
3.2.86 multi-measure-interface 566
3.2.87 multi-measure-rest-interface 567
3.2.88 multi-measure-rest-number-interface 568
3.2.89 note-collision-interface 568
3.2.90 note-column-interface 568

3.2.91 note-head-interface 569
3.2.92 note-name-interface 570
3.2.93 note-spacing-interface 570
3.2.94 number-interface 570
3.2.95 only-prebreak-interface 571
3.2.96 ottava-bracket-interface 571
3.2.97 outside-staff-axis-group-interface 571
3.2.98 outside-staff-interface 572
3.2.99 paper-column-interface 572
3.2.100 parentheses-interface 574
3.2.101 percent-repeat-interface 574
3.2.102 percent-repeat-item-interface 574
3.2.103 piano-pedal-bracket-interface 575
3.2.104 piano-pedal-interface 575
3.2.105 piano-pedal-script-interface 576
3.2.106 pitched-trill-interface 576
3.2.107 pure-from-neighbor-interface 576
3.2.108 rest-collision-interface 576
3.2.109 rest-interface 576
3.2.110 rhythmic-grob-interface 577
3.2.111 rhythmic-head-interface 577
3.2.112 script-column-interface 577
3.2.113 script-interface 578
3.2.114 self-alignment-interface 579
3.2.115 semi-tie-column-interface 579
3.2.116 semi-tie-interface 580
3.2.117 separation-item-interface 581
3.2.118 side-position-interface 581
3.2.119 slur-interface 583
3.2.120 spaceable-grob-interface 585
3.2.121 spacing-interface 586
3.2.122 spacing-options-interface 586
3.2.123 spacing-spanner-interface 586
3.2.124 span-bar-interface 587
3.2.125 spanner-interface 588
3.2.126 staff-grouper-interface 588
3.2.127 staff-spacing-interface 589
3.2.128 staff-symbol-interface 589
3.2.129 staff-symbol-referencer-interface 590
3.2.130 stanza-number-interface 590
3.2.131 stem-interface 590
3.2.132 stem-tremolo-interface 593
3.2.133 sticky-grob-interface 593
3.2.134 string-number-interface 593
3.2.135 stroke-finger-interface 593
3.2.136 system-interface 594
3.2.137 system-start-delimiter-interface 595
3.2.138 system-start-text-interface 595
3.2.139 tab-note-head-interface 596
3.2.140 text-interface 596
3.2.141 text-script-interface 597
3.2.142 tie-column-interface 597
3.2.143 tie-interface 598
3.2.144 time-signature-interface 600

3.2.145 `trill-pitch-accidental-interface` ... 601
3.2.146 `trill-spanner-interface` ... 601
3.2.147 `tuplet-bracket-interface` ... 601
3.2.148 `tuplet-number-interface` ... 603
3.2.149 `unbreakable-spanner-interface` ... 603
3.2.150 `vaticana-ligature-interface` ... 603
3.2.151 `volta-bracket-interface` ... 604
3.2.152 `volta-interface` ... 604
3.3 User backend properties ... 605
3.4 Internal backend properties ... 627

4 Scheme functions ... 634

Appendix A Indices ... 681
A.1 Concept index ... 681
A.2 Function index ... 681

This is the Internals Reference (IR) for version 2.23.5 of LilyPond, the GNU music typesetter.

1 Music definitions

1.1 Music expressions

1.1.1 AbsoluteDynamicEvent

Create a dynamic mark.

Syntax: *note*\x, where \x is a dynamic mark like \ppp or \sfz. A complete list is in file ly/dynamic-scripts-init.ly.

Event classes: absolute-dynamic-event (page 45), dynamic-event (page 47), music-event (page 49), and StreamEvent (page 53).

Accepted by: Dynamic_engraver (page 297), and Dynamic_performer (page 298).

Properties:

name (symbol):
'AbsoluteDynamicEvent

Name of this music object.

types (list):
```
'(post-event
   event
   dynamic-event
   absolute-dynamic-event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.2 AdHocMarkEvent

Insert markup as a rehearsal mark without advancing the rehearsal mark sequence.

Syntax: \mark *markup*

Example: \mark "A"

Event classes: ad-hoc-mark-event (page 45), mark-event (page 49), music-event (page 49), and StreamEvent (page 53).

Accepted by: Mark_tracking_translator (page 308).

Properties:

name (symbol):
'AdHocMarkEvent

Name of this music object.

types (list):
'(ad-hoc-mark-event mark-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.3 AlternativeEvent

Create an alternative event.

Event classes: alternative-event (page 45), music-event (page 49), and StreamEvent (page 53).

Accepted by: Timing_translator (page 329).

Properties:

`name` (symbol):
`'AlternativeEvent`
Name of this music object.

`types` (list):
`'(event alternative-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.4 AnnotateOutputEvent

Print an annotation of an output element.

Event classes: `annotate-output-event` (page 45), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Balloon_engraver` (page 285).

Properties:

`name` (symbol):
`'AnnotateOutputEvent`
Name of this music object.

`types` (list):
`'(event annotate-output-event post-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.5 ApplyContext

Call the argument with the current context during interpreting phase.

Properties:

`iterator-ctor` (procedure):
`ly:apply-context-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'ApplyContext`
Name of this music object.

`types` (list):
`'(apply-context)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.6 ApplyOutputEvent

Call the argument with all current grobs during interpreting phase.

Syntax: `\applyOutput #'`*`context func`*

Arguments to *func* are 1. the grob, 2. the originating context, and 3. the context where *func* is called.

Event classes: `apply-output-event` (page 45), `layout-instruction-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Output_property_engraver` (page 314).

Properties:

`name` (symbol):
`'ApplyOutputEvent`
Name of this music object.

`types` (list):
`'(event apply-output-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.7 ArpeggioEvent

Make an arpeggio on this note.

Syntax: *note*`-\arpeggio`

Event classes: `arpeggio-event` (page 45), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Arpeggio_engraver` (page 284).

Properties:

`name` (symbol):
`'ArpeggioEvent`
Name of this music object.

`types` (list):
`'(post-event arpeggio-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.8 ArticulationEvent

Add an articulation marking to a note.

Syntax: *note*`xy`, where `x` is a direction (`^` for up or `_` for down), or LilyPond's choice (no direction specified), and where `y` is an articulation (such as `-.`, `->`, `\tenuto`, `\downbow`). See the Notation Reference for details.

Event classes: `articulation-event` (page 45), `music-event` (page 49), `script-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Beat_engraver` (page 288), `Beat_performer` (page 288), `Drum_note_performer` (page 296), `Note_performer` (page 313), and `Script_engraver` (page 320).

Properties:

`name` (symbol):
`'ArticulationEvent`
Name of this music object.

`types` (list):
```
'(post-event
   event
   articulation-event
   script-event)
```
The types of this music object; determines by what engraver this music expression is processed.

1.1.9 BarCheck

Check whether this music coincides with the start of the measure.

Properties:

`iterator-ctor` (procedure):
`ly:bar-check-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'BarCheck`
Name of this music object.

`types` (list):
`'(bar-check)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.10 BassFigureEvent

Print a bass-figure text.

Event classes: `bass-figure-event` (page 46), `music-event` (page 49), `rhythmic-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Figured_bass_engraver` (page 299).

Properties:

`name` (symbol):
`'BassFigureEvent`
Name of this music object.

`types` (list):
`'(event rhythmic-event bass-figure-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.11 BeamEvent

Start or stop a beam.

Syntax for manual control: `c8-[ c c-] c8`

Event classes: `beam-event` (page 46), `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Beam_engraver` (page 287), `Beam_performer` (page 288), and `Grace_beam_engraver` (page 302).

Properties:

`name` (symbol):
`'BeamEvent`
Name of this music object.

`types` (list):
`'(post-event event beam-event span-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.12 BeamForbidEvent

Specify that a note may not auto-beamed.

Event classes: `beam-forbid-event` (page 46), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Auto_beam_engraver` (page 284), and `Grace_auto_beam_engraver` (page 302).

Properties:

`name` (symbol):
`'BeamForbidEvent`

Name of this music object.

`types` (list):
`'(post-event event beam-forbid-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.13 BendAfterEvent

A drop/fall/doit jazz articulation.

Event classes: `bend-after-event` (page 46), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Bend_engraver` (page 289).

Properties:

`name` (symbol):
`'BendAfterEvent`

Name of this music object.

`types` (list):
`'(post-event bend-after-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.14 BendSpanEvent

Used to signal where a bend spanner starts and stops.

Event classes: `bend-span-event` (page 46), `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Bend_spanner_engraver` (page 289).

Properties:

`name` (symbol):
`'BendSpanEvent`

Name of this music object.

`types` (list):
`'(bend-span-event post-event span-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.15 BreakDynamicSpanEvent

End an alignment spanner for dynamics here.

Event classes: break-dynamic-span-event (page 46), break-span-event (page 46), music-event (page 49), and StreamEvent (page 53).

Accepted by: Dynamic_engraver (page 297).

Properties:

name (symbol):
'BreakDynamicSpanEvent
Name of this music object.

types (list):
'(post-event
break-span-event
break-dynamic-span-event
event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.16 BreathingEvent

Create a 'breath mark' or 'comma'.

Syntax: *note*\breathe

Event classes: breathing-event (page 46), music-event (page 49), and StreamEvent (page 53).

Accepted by: Breathing_sign_engraver (page 290), and Note_performer (page 313).

Properties:

midi-length (procedure):
breathe::midi-length
Function to determine how long to play a note in MIDI. It should take a moment (the written length of the note) and a context, and return a moment (the length to play the note).

name (symbol):
'BreathingEvent
Name of this music object.

types (list):
'(event breathing-event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.17 ClusterNoteEvent

A note that is part of a cluster.

Event classes: cluster-note-event (page 46), melodic-event (page 49), music-event (page 49), rhythmic-event (page 51), and StreamEvent (page 53).

Accepted by: Cluster_spanner_engraver (page 292).

Properties:

iterator-ctor (procedure):
ly:rhythmic-music-iterator::constructor
Function to construct a music-event-iterator object for this music.

`name` (symbol):

```
'ClusterNoteEvent
```

Name of this music object.

`types` (list):

```
'(cluster-note-event
   melodic-event
   rhythmic-event
   event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.18 CompletizeExtenderEvent

Used internally to signal the end of a lyrics block to ensure extenders are completed correctly when a `Lyrics` context ends before its associated `Voice` context.

Event classes: `completize-extender-event` (page 47), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Extender_engraver` (page 299).

Properties:

`name` (symbol):

```
'CompletizeExtenderEvent
```

Name of this music object.

`types` (list):

```
'(completize-extender-event event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.19 ContextChange

Change staves in Piano staff.

Syntax: `\change Staff = new-id`

Properties:

`iterator-ctor` (procedure):

```
ly:change-iterator::constructor
```

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):

```
'ContextChange
```

Name of this music object.

`types` (list):

```
'(translator-change-instruction)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.20 ContextSpeccedMusic

Interpret the argument music within a specific context.

Properties:

`iterator-ctor` (procedure):

```
ly:context-specced-music-iterator::constructor
```

Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):

`ly:music-wrapper::length-callback`

How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):

`'ContextSpeccedMusic`

Name of this music object.

`start-callback` (procedure):

`ly:music-wrapper::start-callback`

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):

`'(context-specification music-wrapper-music)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.21 CrescendoEvent

Begin or end a crescendo.

Syntax: *note*`\<` ... *note*`\!`

An alternative syntax is *note*`\cr` ... *note*`\endcr`.

Event classes: `crescendo-event` (page 47), `music-event` (page 49), `span-dynamic-event` (page 52), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Dynamic_engraver` (page 297), and `Dynamic_performer` (page 298).

Properties:

`name` (symbol):

`'CrescendoEvent`

Name of this music object.

`types` (list):

```
'(post-event
   span-event
   span-dynamic-event
   crescendo-event
   event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.22 DecrescendoEvent

Begin or end a decrescendo.

Syntax: *note*`\>` ... *note*`\!`

An alternative syntax is *note*`\decr` ... *note*`\enddecr`.

Event classes: `decrescendo-event` (page 47), `music-event` (page 49), `span-dynamic-event` (page 52), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Dynamic_engraver` (page 297), and `Dynamic_performer` (page 298).

Properties:

`name` (symbol):

`'DecrescendoEvent`

Name of this music object.

`types` (list):

```
'(post-event
  span-event
  span-dynamic-event
  decrescendo-event
  event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.23 DoublePercentEvent

Used internally to signal double percent repeats.

Event classes: `double-percent-event` (page 47), `music-event` (page 49), `rhythmic-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Double_percent_repeat_engraver` (page 296).

Properties:

`name` (symbol):

```
'DoublePercentEvent
```

Name of this music object.

`types` (list):

```
'(event double-percent-event rhythmic-event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.24 DurationLineEvent

Initiate a duration line.

Syntax: *note*`\-`

Event classes: `duration-line-event` (page 47), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Duration_line_engraver` (page 297).

Properties:

`name` (symbol):

```
'DurationLineEvent
```

Name of this music object.

`types` (list):

```
'(duration-line-event post-event event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.25 EpisemaEvent

Begin or end an episema.

Event classes: `episema-event` (page 47), `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Episema_engraver` (page 298).

Properties:

`name` (symbol):

```
'EpisemaEvent
```

Name of this music object.

`types` (list):
`'(post-event span-event event episema-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.26 Event

Atomic music event.

Properties:

`name` (symbol):
`'Event`
Name of this music object.

`types` (list):
`'(event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.27 EventChord

Explicitly entered chords.

When iterated, `elements` are converted to events at the current timestep, followed by any `articulations`. Per-chord postevents attached by the parser just follow any rhythmic events in `elements` instead of utilizing `articulations`.

An unexpanded chord repetition '`q`' is recognizable by having its duration stored in `duration`.

Properties:

`iterator-ctor` (procedure):
`ly:event-chord-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-sequence::event-chord-length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'EventChord`
Name of this music object.

`to-relative-callback` (procedure):
`ly:music-sequence::event-chord-relative-callback`
How to transform a piece of music to relative pitches.

`types` (list):
`'(event-chord simultaneous-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.28 ExtenderEvent

Extend lyrics.

Event classes: `extender-event` (page 47), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Extender_engraver` (page 299).

Properties:

`name` (symbol):
`'ExtenderEvent`

Name of this music object.

`types` (list):
`'(post-event extender-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.29 `FineEvent`

End the performance, not necessarily at the written end of the music.

Event classes: `fine-event` (page 47), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Jump_engraver` (page 305), and `Repeat_acknowledge_engraver` (page 318).

Properties:

`iterator-ctor` (procedure):
`ly:fine-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'FineEvent`

Name of this music object.

`types` (list):
`'(fine-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.30 `FingerGlideEvent`

Initiate a line connecting two equal fingerings. This line represents a finger gliding on a string.

Syntax: *note*`\glide-`*finger*

Event classes: `finger-glide-event` (page 47), `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Not accepted by any engraver or performer.

Properties:

`name` (symbol):
`'FingerGlideEvent`

Name of this music object.

`types` (list):
`'(finger-glide-event post-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.31 FingeringEvent

Specify what finger to use for this note.

Event classes: fingering-event (page 47), music-event (page 49), and StreamEvent (page 53).

Accepted by: Fingering_engraver (page 300), Fretboard_engraver (page 301), and Tab_note_heads_engraver (page 325).

Properties:

name (symbol):
'FingeringEvent

Name of this music object.

types (list):
'(post-event fingering-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.32 FootnoteEvent

Footnote a grob.

Event classes: footnote-event (page 48), music-event (page 49), and StreamEvent (page 53).

Not accepted by any engraver or performer.

Properties:

name (symbol):
'FootnoteEvent

Name of this music object.

types (list):
'(event footnote-event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.33 GlissandoEvent

Start a glissando on this note.

Event classes: glissando-event (page 48), music-event (page 49), and StreamEvent (page 53).

Accepted by: Glissando_engraver (page 302).

Properties:

name (symbol):
'GlissandoEvent

Name of this music object.

types (list):
'(post-event glissando-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.34 GraceMusic

Interpret the argument as grace notes.

Properties:

`iterator-ctor` (procedure):
`ly:grace-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length` (moment):
`#<Mom 0>`
The endpoint of this music. This property is unhappily named in that it does not account for any initial grace notes: the full length of the music is `length` minus the start time. A value of `INF-MOMENT` indicates indefinite length.

`name` (symbol):
`'GraceMusic`
Name of this music object.

`start-callback` (procedure):
`ly:grace-music::start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(grace-music music-wrapper-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.35 HarmonicEvent

Mark a note as harmonic.

Event classes: `harmonic-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Not accepted by any engraver or performer.

Properties:

`name` (symbol):
`'HarmonicEvent`
Name of this music object.

`types` (list):
`'(post-event event harmonic-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.36 HyphenEvent

A hyphen between lyric syllables.

Event classes: `hyphen-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Hyphen_engraver` (page 304).

Properties:

`name` (symbol):
`'HyphenEvent`
Name of this music object.

`types` (list):
`'(post-event hyphen-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.37 KeyChangeEvent

Change the key signature.

Syntax: `\key` *name scale*

Event classes: `key-change-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Key_engraver` (page 305), and `Key_performer` (page 306).

Properties:

`name` (symbol):
`'KeyChangeEvent`
Name of this music object.

`to-relative-callback` (procedure):
`#<procedure #f (x p)>`
How to transform a piece of music to relative pitches.

`types` (list):
`'(key-change-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.38 LabelEvent

Place a bookmarking label.

Event classes: `label-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Paper_column_engraver` (page 315).

Properties:

`name` (symbol):
`'LabelEvent`
Name of this music object.

`types` (list):
`'(label-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.39 LaissezVibrerEvent

Don't damp this chord.

Syntax: *note*`\laissezVibrer`

Event classes: `laissez-vibrer-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Laissez_vibrer_engraver` (page 307).

Properties:

`name` (symbol):
`'LaissezVibrerEvent`

Name of this music object.

`types` (list):
`'(post-event event laissez-vibrer-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.40 LigatureEvent

Start or end a ligature.

Event classes: `ligature-event` (page 48), `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Kievan_ligature_engraver` (page 307), `Ligature_bracket_engraver` (page 307), `Mensural_ligature_engraver` (page 310), and `Vaticana_ligature_engraver` (page 331).

Properties:

`name` (symbol):
`'LigatureEvent`
Name of this music object.

`types` (list):
`'(span-event ligature-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.41 LineBreakEvent

Allow, forbid or force a line break.

Event classes: `break-event` (page 46), `line-break-event` (page 48), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Page_turn_engraver` (page 314), and `Paper_column_engraver` (page 315).

Properties:

`name` (symbol):
`'LineBreakEvent`
Name of this music object.

`types` (list):
`'(line-break-event break-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.42 LyricCombineMusic

Align lyrics to the start of notes.

Syntax: `\lyricsto` *voicename lyrics*

Properties:

`iterator-ctor` (procedure):
`ly:lyric-combine-music-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length` (moment):
`#<Mom infinity>`

The endpoint of this music. This property is unhappily named in that it does not account for any initial grace notes: the full length of the music is `length` minus the start time. A value of `INF-MOMENT` indicates indefinite length.

`name` (symbol):
`'LyricCombineMusic`

Name of this music object.

`types` (list):
`'(lyric-combine-music)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.43 LyricEvent

A lyric syllable. Must be entered in lyrics mode, i.e., `\lyrics { twinkle4 twinkle4 }` .

Event classes: `lyric-event` (page 48), `music-event` (page 49), `rhythmic-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Lyric_engraver` (page 307), and `Lyric_performer` (page 308).

Properties:

`iterator-ctor` (procedure):
`ly:rhythmic-music-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'LyricEvent`

Name of this music object.

`types` (list):
`'(rhythmic-event lyric-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.44 MeasureCounterEvent

Used to signal the start and end of a measure count.

Event classes: `measure-counter-event` (page 49), `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Measure_counter_engraver` (page 309).

Properties:

`name` (symbol):
`'MeasureCounterEvent`

Name of this music object.

`types` (list):
`'(measure-counter-event span-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.45 MeasureSpannerEvent

Used to signal the start and end of a measure spanner.

Event classes: measure-spanner-event (page 49), music-event (page 49), span-event (page 52), and StreamEvent (page 53).

Accepted by: Measure_spanner_engraver (page 309).

Properties:

name (symbol):
'MeasureSpannerEvent
Name of this music object.

types (list):
'(measure-spanner-event span-event event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.46 MultiMeasureArticulationEvent

Articulations on multi-measure rests.

Event classes: multi-measure-articulation-event (page 49), music-event (page 49), and StreamEvent (page 53).

Accepted by: Multi_measure_rest_engraver (page 311).

Properties:

name (symbol):
'MultiMeasureArticulationEvent
Name of this music object.

types (list):
```
'(post-event
   event
   multi-measure-articulation-event)
```
The types of this music object; determines by what engraver this music expression is processed.

1.1.47 MultiMeasureRestEvent

Used internally by MultiMeasureRestMusic to signal rests.

Event classes: multi-measure-rest-event (page 49), music-event (page 49), rhythmic-event (page 51), and StreamEvent (page 53).

Accepted by: Multi_measure_rest_engraver (page 311).

Properties:

iterator-ctor (procedure):
ly:rhythmic-music-iterator::constructor
Function to construct a music-event-iterator object for this music.

name (symbol):
'MultiMeasureRestEvent
Name of this music object.

types (list):
'(event rhythmic-event multi-measure-rest-event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.48 MultiMeasureRestMusic

Rests that may be compressed into multi-measure rests.

Syntax: `R2.*4` for 4 measures in 3/4 time.

Properties:

`elements-callback` (procedure):
`mm-rest-child-list`
Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:sequential-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'MultiMeasureRestMusic`
Name of this music object.

`types` (list):
`'(multi-measure-rest)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.49 MultiMeasureTextEvent

Texts on multi-measure rests.

Syntax: `R-\markup { \roman "bla" }`

Note the explicit font switch.

Event classes: `multi-measure-text-event` (page 49), `music-event` (page 49), and `StreamEvent` (page 53).

Accepted by: `Multi_measure_rest_engraver` (page 311).

Properties:

`name` (symbol):
`'MultiMeasureTextEvent`
Name of this music object.

`types` (list):
`'(post-event event multi-measure-text-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.50 Music

Generic type for music expressions.

Properties:

`name` (symbol):
`'Music`
Name of this music object.

`types` (list):
`'()`
The types of this music object; determines by what engraver this music expression is processed.

1.1.51 NoteEvent

A note.

Outside of chords, any events in `articulations` with a listener are broadcast like chord articulations, the others are retained.

For iteration inside of chords, See Section 1.1.27 [EventChord], page 11.

Event classes: `melodic-event` (page 49), `music-event` (page 49), `note-event` (page 50), `rhythmic-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Beat_engraver` (page 288), `Beat_performer` (page 288), `Bend_spanner_engraver` (page 289), `Chord_name_engraver` (page 290), `Completion_heads_engraver` (page 292), `Drum_note_performer` (page 296), `Drum_notes_engraver` (page 296), `Finger_glide_engraver` (page 300), `Fretboard_engraver` (page 301), `Note_heads_engraver` (page 313), `Note_name_engraver` (page 313), `Note_performer` (page 313), `Part_combine_engraver` (page 315), `Phrasing_slur_engraver` (page 316), `Slur_engraver` (page 322), and `Tab_note_heads_engraver` (page 325).

Properties:

`iterator-ctor` (procedure):
`ly:rhythmic-music-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'NoteEvent`
Name of this music object.

`types` (list):
`'(event note-event rhythmic-event melodic-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.52 NoteGroupingEvent

Start or stop grouping brackets.

Event classes: `music-event` (page 49), `note-grouping-event` (page 50), and `StreamEvent` (page 53).

Accepted by: `Horizontal_bracket_engraver` (page 304).

Properties:

`name` (symbol):
`'NoteGroupingEvent`
Name of this music object.

`types` (list):
`'(post-event event note-grouping-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.53 OttavaEvent

Start or stop an ottava bracket.

Event classes: `music-event` (page 49), `ottava-event` (page 50), and `StreamEvent` (page 53).

Accepted by: `Ottava_spanner_engraver` (page 314).

Properties:

`name` (symbol):
`'OttavaEvent`

Name of this music object.

`types` (list):
`'(ottava-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.54 `OverrideProperty`

Extend the definition of a graphical object.

Syntax: `\override` [*context* `.`] *object property* `=` *value*

Properties:

`iterator-ctor` (procedure):
`ly:push-property-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'OverrideProperty`

Name of this music object.

`types` (list):
```
'(layout-instruction-event
   override-property-event)
```

The types of this music object; determines by what engraver this music expression is processed.

`untransposable` (boolean):
`#t`

If set, this music is not transposed.

1.1.55 `PageBreakEvent`

Allow, forbid or force a page break.

Event classes: `break-event` (page 46), `music-event` (page 49), `page-break-event` (page 50), and `StreamEvent` (page 53).

Accepted by: `Page_turn_engraver` (page 314), and `Paper_column_engraver` (page 315).

Properties:

`name` (symbol):
`'PageBreakEvent`

Name of this music object.

`types` (list):
`'(break-event page-break-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.56 PageTurnEvent

Allow, forbid or force a page turn.

Event classes: `break-event` (page 46), `music-event` (page 49), `page-turn-event` (page 50), and `StreamEvent` (page 53).

Accepted by: `Page_turn_engraver` (page 314), and `Paper_column_engraver` (page 315).

Properties:

`name` (symbol):
`'PageTurnEvent`
Name of this music object.

`types` (list):
`'(break-event page-turn-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.57 PartCombineMusic

Combine two parts on a staff, either merged or as separate voices.

Properties:

`iterator-ctor` (procedure):
`ly:part-combine-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-sequence::maximum-length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'PartCombineMusic`
Name of this music object.

`start-callback` (procedure):
`ly:music-sequence::minimum-start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(part-combine-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.58 PartialSet

Create an anacrusis or upbeat (partial measure).

Properties:

`iterator-ctor` (procedure):
`ly:partial-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-sequence::cumulative-length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'PartialSet`

Name of this music object.

`types` (list):
`'(partial-set)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.59 PercentEvent

Used internally to signal percent repeats.

Event classes: `music-event` (page 49), `percent-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Percent_repeat_engraver` (page 316).

Properties:

`name` (symbol):
`'PercentEvent`

Name of this music object.

`types` (list):
`'(event percent-event rhythmic-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.60 PercentRepeatedMusic

Repeats encoded by percents and slashes.

Properties:

`elements-callback` (procedure):
`make-percent-set`

Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:percent-repeat-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:calculated-sequential-music::length`

How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'PercentRepeatedMusic`

Name of this music object.

`start-callback` (procedure):
`ly:calculated-sequential-music::start`

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(repeated-music percent-repeated-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.61 PesOrFlexaEvent

Within a ligature, mark the previous and the following note to form a pes (if melody goes up) or a flexa (if melody goes down).

Event classes: `music-event` (page 49), `pes-or-flexa-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Vaticana_ligature_engraver` (page 331).

Properties:

`name` (symbol):
`'PesOrFlexaEvent`
Name of this music object.

`types` (list):
`'(pes-or-flexa-event event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.62 PhrasingSlurEvent

Start or end phrasing slur.

Syntax: *note*`\(` and *note*`\)`

Event classes: `music-event` (page 49), `phrasing-slur-event` (page 51), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Phrasing_slur_engraver` (page 316).

Properties:

`name` (symbol):
`'PhrasingSlurEvent`
Name of this music object.

`types` (list):
`'(post-event span-event event phrasing-slur-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.63 PostEvents

Container for several postevents.

This can be used to package several events into a single one. Should not be seen outside of the parser.

Properties:

`name` (symbol):
`'PostEvents`
Name of this music object.

`types` (list):
`'(post-event post-event-wrapper)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.64 PropertySet

Set a context property.

Syntax: `\set` *`context.prop`* `=` *`scheme-val`*

Properties:

`iterator-ctor` (procedure):
`ly:property-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'PropertySet`
Name of this music object.

`types` (list):
`'(layout-instruction-event)`
The types of this music object; determines by what engraver this music expression is processed.

`untransposable` (boolean):
`#t`
If set, this music is not transposed.

1.1.65 PropertyUnset

Restore the default setting for a context property. See Section 1.1.64 [PropertySet], page 25.

Syntax: `\unset` *`context.prop`*

Properties:

`iterator-ctor` (procedure):
`ly:property-unset-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'PropertyUnset`
Name of this music object.

`types` (list):
`'(layout-instruction-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.66 QuoteMusic

Quote preprocessed snippets of music.

Properties:

`iterator-ctor` (procedure):
`ly:music-wrapper-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-wrapper::length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'QuoteMusic`

Name of this music object.

`start-callback` (procedure):
`ly:music-wrapper::start-callback`

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(music-wrapper-music)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.67 RehearsalMarkEvent

Insert a rehearsal mark.

Syntax: `\mark` *marker*

Example: `\mark "A"`

Event classes: `mark-event` (page 49), `music-event` (page 49), `rehearsal-mark-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Mark_tracking_translator` (page 308).

Properties:

`name` (symbol):
`'RehearsalMarkEvent`

Name of this music object.

`types` (list):
`'(rehearsal-mark-event mark-event event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.68 RelativeOctaveCheck

Check if a pitch is in the correct octave.

Properties:

`name` (symbol):
`'RelativeOctaveCheck`

Name of this music object.

`to-relative-callback` (procedure):
`ly:relative-octave-check::relative-callback`

How to transform a piece of music to relative pitches.

`types` (list):
`'(relative-octave-check)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.69 RelativeOctaveMusic

Music in which the assignment of octaves is complete.

Properties:

`iterator-ctor` (procedure):
`ly:music-wrapper-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-wrapper::length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'RelativeOctaveMusic`
Name of this music object.

`start-callback` (procedure):
`ly:music-wrapper::start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`to-relative-callback` (procedure):
`ly:relative-octave-music::relative-callback`
How to transform a piece of music to relative pitches.

`types` (list):
`'(music-wrapper-music relative-octave-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.70 RepeatSlashEvent

Used internally to signal beat repeats.

Event classes: `music-event` (page 49), `repeat-slash-event` (page 51), `rhythmic-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Slash_repeat_engraver` (page 321).

Properties:

`name` (symbol):
`'RepeatSlashEvent`
Name of this music object.

`types` (list):
`'(event repeat-slash-event rhythmic-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.71 RepeatTieEvent

Ties for starting a second volta bracket.

Event classes: `music-event` (page 49), `repeat-tie-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Repeat_tie_engraver` (page 319).

Properties:

`name` (symbol):
`'RepeatTieEvent`

Name of this music object.

`types` (list):
`'(post-event event repeat-tie-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.72 `RestEvent`

A Rest.

Syntax: `r4` for a quarter rest.

Event classes: `music-event` (page 49), `rest-event` (page 51), `rhythmic-event` (page 51), and `StreamEvent` (page 53).

Accepted by: `Chord_name_engraver` (page 290), `Completion_rest_engraver` (page 293), `Figured_bass_engraver` (page 299), and `Rest_engraver` (page 320).

Properties:

`iterator-ctor` (procedure):
`ly:rhythmic-music-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'RestEvent`

Name of this music object.

`types` (list):
`'(event rhythmic-event rest-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.73 `RevertProperty`

The opposite of Section 1.1.54 [`OverrideProperty`], page 21: remove a previously added property from a graphical object definition.

Properties:

`iterator-ctor` (procedure):
`ly:pop-property-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'RevertProperty`

Name of this music object.

`types` (list):
`'(layout-instruction-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.74 ScriptEvent

Add an articulation mark to a note.

Event classes: music-event (page 49), script-event (page 51), and StreamEvent (page 53).

Not accepted by any engraver or performer.

Properties:

name (symbol):
'ScriptEvent

Name of this music object.

types (list):
'(event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.75 SectionEvent

Add a section division, which is typically written as a thin double bar line.

Event classes: music-event (page 49), section-event (page 52), and StreamEvent (page 53).

Accepted by: Repeat_acknowledge_engraver (page 318).

Properties:

name (symbol):
'SectionEvent

Name of this music object.

types (list):
'(section-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.76 SegnoEvent

Add a segno mark or bar line.

Event classes: music-event (page 49), segno-event (page 52), and StreamEvent (page 53).

Accepted by: Repeat_acknowledge_engraver (page 318).

Properties:

name (symbol):
'SegnoEvent

Name of this music object.

types (list):
'(segno-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.77 SequentialAlternativeMusic

Repeat alternatives in sequence.

Syntax: `\alternative { alternatives }`

Properties:

`elements-callback` (procedure):
`#<procedure #f (m)>`
Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:alternative-sequence-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-sequence::cumulative-length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'SequentialAlternativeMusic`
Name of this music object.

`start-callback` (procedure):
`ly:music-sequence::first-start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(sequential-music sequential-alternative-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.78 SequentialMusic

Music expressions concatenated.

Syntax: `\sequential { ... }` or simply `{ ... }`

Properties:

`elements-callback` (procedure):
`#<procedure #f (m)>`
Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:sequential-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-sequence::cumulative-length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'SequentialMusic`
Name of this music object.

`start-callback` (procedure):
`ly:music-sequence::first-start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(sequential-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.79 SimultaneousMusic

Music playing together.

Syntax: `\simultaneous { ... }` or `<< ... >>`

Properties:

`iterator-ctor` (procedure):
`ly:simultaneous-music-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-sequence::maximum-length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'SimultaneousMusic`
Name of this music object.

`start-callback` (procedure):
`ly:music-sequence::minimum-start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`to-relative-callback` (procedure):
`ly:music-sequence::simultaneous-relative-callback`
How to transform a piece of music to relative pitches.

`types` (list):
`'(simultaneous-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.80 SkipEvent

Filler that takes up duration, but does not print anything.

Syntax: `s4` for a skip equivalent to a quarter rest.

Event classes: `music-event` (page 49), `rhythmic-event` (page 51), `skip-event` (page 52), and `StreamEvent` (page 53).

Not accepted by any engraver or performer.

Properties:

`iterator-ctor` (procedure):
`ly:rhythmic-music-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'SkipEvent`
Name of this music object.

`types` (list):
`'(event rhythmic-event skip-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.81 SkipMusic

Filler that takes up duration, does not print anything, and also does not create staves or voices implicitly.

Syntax: `\skip` *duration*

Properties:

`iterator-ctor` (procedure):
`ly:simple-music-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'SkipMusic`
Name of this music object.

`types` (list):
`'(event skip-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.82 SlurEvent

Start or end slur.

Syntax: *note*`(` and *note*`)`

Event classes: `music-event` (page 49), `slur-event` (page 52), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Slur_engraver` (page 322), and `Slur_performer` (page 322).

Properties:

`name` (symbol):
`'SlurEvent`
Name of this music object.

`types` (list):
`'(post-event span-event event slur-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.83 SoloOneEvent

Print ‘Solo 1’.

Event classes: `music-event` (page 49), `part-combine-event` (page 50), `solo-one-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Part_combine_engraver` (page 315).

Properties:

`name` (symbol):
`'SoloOneEvent`

Name of this music object.

`part-combine-status` (symbol):
`'solo1`

Change to what kind of state? Options are `solo1`, `solo2` and `unisono`.

`types` (list):
`'(event part-combine-event solo-one-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.84 SoloTwoEvent

Print 'Solo 2'.

Event classes: `music-event` (page 49), `part-combine-event` (page 50), `solo-two-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Part_combine_engraver` (page 315).

Properties:

`name` (symbol):
`'SoloTwoEvent`

Name of this music object.

`part-combine-status` (symbol):
`'solo2`

Change to what kind of state? Options are `solo1`, `solo2` and `unisono`.

`types` (list):
`'(event part-combine-event solo-two-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.85 SostenutoEvent

Depress or release sostenuto pedal.

Event classes: `music-event` (page 49), `pedal-event` (page 51), `sostenuto-event` (page 52), `span-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Piano_pedal_engraver` (page 317), and `Piano_pedal_performer` (page 317).

Properties:

`name` (symbol):
`'SostenutoEvent`

Name of this music object.

`types` (list):
`'(post-event event pedal-event sostenuto-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.86 SpacingSectionEvent

Start a new spacing section.

Event classes: `music-event` (page 49), `spacing-section-event` (page 52), and `StreamEvent` (page 53).

Accepted by: `Spacing_engraver` (page 322).

Properties:

`name` (symbol):
`'SpacingSectionEvent`

Name of this music object.

`types` (list):
`'(event spacing-section-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.87 SpanEvent

Event for anything that is started at a different time than stopped.

Event classes: `music-event` (page 49), `span-event` (page 52), and `StreamEvent` (page 53).

Not accepted by any engraver or performer.

Properties:

`name` (symbol):
`'SpanEvent`

Name of this music object.

`types` (list):
`'(event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.88 StaffSpanEvent

Start or stop a staff symbol.

Event classes: `music-event` (page 49), `span-event` (page 52), `staff-span-event` (page 53), and `StreamEvent` (page 53).

Accepted by: `Staff_symbol_engraver` (page 324).

Properties:

`name` (symbol):
`'StaffSpanEvent`

Name of this music object.

`types` (list):
`'(event span-event staff-span-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.89 StringNumberEvent

Specify on which string to play this note.

Syntax: *\number*

Event classes: music-event (page 49), StreamEvent (page 53), and string-number-event (page 53).

Accepted by: Bend_spanner_engraver (page 289), Fretboard_engraver (page 301), and Tab_note_heads_engraver (page 325).

Properties:

name (symbol):
'StringNumberEvent
Name of this music object.

types (list):
'(post-event string-number-event event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.90 StrokeFingerEvent

Specify with which finger to pluck a string.

Syntax: \rightHandFinger *text*

Event classes: music-event (page 49), StreamEvent (page 53), and stroke-finger-event (page 53).

Not accepted by any engraver or performer.

Properties:

name (symbol):
'StrokeFingerEvent
Name of this music object.

types (list):
'(post-event stroke-finger-event event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.91 SustainEvent

Depress or release sustain pedal.

Event classes: music-event (page 49), pedal-event (page 51), span-event (page 52), StreamEvent (page 53), and sustain-event (page 53).

Accepted by: Piano_pedal_engraver (page 317), and Piano_pedal_performer (page 317).

Properties:

name (symbol):
'SustainEvent
Name of this music object.

types (list):
'(post-event event pedal-event sustain-event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.92 TempoChangeEvent

A metronome mark or tempo indication.

Event classes: music-event (page 49), StreamEvent (page 53), and tempo-change-event (page 54).

Accepted by: Metronome_mark_engraver (page 310).

Properties:

name (symbol):
'TempoChangeEvent

Name of this music object.

types (list):
'(event tempo-change-event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.93 TextScriptEvent

Print text.

Event classes: music-event (page 49), script-event (page 51), StreamEvent (page 53), and text-script-event (page 54).

Accepted by: Text_engraver (page 327).

Properties:

name (symbol):
'TextScriptEvent

Name of this music object.

types (list):
'(post-event script-event text-script-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.94 TextSpanEvent

Start a text spanner, for example, an octavation.

Event classes: music-event (page 49), span-event (page 52), StreamEvent (page 53), and text-span-event (page 54).

Accepted by: Text_spanner_engraver (page 327).

Properties:

name (symbol):
'TextSpanEvent

Name of this music object.

types (list):
'(post-event span-event event text-span-event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.95 TieEvent

A tie.

Syntax: *note*-~

Event classes: music-event (page 49), StreamEvent (page 53), and tie-event (page 54).

Accepted by: Drum_note_performer (page 296), Note_performer (page 313), Tie_engraver (page 327), and Tie_performer (page 328).

Properties:

name (symbol):
'TieEvent

Name of this music object.

types (list):
'(post-event tie-event event)

The types of this music object; determines by what engraver this music expression is processed.

1.1.96 TimeScaledMusic

Multiply durations, as in tuplets.

Syntax: \times *fraction music*, e.g., \times 2/3 { ... } for triplets.

Properties:

iterator-ctor (procedure):
ly:tuplet-iterator::constructor

Function to construct a music-event-iterator object for this music.

length-callback (procedure):
ly:music-wrapper::length-callback

How to compute the duration of this music. This property can only be defined as initializer in scm/define-music-types.scm.

name (symbol):
'TimeScaledMusic

Name of this music object.

start-callback (procedure):
ly:music-wrapper::start-callback

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in scm/define-music-types.scm.

types (list):
'(time-scaled-music)

The types of this music object; determines by what engraver this music expression is processed.

1.1.97 TimeSignatureEvent

An event created when setting a new time signature

Event classes: music-event (page 49), StreamEvent (page 53), and time-signature-event (page 54).

Accepted by: Time_signature_engraver (page 328), and Time_signature_performer (page 328).

Properties:

`name` (symbol):
`'TimeSignatureEvent`

Name of this music object.

`types` (list):
`'(event time-signature-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.98 TimeSignatureMusic

Set a new time signature

Properties:

`elements-callback` (procedure):
`make-time-signature-set`

Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:sequential-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`name` (symbol):
`'TimeSignatureMusic`

Name of this music object.

`types` (list):
`'(time-signature-music)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.99 TransposedMusic

Music that has been transposed.

Properties:

`iterator-ctor` (procedure):
`ly:music-wrapper-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-wrapper::length-callback`

How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'TransposedMusic`

Name of this music object.

`start-callback` (procedure):
`ly:music-wrapper::start-callback`

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`to-relative-callback` (procedure):
`ly:relative-octave-music::no-relative-callback`

How to transform a piece of music to relative pitches.

`types` (list):
`'(music-wrapper-music transposed-music)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.100 TremoloEvent

Unmeasured tremolo.

Event classes: `music-event` (page 49), `StreamEvent` (page 53), and `tremolo-event` (page 54).

Accepted by: `Stem_engraver` (page 324).

Properties:

`name` (symbol):
`'TremoloEvent`

Name of this music object.

`types` (list):
`'(post-event event tremolo-event)`

The types of this music object; determines by what engraver this music expression is processed.

1.1.101 TremoloRepeatedMusic

Repeated notes denoted by tremolo beams.

Properties:

`elements-callback` (procedure):
`make-tremolo-set`

Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:sequential-iterator::constructor`

Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:calculated-sequential-music::length`

How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'TremoloRepeatedMusic`

Name of this music object.

`start-callback` (procedure):
`ly:calculated-sequential-music::start`

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):
`'(repeated-music tremolo-repeated-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.102 TremoloSpanEvent

Tremolo over two stems.

Event classes: `music-event` (page 49), `span-event` (page 52), `StreamEvent` (page 53), and `tremolo-span-event` (page 54).

Accepted by: `Chord_tremolo_engraver` (page 291).

Properties:

`name` (symbol):
`'TremoloSpanEvent`
Name of this music object.

`types` (list):
`'(event span-event tremolo-span-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.103 TrillSpanEvent

Start a trill spanner.

Event classes: `music-event` (page 49), `span-event` (page 52), `StreamEvent` (page 53), and `trill-span-event` (page 54).

Accepted by: `Trill_spanner_engraver` (page 330).

Properties:

`name` (symbol):
`'TrillSpanEvent`
Name of this music object.

`types` (list):
`'(post-event span-event event trill-span-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.104 TupletSpanEvent

Used internally to signal where tuplet brackets start and stop.

Event classes: `music-event` (page 49), `span-event` (page 52), `StreamEvent` (page 53), and `tuplet-span-event` (page 54).

Accepted by: `Stem_engraver` (page 324), and `Tuplet_engraver` (page 330).

Properties:

`name` (symbol):
`'TupletSpanEvent`
Name of this music object.

`types` (list):
`'(tuplet-span-event span-event event post-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.105 UnaCordaEvent

Depress or release una-corda pedal.

Event classes: music-event (page 49), pedal-event (page 51), span-event (page 52), StreamEvent (page 53), and una-corda-event (page 54).

Accepted by: Piano_pedal_engraver (page 317), and Piano_pedal_performer (page 317).

Properties:

name (symbol):
'UnaCordaEvent
Name of this music object.

types (list):
'(post-event event pedal-event una-corda-event)
The types of this music object; determines by what engraver this music expression is processed.

1.1.106 UnfoldedRepeatedMusic

Repeated music which is fully written (and played) out.

Properties:

elements-callback (procedure):
make-unfolded-set
Return a list of children, for use by a sequential iterator. Takes a single music parameter.

iterator-ctor (procedure):
ly:sequential-iterator::constructor
Function to construct a music-event-iterator object for this music.

length-callback (procedure):
ly:calculated-sequential-music::length
How to compute the duration of this music. This property can only be defined as initializer in scm/define-music-types.scm.

name (symbol):
'UnfoldedRepeatedMusic
Name of this music object.

start-callback (procedure):
ly:calculated-sequential-music::start
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in scm/define-music-types.scm.

types (list):
'(repeated-music unfolded-repeated-music)
The types of this music object; determines by what engraver this music expression is processed.

1.1.107 UnfoldedSpeccedMusic

Music that appears once repeated music is unfolded.

Properties:

iterator-ctor (procedure):
ly:music-iterator::constructor

Function to construct a `music-event-iterator` object for this music.

`length` (moment):
`#<Mom 0>`
The endpoint of this music. This property is unhappily named in that it does not account for any initial grace notes: the full length of the music is `length` minus the start time. A value of `INF-MOMENT` indicates indefinite length.

`name` (symbol):
`'UnfoldedSpeccedMusic`
Name of this music object.

`types` (list):
`'(unfolded-specification music-wrapper-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.108 UnisonoEvent

Print 'a 2'.

Event classes: `music-event` (page 49), `part-combine-event` (page 50), `StreamEvent` (page 53), and `unisono-event` (page 54).

Accepted by: `Part_combine_engraver` (page 315).

Properties:

`name` (symbol):
`'UnisonoEvent`
Name of this music object.

`part-combine-status` (symbol):
`'unisono`
Change to what kind of state? Options are `solo1`, `solo2` and `unisono`.

`types` (list):
`'(event part-combine-event unisono-event)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.109 UnrelativableMusic

Music that cannot be converted from relative to absolute notation. For example, transposed music.

Properties:

`iterator-ctor` (procedure):
`ly:music-wrapper-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:music-wrapper::length-callback`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'UnrelativableMusic`
Name of this music object.

`start-callback` (procedure):
`ly:music-wrapper::start-callback`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`to-relative-callback` (procedure):
`ly:relative-octave-music::no-relative-callback`
How to transform a piece of music to relative pitches.

`types` (list):
`'(music-wrapper-music unrelativable-music)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.110 `VoiceSeparator`

Separate polyphonic voices in simultaneous music.

Syntax: `\\`

Properties:

`name` (symbol):
`'VoiceSeparator`
Name of this music object.

`types` (list):
`'(separator)`
The types of this music object; determines by what engraver this music expression is processed.

1.1.111 `VoltaRepeatedMusic`

Repeats with alternatives placed sequentially.

Properties:

`elements-callback` (procedure):
`make-volta-set`
Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`iterator-ctor` (procedure):
`ly:volta-repeat-iterator::constructor`
Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):
`ly:calculated-sequential-music::length`
How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):
`'VoltaRepeatedMusic`
Name of this music object.

`start-callback` (procedure):
`ly:calculated-sequential-music::start`
Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):

```
'(volta-repeated-music
  folded-repeated-music
  repeated-music)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.112 VoltaSpanEvent

Used internally to signal where volta brackets start and stop.

Event classes: `music-event` (page 49), `span-event` (page 52), `StreamEvent` (page 53), and `volta-span-event` (page 55).

Accepted by: `Repeat_acknowledge_engraver` (page 318), and `Volta_engraver` (page 331).

Properties:

`name` (symbol):

```
'VoltaSpanEvent
```

Name of this music object.

`types` (list):

```
'(volta-span-event span-event event post-event)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.113 VoltaSpeccedMusic

Music for a specific volta within repeated music.

Properties:

`iterator-ctor` (procedure):

```
ly:volta-specced-music-iterator::constructor
```

Function to construct a `music-event-iterator` object for this music.

`length-callback` (procedure):

```
ly:music-wrapper::length-callback
```

How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`name` (symbol):

```
'VoltaSpeccedMusic
```

Name of this music object.

`start-callback` (procedure):

```
ly:music-wrapper::start-callback
```

Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`types` (list):

```
'(volta-specification music-wrapper-music)
```

The types of this music object; determines by what engraver this music expression is processed.

1.1.114 VowelTransitionEvent

A vowel transition between lyric syllables.

Event classes: music-event (page 49), StreamEvent (page 53), and vowel-transition-event (page 55).

Accepted by: Hyphen_engraver (page 304).

Properties:

name (symbol):
'VowelTransitionEvent
Name of this music object.

types (list):
'(post-event vowel-transition-event event)
The types of this music object; determines by what engraver this music expression is processed.

1.2 Music classes

1.2.1 absolute-dynamic-event

Music event type absolute-dynamic-event is in music objects of type AbsoluteDynamicEvent (page 2).

Accepted by: Dynamic_engraver (page 297), and Dynamic_performer (page 298).

1.2.2 ad-hoc-mark-event

Music event type ad-hoc-mark-event is in music objects of type AdHocMarkEvent (page 2).

Accepted by: Mark_tracking_translator (page 308).

1.2.3 alternative-event

Music event type alternative-event is in music objects of type AlternativeEvent (page 2).

Accepted by: Timing_translator (page 329).

1.2.4 annotate-output-event

Music event type annotate-output-event is in music objects of type AnnotateOutputEvent (page 3).

Accepted by: Balloon_engraver (page 285).

1.2.5 apply-output-event

Music event type apply-output-event is in music objects of type ApplyOutputEvent (page 3).

Accepted by: Output_property_engraver (page 314).

1.2.6 arpeggio-event

Music event type arpeggio-event is in music objects of type ArpeggioEvent (page 4).

Accepted by: Arpeggio_engraver (page 284).

1.2.7 articulation-event

Music event type articulation-event is in music objects of type ArticulationEvent (page 4).

Accepted by: Beat_engraver (page 288), Beat_performer (page 288), Drum_note_performer (page 296), Note_performer (page 313), and Script_engraver (page 320).

1.2.8 bass-figure-event

Music event type `bass-figure-event` is in music objects of type `BassFigureEvent` (page 5).

Accepted by: `Figured_bass_engraver` (page 299).

1.2.9 beam-event

Music event type `beam-event` is in music objects of type `BeamEvent` (page 5).

Accepted by: `Beam_engraver` (page 287), `Beam_performer` (page 288), and `Grace_beam_engraver` (page 302).

1.2.10 beam-forbid-event

Music event type `beam-forbid-event` is in music objects of type `BeamForbidEvent` (page 6).

Accepted by: `Auto_beam_engraver` (page 284), and `Grace_auto_beam_engraver` (page 302).

1.2.11 bend-after-event

Music event type `bend-after-event` is in music objects of type `BendAfterEvent` (page 6).

Accepted by: `Bend_engraver` (page 289).

1.2.12 bend-span-event

Music event type `bend-span-event` is in music objects of type `BendSpanEvent` (page 6).

Accepted by: `Bend_spanner_engraver` (page 289).

1.2.13 break-dynamic-span-event

Music event type `break-dynamic-span-event` is in music objects of type `BreakDynamicSpanEvent` (page 7).

Not accepted by any engraver or performer.

1.2.14 break-event

Music event type `break-event` is in music objects of type `LineBreakEvent` (page 16), `PageBreakEvent` (page 21), and `PageTurnEvent` (page 22).

Accepted by: `Page_turn_engraver` (page 314), and `Paper_column_engraver` (page 315).

1.2.15 break-span-event

Music event type `break-span-event` is in music objects of type `BreakDynamicSpanEvent` (page 7).

Accepted by: `Dynamic_engraver` (page 297).

1.2.16 breathing-event

Music event type `breathing-event` is in music objects of type `BreathingEvent` (page 7).

Accepted by: `Breathing_sign_engraver` (page 290), and `Note_performer` (page 313).

1.2.17 cluster-note-event

Music event type `cluster-note-event` is in music objects of type `ClusterNoteEvent` (page 7).

Accepted by: `Cluster_spanner_engraver` (page 292).

1.2.18 completize-extender-event

Music event type `completize-extender-event` is in music objects of type `CompletizeExtenderEvent` (page 8).

Accepted by: `Extender_engraver` (page 299).

1.2.19 crescendo-event

Music event type `crescendo-event` is in music objects of type `CrescendoEvent` (page 9).

Accepted by: `Dynamic_performer` (page 298).

1.2.20 decrescendo-event

Music event type `decrescendo-event` is in music objects of type `DecrescendoEvent` (page 9).

Accepted by: `Dynamic_performer` (page 298).

1.2.21 double-percent-event

Music event type `double-percent-event` is in music objects of type `DoublePercentEvent` (page 10).

Accepted by: `Double_percent_repeat_engraver` (page 296).

1.2.22 duration-line-event

Music event type `duration-line-event` is in music objects of type `DurationLineEvent` (page 10).

Accepted by: `Duration_line_engraver` (page 297).

1.2.23 dynamic-event

Music event type `dynamic-event` is in music objects of type `AbsoluteDynamicEvent` (page 2).

Not accepted by any engraver or performer.

1.2.24 episema-event

Music event type `episema-event` is in music objects of type `EpisemaEvent` (page 10).

Accepted by: `Episema_engraver` (page 298).

1.2.25 extender-event

Music event type `extender-event` is in music objects of type `ExtenderEvent` (page 11).

Accepted by: `Extender_engraver` (page 299).

1.2.26 fine-event

Music event type `fine-event` is in music objects of type `FineEvent` (page 12).

Accepted by: `Jump_engraver` (page 305), and `Repeat_acknowledge_engraver` (page 318).

1.2.27 finger-glide-event

Music event type `finger-glide-event` is in music objects of type `FingerGlideEvent` (page 12).

Not accepted by any engraver or performer.

1.2.28 fingering-event

Music event type `fingering-event` is in music objects of type `FingeringEvent` (page 13).

Accepted by: `Fingering_engraver` (page 300), `Fretboard_engraver` (page 301), and `Tab_note_heads_engraver` (page 325).

1.2.29 footnote-event

Music event type `footnote-event` is in music objects of type `FootnoteEvent` (page 13).

Not accepted by any engraver or performer.

1.2.30 glissando-event

Music event type `glissando-event` is in music objects of type `GlissandoEvent` (page 13).

Accepted by: `Glissando_engraver` (page 302).

1.2.31 harmonic-event

Music event type `harmonic-event` is in music objects of type `HarmonicEvent` (page 14).

Not accepted by any engraver or performer.

1.2.32 hyphen-event

Music event type `hyphen-event` is in music objects of type `HyphenEvent` (page 14).

Accepted by: `Hyphen_engraver` (page 304).

1.2.33 key-change-event

Music event type `key-change-event` is in music objects of type `KeyChangeEvent` (page 15).

Accepted by: `Key_engraver` (page 305), and `Key_performer` (page 306).

1.2.34 label-event

Music event type `label-event` is in music objects of type `LabelEvent` (page 15).

Accepted by: `Paper_column_engraver` (page 315).

1.2.35 laissez-vibrer-event

Music event type `laissez-vibrer-event` is in music objects of type `LaissezVibrerEvent` (page 15).

Accepted by: `Laissez_vibrer_engraver` (page 307).

1.2.36 layout-instruction-event

Music event type `layout-instruction-event` is in music objects of type `ApplyOutputEvent` (page 3).

Not accepted by any engraver or performer.

1.2.37 ligature-event

Music event type `ligature-event` is in music objects of type `LigatureEvent` (page 16).

Accepted by: `Kievan_ligature_engraver` (page 307), `Ligature_bracket_engraver` (page 307), `Mensural_ligature_engraver` (page 310), and `Vaticana_ligature_engraver` (page 331).

1.2.38 line-break-event

Music event type `line-break-event` is in music objects of type `LineBreakEvent` (page 16).

Not accepted by any engraver or performer.

1.2.39 lyric-event

Music event type `lyric-event` is in music objects of type `LyricEvent` (page 17).

Accepted by: `Lyric_engraver` (page 307), and `Lyric_performer` (page 308).

1.2.40 mark-event

Music event type `mark-event` is in music objects of type `AdHocMarkEvent` (page 2), and `RehearsalMarkEvent` (page 26).

Not accepted by any engraver or performer.

1.2.41 measure-counter-event

Music event type `measure-counter-event` is in music objects of type `MeasureCounterEvent` (page 17).

Accepted by: `Measure_counter_engraver` (page 309).

1.2.42 measure-spanner-event

Music event type `measure-spanner-event` is in music objects of type `MeasureSpannerEvent` (page 18).

Accepted by: `Measure_spanner_engraver` (page 309).

1.2.43 melodic-event

Music event type `melodic-event` is in music objects of type `ClusterNoteEvent` (page 7), and `NoteEvent` (page 20).

Not accepted by any engraver or performer.

1.2.44 multi-measure-articulation-event

Music event type `multi-measure-articulation-event` is in music objects of type `MultiMeasureArticulationEvent` (page 18).

Accepted by: `Multi_measure_rest_engraver` (page 311).

1.2.45 multi-measure-rest-event

Music event type `multi-measure-rest-event` is in music objects of type `MultiMeasureRestEvent` (page 18).

Accepted by: `Multi_measure_rest_engraver` (page 311).

1.2.46 multi-measure-text-event

Music event type `multi-measure-text-event` is in music objects of type `MultiMeasureTextEvent` (page 19).

Accepted by: `Multi_measure_rest_engraver` (page 311).

1.2.47 music-event

Music event type `music-event` is in music objects of type `AbsoluteDynamicEvent` (page 2), `AdHocMarkEvent` (page 2), `AlternativeEvent` (page 2), `AnnotateOutputEvent` (page 3), `ApplyOutputEvent` (page 3), `ArpeggioEvent` (page 4), `ArticulationEvent` (page 4), `BassFigureEvent` (page 5), `BeamEvent` (page 5), `BeamForbidEvent` (page 6), `BendAfterEvent` (page 6), `BendSpanEvent` (page 6), `BreakDynamicSpanEvent` (page 7), `BreathingEvent` (page 7), `ClusterNoteEvent` (page 7), `CompletizeExtenderEvent` (page 8), `CrescendoEvent` (page 9), `DecrescendoEvent` (page 9), `DoublePercentEvent` (page 10), `DurationLineEvent` (page 10), `EpisemaEvent` (page 10), `ExtenderEvent` (page 11), `FineEvent` (page 12), `FingerGlideEvent` (page 12), `FingeringEvent` (page 13), `FootnoteEvent` (page 13), `GlissandoEvent` (page 13), `HarmonicEvent` (page 14), `HyphenEvent` (page 14), `KeyChangeEvent` (page 15), `LabelEvent` (page 15), `LaissezVibrerEvent` (page 15), `LigatureEvent` (page 16), `LineBreakEvent` (page 16), `LyricEvent` (page 17), `MeasureCounterEvent` (page 17), `MeasureSpannerEvent` (page 18),

MultiMeasureArticulationEvent (page 18), MultiMeasureRestEvent (page 18), MultiMeasureTextEvent (page 19), NoteEvent (page 20), NoteGroupingEvent (page 20), OttavaEvent (page 20), PageBreakEvent (page 21), PageTurnEvent (page 22), PercentEvent (page 23), PesOrFlexaEvent (page 24), PhrasingSlurEvent (page 24), RehearsalMarkEvent (page 26), RepeatSlashEvent (page 27), RepeatTieEvent (page 27), RestEvent (page 28), ScriptEvent (page 29), SectionEvent (page 29), SegnoEvent (page 29), SkipEvent (page 31), SlurEvent (page 32), SoloOneEvent (page 32), SoloTwoEvent (page 33), SostenutoEvent (page 33), SpacingSectionEvent (page 34), SpanEvent (page 34), StaffSpanEvent (page 34), StringNumberEvent (page 35), StrokeFingerEvent (page 35), SustainEvent (page 35), TempoChangeEvent (page 36), TextScriptEvent (page 36), TextSpanEvent (page 36), TieEvent (page 37), TimeSignatureEvent (page 37), TremoloEvent (page 39), TremoloSpanEvent (page 40), TrillSpanEvent (page 40), TupletSpanEvent (page 40), UnaCordaEvent (page 41), UnisonoEvent (page 42), VoltaSpanEvent (page 44), and VowelTransitionEvent (page 45).

Not accepted by any engraver or performer.

1.2.48 note-event

Music event type note-event is in music objects of type NoteEvent (page 20).

Accepted by: Beat_engraver (page 288), Beat_performer (page 288), Bend_spanner_engraver (page 289), Chord_name_engraver (page 290), Completion_heads_engraver (page 292), Drum_note_performer (page 296), Drum_notes_engraver (page 296), Finger_glide_engraver (page 300), Fretboard_engraver (page 301), Note_heads_engraver (page 313), Note_name_engraver (page 313), Note_performer (page 313), Part_combine_engraver (page 315), Phrasing_slur_engraver (page 316), Slur_engraver (page 322), and Tab_note_heads_engraver (page 325).

1.2.49 note-grouping-event

Music event type note-grouping-event is in music objects of type NoteGroupingEvent (page 20).

Accepted by: Horizontal_bracket_engraver (page 304).

1.2.50 ottava-event

Music event type ottava-event is in music objects of type OttavaEvent (page 20).

Accepted by: Ottava_spanner_engraver (page 314).

1.2.51 page-break-event

Music event type page-break-event is in music objects of type PageBreakEvent (page 21).

Not accepted by any engraver or performer.

1.2.52 page-turn-event

Music event type page-turn-event is in music objects of type PageTurnEvent (page 22).

Not accepted by any engraver or performer.

1.2.53 part-combine-event

Music event type part-combine-event is in music objects of type SoloOneEvent (page 32), SoloTwoEvent (page 33), and UnisonoEvent (page 42).

Accepted by: Part_combine_engraver (page 315).

1.2.54 pedal-event

Music event type `pedal-event` is in music objects of type `SostenutoEvent` (page 33), `SustainEvent` (page 35), and `UnaCordaEvent` (page 41).

Not accepted by any engraver or performer.

1.2.55 percent-event

Music event type `percent-event` is in music objects of type `PercentEvent` (page 23).

Accepted by: `Percent_repeat_engraver` (page 316).

1.2.56 pes-or-flexa-event

Music event type `pes-or-flexa-event` is in music objects of type `PesOrFlexaEvent` (page 24).

Accepted by: `Vaticana_ligature_engraver` (page 331).

1.2.57 phrasing-slur-event

Music event type `phrasing-slur-event` is in music objects of type `PhrasingSlurEvent` (page 24).

Accepted by: `Phrasing_slur_engraver` (page 316).

1.2.58 rehearsal-mark-event

Music event type `rehearsal-mark-event` is in music objects of type `RehearsalMarkEvent` (page 26).

Accepted by: `Mark_tracking_translator` (page 308).

1.2.59 repeat-slash-event

Music event type `repeat-slash-event` is in music objects of type `RepeatSlashEvent` (page 27).

Accepted by: `Slash_repeat_engraver` (page 321).

1.2.60 repeat-tie-event

Music event type `repeat-tie-event` is in music objects of type `RepeatTieEvent` (page 27).

Accepted by: `Repeat_tie_engraver` (page 319).

1.2.61 rest-event

Music event type `rest-event` is in music objects of type `RestEvent` (page 28).

Accepted by: `Chord_name_engraver` (page 290), `Completion_rest_engraver` (page 293), `Figured_bass_engraver` (page 299), and `Rest_engraver` (page 320).

1.2.62 rhythmic-event

Music event type `rhythmic-event` is in music objects of type `BassFigureEvent` (page 5), `ClusterNoteEvent` (page 7), `DoublePercentEvent` (page 10), `LyricEvent` (page 17), `MultiMeasureRestEvent` (page 18), `NoteEvent` (page 20), `RepeatSlashEvent` (page 27), `RestEvent` (page 28), and `SkipEvent` (page 31).

Not accepted by any engraver or performer.

1.2.63 script-event

Music event type `script-event` is in music objects of type `ArticulationEvent` (page 4), `ScriptEvent` (page 29), and `TextScriptEvent` (page 36).

Not accepted by any engraver or performer.

1.2.64 section-event

Music event type section-event is in music objects of type SectionEvent (page 29).

Accepted by: Repeat_acknowledge_engraver (page 318).

1.2.65 segno-event

Music event type segno-event is in music objects of type SegnoEvent (page 29).

Accepted by: Repeat_acknowledge_engraver (page 318).

1.2.66 skip-event

Music event type skip-event is in music objects of type SkipEvent (page 31).

Not accepted by any engraver or performer.

1.2.67 slur-event

Music event type slur-event is in music objects of type SlurEvent (page 32).

Accepted by: Slur_engraver (page 322), and Slur_performer (page 322).

1.2.68 solo-one-event

Music event type solo-one-event is in music objects of type SoloOneEvent (page 32).

Not accepted by any engraver or performer.

1.2.69 solo-two-event

Music event type solo-two-event is in music objects of type SoloTwoEvent (page 33).

Not accepted by any engraver or performer.

1.2.70 sostenuto-event

Music event type sostenuto-event is in music objects of type SostenutoEvent (page 33).

Accepted by: Piano_pedal_engraver (page 317), and Piano_pedal_performer (page 317).

1.2.71 spacing-section-event

Music event type spacing-section-event is in music objects of type SpacingSectionEvent (page 34).

Accepted by: Spacing_engraver (page 322).

1.2.72 span-dynamic-event

Music event type span-dynamic-event is in music objects of type CrescendoEvent (page 9), and DecrescendoEvent (page 9).

Accepted by: Dynamic_engraver (page 297).

1.2.73 span-event

Music event type span-event is in music objects of type BeamEvent (page 5), BendSpanEvent (page 6), CrescendoEvent (page 9), DecrescendoEvent (page 9), EpisemaEvent (page 10), FingerGlideEvent (page 12), LigatureEvent (page 16), MeasureCounterEvent (page 17), MeasureSpannerEvent (page 18), PhrasingSlurEvent (page 24), SlurEvent (page 32), SostenutoEvent (page 33), SpanEvent (page 34), StaffSpanEvent (page 34), SustainEvent (page 35), TextSpanEvent (page 36), TremoloSpanEvent (page 40), TrillSpanEvent (page 40), TupletSpanEvent (page 40), UnaCordaEvent (page 41), and VoltaSpanEvent (page 44).

Not accepted by any engraver or performer.

1.2.74 staff-span-event

Music event type staff-span-event is in music objects of type StaffSpanEvent (page 34).

Accepted by: Staff_symbol_engraver (page 324).

1.2.75 StreamEvent

Music event type StreamEvent is in music objects of type AbsoluteDynamicEvent (page 2), AdHocMarkEvent (page 2), AlternativeEvent (page 2), AnnotateOutputEvent (page 3), ApplyOutputEvent (page 3), ArpeggioEvent (page 4), ArticulationEvent (page 4), BassFigureEvent (page 5), BeamEvent (page 5), BeamForbidEvent (page 6), BendAfterEvent (page 6), BendSpanEvent (page 6), BreakDynamicSpanEvent (page 7), BreathingEvent (page 7), ClusterNoteEvent (page 7), CompletizeExtenderEvent (page 8), CrescendoEvent (page 9), DecrescendoEvent (page 9), DoublePercentEvent (page 10), DurationLineEvent (page 10), EpisemaEvent (page 10), ExtenderEvent (page 11), FineEvent (page 12), FingerGlideEvent (page 12), FingeringEvent (page 13), FootnoteEvent (page 13), GlissandoEvent (page 13), HarmonicEvent (page 14), HyphenEvent (page 14), KeyChangeEvent (page 15), LabelEvent (page 15), LaissezVibrerEvent (page 15), LigatureEvent (page 16), LineBreakEvent (page 16), LyricEvent (page 17), MeasureCounterEvent (page 17), MeasureSpannerEvent (page 18), MultiMeasureArticulationEvent (page 18), MultiMeasureRestEvent (page 18), MultiMeasureTextEvent (page 19), NoteEvent (page 20), NoteGroupingEvent (page 20), OttavaEvent (page 20), PageBreakEvent (page 21), PageTurnEvent (page 22), PercentEvent (page 23), PesOrFlexaEvent (page 24), PhrasingSlurEvent (page 24), RehearsalMarkEvent (page 26), RepeatSlashEvent (page 27), RepeatTieEvent (page 27), RestEvent (page 28), ScriptEvent (page 29), SectionEvent (page 29), SegnoEvent (page 29), SkipEvent (page 31), SlurEvent (page 32), SoloOneEvent (page 32), SoloTwoEvent (page 33), SostenutoEvent (page 33), SpacingSectionEvent (page 34), SpanEvent (page 34), StaffSpanEvent (page 34), StringNumberEvent (page 35), StrokeFingerEvent (page 35), SustainEvent (page 35), TempoChangeEvent (page 36), TextScriptEvent (page 36), TextSpanEvent (page 36), TieEvent (page 37), TimeSignatureEvent (page 37), TremoloEvent (page 39), TremoloSpanEvent (page 40), TrillSpanEvent (page 40), TupletSpanEvent (page 40), UnaCordaEvent (page 41), UnisonoEvent (page 42), VoltaSpanEvent (page 44), and VowelTransitionEvent (page 45).

Not accepted by any engraver or performer.

1.2.76 string-number-event

Music event type string-number-event is in music objects of type StringNumberEvent (page 35).

Accepted by: Bend_spanner_engraver (page 289), Fretboard_engraver (page 301), and Tab_note_heads_engraver (page 325).

1.2.77 stroke-finger-event

Music event type stroke-finger-event is in music objects of type StrokeFingerEvent (page 35).

Not accepted by any engraver or performer.

1.2.78 sustain-event

Music event type sustain-event is in music objects of type SustainEvent (page 35).

Accepted by: Piano_pedal_engraver (page 317), and Piano_pedal_performer (page 317).

1.2.79 tempo-change-event

Music event type `tempo-change-event` is in music objects of type `TempoChangeEvent` (page 36).

Accepted by: `Metronome_mark_engraver` (page 310).

1.2.80 text-script-event

Music event type `text-script-event` is in music objects of type `TextScriptEvent` (page 36).

Accepted by: `Text_engraver` (page 327).

1.2.81 text-span-event

Music event type `text-span-event` is in music objects of type `TextSpanEvent` (page 36).

Accepted by: `Text_spanner_engraver` (page 327).

1.2.82 tie-event

Music event type `tie-event` is in music objects of type `TieEvent` (page 37).

Accepted by: `Drum_note_performer` (page 296), `Note_performer` (page 313), `Tie_engraver` (page 327), and `Tie_performer` (page 328).

1.2.83 time-signature-event

Music event type `time-signature-event` is in music objects of type `TimeSignatureEvent` (page 37).

Accepted by: `Time_signature_engraver` (page 328), and `Time_signature_performer` (page 328).

1.2.84 tremolo-event

Music event type `tremolo-event` is in music objects of type `TremoloEvent` (page 39).

Accepted by: `Stem_engraver` (page 324).

1.2.85 tremolo-span-event

Music event type `tremolo-span-event` is in music objects of type `TremoloSpanEvent` (page 40).

Accepted by: `Chord_tremolo_engraver` (page 291).

1.2.86 trill-span-event

Music event type `trill-span-event` is in music objects of type `TrillSpanEvent` (page 40).

Accepted by: `Trill_spanner_engraver` (page 330).

1.2.87 tuplet-span-event

Music event type `tuplet-span-event` is in music objects of type `TupletSpanEvent` (page 40).

Accepted by: `Stem_engraver` (page 324), and `Tuplet_engraver` (page 330).

1.2.88 una-corda-event

Music event type `una-corda-event` is in music objects of type `UnaCordaEvent` (page 41).

Accepted by: `Piano_pedal_engraver` (page 317), and `Piano_pedal_performer` (page 317).

1.2.89 unisono-event

Music event type `unisono-event` is in music objects of type `UnisonoEvent` (page 42).

Not accepted by any engraver or performer.

1.2.90 volta-span-event

Music event type `volta-span-event` is in music objects of type `VoltaSpanEvent` (page 44).

Accepted by: `Repeat_acknowledge_engraver` (page 318), and `Volta_engraver` (page 331).

1.2.91 vowel-transition-event

Music event type `vowel-transition-event` is in music objects of type `VowelTransitionEvent` (page 45).

Accepted by: `Hyphen_engraver` (page 304).

1.3 Music properties

`absolute-octave` (integer)
: The absolute octave for an octave check note.

`alteration` (number)
: Alteration for figured bass.

`alteration-bracket` (boolean)
: Put brackets around bass figure alteration.

`alternative-dir` (direction)
: Indicates that an `alternative-event` is the first (-1), middle (0), or last (1) of group of alternate endings.

`articulation-type` (string)
: Key for script definitions alist.

  TODO: Consider making type into symbol.

`articulations` (list of music objects)
: Articulation events specifically for this note.

`associated-context` (string)
: Name of the context associated with this `\lyricsto` section.

`associated-context-type` (symbol)
: Type of the context associated with this `\lyricsto` section.

`augmented` (boolean)
: This figure is for an augmented figured bass (with `+` sign).

`augmented-slash` (boolean)
: This figure is for an augmented figured bass (back-slashed number).

`automatically-numbered` (boolean)
: Should a footnote be automatically numbered?

`autosplit-end` (boolean)
: Duration of event was truncated by automatic splitting in `Completion_heads_engraver`.

`bass` (boolean)
: Set if this note is a bass note in a chord.

`beat-structure` (list)
: A beatStructure to be used in autobeaming.

`bracket-start` (boolean)
: Start a bracket here.

  TODO: Use SpanEvents?

`bracket-stop` (boolean)
: Stop a bracket here.

`break-penalty` (number)
: Penalty for line break hint.

`break-permission` (symbol)
: Whether to allow, forbid or force a line break.

`cautionary` (boolean)
: If set, this alteration needs a cautionary accidental.

`change-tag` (symbol)
: Tag identifying the musical scope of a context change. The change applies to the nearest enclosing music with this tag.

`change-to-id` (string)
: Name of the context to change to.

`change-to-type` (symbol)
: Type of the context to change to.

`class` (symbol)
: The class name of an event class.

`context` (context)
: The context to which an event is sent.

`context-id` (string)
: Name of context.

`context-type` (symbol)
: Type of context.

`create-new` (boolean)
: Create a fresh context.

`delta-step` (number)
: How much should a fall change pitch?

`denominator` (integer)
: Denominator in a time signature.

`digit` (non-negative, exact integer)
: Digit for fingering.

`diminished` (boolean)
: This bass figure should be slashed.

`direction` (direction)
: Print this up or down?

`drum-type` (symbol)
: Which percussion instrument to play this note on.

`duration` (duration)
: Duration of this note or lyric.

`element` (music)
: The single child of a Music_wrapper music object, or the body of a repeat.

`elements` (list of music objects)
: A list of elements for sequential of simultaneous music, or the alternatives of repeated music.

`elements-callback` (procedure)
: Return a list of children, for use by a sequential iterator. Takes a single music parameter.

`error-found` (boolean)
: If true, a parsing error was found in this expression.

`figure` (integer)
: A bass figure.

`footnote-text` (markup)
: Text to appear in a footnote.

`force-accidental` (boolean)
: If set, a cautionary accidental should always be printed on this note.

`grob-property` (symbol)
: The symbol of the grob property to set.

`grob-property-path` (list)
: A list of symbols, locating a nested grob property, e.g., `(beamed-lengths details)`.

`grob-value` (any type)
: The value of the grob property to set.

`id` (symbol)
: The ID of an event.

`input-tag` (any type)
: Arbitrary marker to relate input and output.

`inversion` (boolean)
: If set, this chord note is inverted.

`iterator-ctor` (procedure)
: Function to construct a `music-event-iterator` object for this music.

`label` (non-negative, exact integer)
: Sequence number of a mark. 1 is first.

`last-pitch` (pitch)
: The last pitch after relativization.

`length` (moment)
: The endpoint of this music. This property is unhappily named in that it does not account for any initial grace notes: the full length of the music is `length` minus the start time. A value of `INF-MOMENT` indicates indefinite length.

`length-callback` (procedure)
: How to compute the duration of this music. This property can only be defined as initializer in `scm/define-music-types.scm`.

`line-break-permission` (symbol)
: When the music is at top-level, whether to allow, forbid or force a line break.

`metronome-count` (number or pair)
: How many beats in a minute?

`midi-extra-velocity` (integer)
: How much louder or softer should this note be in MIDI output? The default is 0.

`midi-length` (procedure)
: Function to determine how long to play a note in MIDI. It should take a moment (the written length of the note) and a context, and return a moment (the length to play the note).

`moment` (moment)
: The moment at which an event happens.

`music-cause` (music)
: The music object that is the cause of an event.

`name` (symbol)
: Name of this music object.

`no-continuation` (boolean)
: If set, disallow continuation lines.

`numerator` (integer)
: Numerator of a time signature.

`octavation` (integer)
: This pitch was octavated by how many octaves? For chord inversions, this is negative.

`once` (boolean)
: Apply this operation only during one time step?

`ops` (any type)
: The operations to apply during the creation of a context.

`origin` (input location)
: Where was this piece of music defined?

`ottava-number` (integer)
: The octavation for `\ottava`.

`page-break-permission` (symbol)
: When the music is at top-level, whether to allow, forbid or force a page break.

`page-label` (symbol)
: The label of a page marker.

`page-marker` (boolean)
: If true, and the music expression is found at top-level, a page marker object is instanciated instead of a score.

`page-turn-permission` (symbol)
: When the music is at top-level, whether to allow, forbid or force a page turn.

`part-combine-status` (symbol)
: Change to what kind of state? Options are `solo1`, `solo2` and `unisono`.

`pitch` (pitch)
: The pitch of this note.

`pitch-alist` (list)
: A list of pitches jointly forming the scale of a key signature.

`pop-first` (boolean)
: Do a revert before we try to do an override on some grob property.

`procedure` (procedure)
: The function to run with `\applycontext`. It must take a single argument, being the context.

`property-operations` (list)
: Do these operations for instantiating the context.

`property-path` (symbol)
: The path of a property.

`quoted-context-id` (string)
: The ID of the context to direct quotes to, e.g., `cue`.

`quoted-context-type` (symbol)
: The name of the context to direct quotes to, e.g., `Voice`.

`quoted-events` (vector)
: A vector of with `moment` and `event-list` entries.

`quoted-music-clef` (string)
: The clef of the voice to quote.

`quoted-music-name` (string)
: The name of the voice to quote.

`quoted-transposition` (pitch)
: The pitch used for the quote, overriding `\transposition`.

`quoted-voice-direction` (direction)
: Should the quoted voice be up-stem or down-stem?

`repeat-count` (integer)
: Do a `\repeat` how often?

`search-direction` (direction)
: Limits the scope of `\context` searches.

`slash-count` (integer)
: The number of slashes in a single-beat repeat. If zero, signals a beat containing varying durations.

`span-direction` (direction)
: Does this start or stop a spanner?

`span-text` (markup)
: The displayed text for dynamic text spanners (e.g., cresc.)

`span-type` (symbol)
: What kind of dynamic spanner should be created? Options are `'text` and `'hairpin`.

`spanner-id` (index or symbol)
: Identifier to distinguish concurrent spanners.

`start-callback` (procedure)
: Function to compute the negative length of starting grace notes. This property can only be defined as initializer in `scm/define-music-types.scm`.

`string-number` (integer)
: The number of the string in a `StringNumberEvent`.

`symbol` (symbol)
: Grob name to perform an override or revert on.

`tags` (list)
: List of symbols that for denoting extra details, e.g., `\tag #'part ...` could tag a piece of music as only being active in a part.

`tempo-unit` (duration)
: The unit for the metronome count.

`text` (markup)
: Markup expression to be printed.

`to-relative-callback` (procedure)
: How to transform a piece of music to relative pitches.

`tonic` (pitch)
: Base of the scale.

`tremolo-type` (integer)
: Speed of tremolo, e.g., 16 for `c4:16`.

`trill-pitch` (pitch)
: Pitch of other note of the trill.

`tweaks` (list)
: An alist of properties to override in the backend for the grob made of this event.

`type` (symbol)
: The type of this music object. Determines iteration in some cases.

`types` (list)
: The types of this music object; determines by what engraver this music expression is processed.

`untransposable` (boolean)
: If set, this music is not transposed.

`value` (any type)
: Assignment value for a translation property.

`void` (boolean)
: If this property is `#t`, then the music expression is to be discarded by the toplevel music handler.

`volta-numbers` (number list)
: Volte to which this music applies.

`what` (symbol)
: What to change for auto-change.

  FIXME: Naming.

`X-offset` (number)
: Offset of resulting grob; only used for balloon texts.

`Y-offset` (number)
: Offset of resulting grob; only used for balloon texts.

2 Translation

2.1 Contexts

2.1.1 ChoirStaff

Identical to StaffGroup except that the contained staves are not connected vertically.

This context creates the following layout object(s): Arpeggio (page 355), InstrumentName (page 420), SpanBarStub (page 478), SystemStartBar (page 491), SystemStartBrace (page 492), SystemStartBracket (page 493), SystemStartSquare (page 494), and VerticalAlignment (page 514).

This context sets the following properties:

- Set grob property extra-spacing-width in DynamicText (page 402), to #f.
- Set translator property instrumentName to '().
- Set translator property instrumentName to '().
- Set translator property localAlterations to #f.
- Set translator property localAlterations to '().
- Set translator property localAlterations to '().
- Set translator property shortInstrumentName to '().
- Set translator property shortInstrumentName to '().
- Set translator property systemStartDelimiter to 'SystemStartBracket.
- Set translator property topLevelAlignment to #f.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type Staff (page 220).

Context ChoirStaff can contain ChoirStaff (page 61), ChordNames (page 63), Devnull (page 76), DrumStaff (page 76), Dynamics (page 92), FiguredBass (page 95), FretBoards (page 97), GrandStaff (page 99), GregorianTranscriptionStaff (page 101), KievanStaff (page 122), Lyrics (page 143), MensuralStaff (page 145), NoteNames (page 166), OneStaff (page 170), PetrucciStaff (page 171), PianoStaff (page 192), RhythmicStaff (page 194), Staff (page 220), StaffGroup (page 229), TabStaff (page 231), and VaticanaStaff (page 251).

This context is built from the following engraver(s):

Instrument_name_engraver (page 304)

Create a system start text for instrument or vocal names.

Properties (read)

currentCommandColumn (graphical (layout) object)

Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

instrumentName (markup)

The name to print left of a staff. The instrumentName property labels the staff in the first system, and the shortInstrumentName property labels following lines.

shortInstrumentName (markup)

See instrumentName.

shortVocalName (markup)

Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Span_arpeggio_engraver` (page 322)
: Make arpeggios that span multiple staves.

Properties (read)

`connectArpeggios` (boolean)
: If set, connect arpeggios across piano staff.

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Span_bar_stub_engraver` (page 323)
: Make stubs for span bars in all contexts that the span bars cross.

This engraver creates the following layout object(s): `SpanBarStub` (page 478).

`System_start_delimiter_engraver` (page 325)
: Create a system start delimiter (i.e., a `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket` or `SystemStartSquare` spanner).

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`systemStartDelimiter` (symbol)
: Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

`systemStartDelimiterHierarchy` (pair)
: A nested list, indicating the nesting of a start delimiters.

This engraver creates the following layout object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

`Vertical_align_engraver` (page 331)
: Catch groups (staves, lyrics lines, etc.) and stack them vertically.

Properties (read)

`alignAboveContext` (string)
: Where to insert newly created context in vertical alignment.

`alignBelowContext` (string)
: Where to insert newly created context in vertical alignment.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAlignment` (page 514).

2.1.2 ChordNames

Typesets chord names.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `ChordName` (page 377), `StaffSpacing` (page 479), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `font-size` in `Parentheses` (page 459), to `1.5`.
- Set grob property `nonstaff-nonstaff-spacing.padding` in `VerticalAxisGroup` (page 514), to `0.5`.
- Set grob property `nonstaff-relatedstaff-spacing.padding` in `VerticalAxisGroup` (page 514), to `0.5`.
- Set grob property `remove-empty` in `VerticalAxisGroup` (page 514), to `#t`.
- Set grob property `remove-first` in `VerticalAxisGroup` (page 514), to `#t`.
- Set grob property `staff-affinity` in `VerticalAxisGroup` (page 514), to `-1`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Alteration_glyph_engraver` (page 283)

Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)

Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)

Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)

Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)

True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)

A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)

True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Chord_name_engraver` (page 290)
Catch note and rest events and generate the appropriate chordname.
Music types accepted: `note-event` (page 50), and `rest-event` (page 51),
Properties (read)

`chordChanges` (boolean)
Only show changes in chords scheme?

`chordNameExceptions` (list)
An alist of chord exceptions. Contains `(chord . markup)` entries.

`chordNameExceptions` (list)
An alist of chord exceptions. Contains `(chord . markup)` entries.

`chordNameFunction` (procedure)
The function that converts lists of pitches to chord names.

`chordNoteNamer` (procedure)
A function that converts from a pitch object to a text markup. Used for single pitches.

`chordRootNamer` (procedure)
A function that converts from a pitch object to a text markup. Used for chords.

`lastChord` (markup)
Last chord, used for detecting chord changes.

`majorSevenSymbol` (markup)
How should the major 7th be formatted in a chord name?

`noChordSymbol` (markup)
Markup to be displayed for rests in a ChordNames context.

Properties (write)

`lastChord` (markup)
Last chord, used for detecting chord changes.

This engraver creates the following layout object(s): `ChordName` (page 377).

`Output_property_engraver` (page 314)
Apply a procedure to any grob acknowledged.
Music types accepted: `apply-output-event` (page 45),

`Separating_line_group_engraver` (page 321)
Generate objects for computing spacing parameters.
Properties (read)

`createSpacing` (boolean)
Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

2.1.3 CueVoice

Corresponds to a voice on a staff. This context handles the conversion of dynamic signs, stems, beams, super- and subscripts, slurs, ties, and rests.

You have to instantiate this explicitly if you want to have multiple voices on the same staff.

This context also accepts commands for the following context(s): Voice (page 271).

This context creates the following layout object(s): Arpeggio (page 355), Beam (page 366), BendAfter (page 368), BreathingSign (page 373), ClusterSpanner (page 382), ClusterSpannerBeacon (page 383), CombineTextScript (page 383), Dots (page 394), DoublePercentRepeat (page 395), DoublePercentRepeatCounter (page 396), DoubleRepeatSlash (page 398), DynamicLineSpanner (page 401), DynamicText (page 402), DynamicTextSpanner (page 403), FingerGlideSpanner (page 406), Fingering (page 408), Flag (page 410), Glissando (page 414), Hairpin (page 417), InstrumentSwitch (page 421), LaissezVibrerTie (page 430), LaissezVibrerTieColumn (page 431), LigatureBracket (page 434), MultiMeasureRest (page 446), MultiMeasureRestNumber (page 447), MultiMeasureRestScript (page 449), MultiMeasureRestText (page 450), NoteColumn (page 454), NoteHead (page 455), NoteSpacing (page 456), PercentRepeat (page 460), PercentRepeatCounter (page 461), PhrasingSlur (page 462), RepeatSlash (page 467), RepeatTie (page 467), RepeatTieColumn (page 469), Rest (page 469), Script (page 470), ScriptColumn (page 471), Slur (page 472), Stem (page 481), StemStub (page 483), StemTremolo (page 484), StringNumber (page 485), StrokeFinger (page 487), TextScript (page 496), TextSpanner (page 498), Tie (page 500), TieColumn (page 501), TrillPitchAccidental (page 504), TrillPitchGroup (page 505), TrillPitchHead (page 506), TrillSpanner (page 507), TupletBracket (page 509), TupletNumber (page 510), and VoiceFollower (page 516).

This context sets the following properties:

- Set grob property beam-thickness in Beam (page 366), to 0.35.
- Set grob property beam-thickness in StemTremolo (page 484), to 0.35.
- Set grob property ignore-ambitus in NoteHead (page 455), to #t.
- Set grob property length-fraction in Beam (page 366), to 0.629960524947437.
- Set grob property length-fraction in Stem (page 481), to 0.629960524947437.
- Set translator property fontSize to -4.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

Arpeggio_engraver (page 284)

Generate an Arpeggio symbol.

Music types accepted: arpeggio-event (page 45),

This engraver creates the following layout object(s): Arpeggio (page 355).

Auto_beam_engraver (page 284)

Generate beams based on measure characteristics and observed Stems. Uses baseMoment, beatStructure, beamExceptions, measureLength, and measurePosition to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [Stem_engraver], page 324, properties stemLeftBeamCount and stemRightBeamCount.

Music types accepted: beam-forbid-event (page 46),

Properties (read)

autoBeaming (boolean)

If set to true then beams are generated automatically.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)

Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)

Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
: Generate beams for tremolo repeats.

  Music types accepted: `tremolo-span-event` (page 54),

  This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
: Engrave a cluster using `Spanner` notation.

  Music types accepted: `cluster-note-event` (page 46),

  This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
: Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

  This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
: Make double measure repeats.

  Music types accepted: `double-percent-event` (page 47),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `measureLength` (moment)
  : Length of one measure in the current time signature.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  Properties (write)

  `forbidBreak` (boolean)
  : If set to `#t`, prevent a line break at this point.

  This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
: Align hairpins and dynamic texts on a horizontal line.

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
: Create hairpins, dynamic texts and dynamic text spanners.

  Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)
: Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)
: Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
: Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
: A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Ligature_bracket_engraver` (page 307)
Handle `Ligature_events` by engraving `Ligature` brackets.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `LigatureBracket` (page 434).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)
: Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
: Generate note heads.

Music types accepted: `note-event` (page 50),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

This engraver creates the following layout object(s): `NoteHead` (page 455).

`Note_spacing_engraver` (page 313)
: Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

  This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings 'a2', 'Solo', 'Solo II', and 'unisono'.

  Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

  Properties (read)

  `aDueText` (markup)
  : Text to print at a unisono passage.

  `partCombineTextsOnNote` (boolean)
  : Print part-combine texts only on the next note rather than immediately on rests or skips.

  `printPartCombineTexts` (boolean)
  : Set 'Solo' and 'A due' texts in the part combiner?

  `soloIIText` (markup)
  : The text for the start of a solo for voice 'two' when part-combining.

  `soloText` (markup)
  : The text for the start of a solo when part-combining.

  This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
: Make whole measure repeats.

  Music types accepted: `percent-event` (page 51),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)

Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)

Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)

Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)

The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)

Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)

Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)

Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)

The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)

Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
Build slur grobs from slur events.

Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

Properties (read)

`doubleSlurs` (boolean)
If set, two slurs are created for every slurred note, one above and one below the chord.

`slurMelismaBusy` (boolean)
Signal if a slur is present.

This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
See `stemLeftBeamCount`.

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
Create text scripts.

Music types accepted: `text-script-event` (page 54),

This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
Create text spanner from an event.

Music types accepted: `text-span-event` (page 54),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)

Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)

Create trill spanner from an event.

Music types accepted: `trill-span-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)

Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.4 Devnull

Silently discards all musical information given to this context.

This context also accepts commands for the following context(s): `Staff` (page 220), and `Voice` (page 271).

This context creates the following layout object(s): none.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

2.1.5 DrumStaff

Handles typesetting for percussion.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `RestCollision` (page 470), `ScriptRow` (page 472), `SostenutoPedalLineSpanner` (page 475), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `SustainPedalLineSpanner` (page 489), `TimeSignature` (page 502), `UnaCordaPedalLineSpanner` (page 512), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `staff-padding` in `Script` (page 470), to `0.75`.
- Set translator property `clefGlyph` to `"clefs.percussion"`.
- Set translator property `clefPosition` to `0`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `ignoreFiguredBassRest` to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `ottavationMarkups` to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property `shortInstrumentName` to `'()`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `DrumVoice` (page 82).

Context `DrumStaff` can contain `CueVoice` (page 65), `DrumVoice` (page 82), and `NullVoice` (page 168).

This context is built from the following engraver(s):

`Alteration_glyph_engraver` (page 283)
: Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
: Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

  This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

  Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

  Properties (read)

  `figuredBassAlterationDirection` (direction)
  : Where to put alterations relative to the main figure.

  `figuredBassCenterContinuations` (boolean)
  : Whether to vertically center pairs of extender lines. This does not work with three or more lines.

  `figuredBassFormatter` (procedure)
  : A routine generating a markup for a bass figure.

  `ignoreFiguredBassRest` (boolean)
  : Don't swallow rest events.

  `implicitBassFigures` (list)
  : A list of bass figures that are not printed as numbers, but only as extender lines.

  `useBassFigureExtenders` (boolean)
  : Whether to use extender lines for repeated bass figures.

  This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
: Position figured bass alignments over notes.

  This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
: Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

  Properties (read)

  `fontSize` (number)
  : The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
: A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
: A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
: Engraver to merge multi-measure rest numbers in multiple voices.

This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
: Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Pure_from_neighbor_engraver` (page 318)
: Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
: Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
: Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
: Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
: Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)

Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)

break visibility for the initial time signature.

`partialBusy` (boolean)

Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)

A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.6 `DrumVoice`

A voice on a percussion staff.

This context also accepts commands for the following context(s): `Voice` (page 271).

This context creates the following layout object(s): `Beam` (page 366), `BendAfter` (page 368), `BreathingSign` (page 373), `CombineTextScript` (page 383), `Dots` (page 394), `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `FingerGlideSpanner` (page 406), `Flag` (page 410), `Hairpin` (page 417), `InstrumentSwitch` (page 421), `LaissezVibrerTie` (page 430), `LaissezVibrerTieColumn` (page 431), `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `NoteColumn` (page 454), `NoteHead` (page 455), `NoteSpacing` (page 456), `PercentRepeat` (page 460), `PercentRepeatCounter` (page 461), `PhrasingSlur` (page 462), `RepeatSlash` (page 467), `RepeatTie` (page 467), `RepeatTieColumn` (page 469), `Rest` (page 469), `Script` (page 470), `ScriptColumn` (page 471), `Slur` (page 472), `Stem` (page 481), `StemStub` (page 483), `StemTremolo` (page 484), `TextScript` (page 496), `TextSpanner` (page 498), `Tie` (page 500), `TieColumn` (page 501), `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), `TrillPitchHead` (page 506), `TrillSpanner` (page 507), `TupletBracket` (page 509), and `TupletNumber` (page 510).

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Auto_beam_engraver` (page 284)

Generate beams based on measure characteristics and observed Stems. Uses `baseMoment`, `beatStructure`, `beamExceptions`, `measureLength`, and `measurePosition` to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [`Stem_engraver`], page 324, properties `stemLeftBeamCount` and `stemRightBeamCount`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)

If set to true then beams are generated automatically.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)

Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)

Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
: Generate beams for tremolo repeats.

  Music types accepted: `tremolo-span-event` (page 54),

  This engraver creates the following layout object(s): `Beam` (page 366).

`Dots_engraver` (page 296)
: Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

  This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
: Make double measure repeats.

  Music types accepted: `double-percent-event` (page 47),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `measureLength` (moment)
  : Length of one measure in the current time signature.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  Properties (write)

  `forbidBreak` (boolean)
  : If set to `#t`, prevent a line break at this point.

  This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Drum_notes_engraver` (page 296)
: Generate drum note heads.

  Music types accepted: `note-event` (page 50),

  Properties (read)

  `drumStyleTable` (hash table)
  : A hash table which maps drums to layout settings. Predefined values: '`drums-style`', '`agostini-drums-style`', '`timbales-style`', '`congas-style`', '`bongos-style`', and '`percussion-style`'.

    The layout style is a hash table, containing the drum-pitches (e.g., the symbol '`hihat`') as keys, and a list (*`notehead-style script vertical-position`*) as values.

  This engraver creates the following layout object(s): `NoteHead` (page 455), and `Script` (page 470).

`Dynamic_align_engraver` (page 297)
: Align hairpins and dynamic texts on a horizontal line.

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)

Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)

Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Font_size_engraver` (page 300)

Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)

Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`Grace_auto_beam_engraver` (page 302)
: Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

  Music types accepted: `beam-forbid-event` (page 46),

  Properties (read)

  `autoBeaming` (boolean)
  : If set to true then beams are generated automatically.

  This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
: Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

  Music types accepted: `beam-event` (page 46),

  Properties (read)

  `baseMoment` (moment)
  : Smallest unit of time that will stand on its own as a subdivided section.

  `beamMelismaBusy` (boolean)
  : Signal if a beam is present.

  `beatStructure` (list)
  : List of `baseMoment`s that are combined to make beats.

  `subdivideBeams` (boolean)
  : If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

  This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
: Set font size and other properties for grace notes.

  Properties (read)

  `graceSettings` (list)
  : Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

  Properties (read)

  `busyGrobs` (list)
  : A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

  Properties (write)

  `busyGrobs` (list)
  : A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
True at the beginning of a measure.

`restNumberThreshold` (number)
If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`Note_spacing_engraver` (page 313)
Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
Part combine engraver for orchestral scores: Print markings 'a2', 'Solo', 'Solo II', and 'unisono'.

Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

Properties (read)

`aDueText` (markup)
Text to print at a unisono passage.

`partCombineTextsOnNote` (boolean)
Print part-combine texts only on the next note rather than immediately on rests or skips.

`printPartCombineTexts` (boolean)
Set 'Solo' and 'A due' texts in the part combiner?

`soloIIText` (markup)
The text for the start of a solo for voice 'two' when part-combining.

`soloText` (markup)
The text for the start of a solo when part-combining.

This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
Make whole measure repeats.

Music types accepted: `percent-event` (page 51),

Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)
Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)
Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)
The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)
The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
: Make beat repeats.

  Music types accepted: `repeat-slash-event` (page 51),

  This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
: Build slur grobs from slur events.

  Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

  Properties (read)

  `doubleSlurs` (boolean)
  : If set, two slurs are created for every slurred note, one above and one below the chord.

  `slurMelismaBusy` (boolean)
  : Signal if a slur is present.

  This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
: Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
: Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

  Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

  Properties (read)

  `stemLeftBeamCount` (integer)
  : Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

  `stemRightBeamCount` (integer)
  : See `stemLeftBeamCount`.

  `whichBar` (string)
  : This property is read to determine what type of bar line to create.

    Example:

    ```
    \set Staff.whichBar = ".|:"
    ```

    This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

  This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
: Create text scripts.

  Music types accepted: `text-script-event` (page 54),

  This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
Create text spanner from an event.

Music types accepted: `text-span-event` (page 54),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)
Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)
Create trill spanner from an event.

Music types accepted: `trill-span-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)
Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.7 Dynamics

Holds a single line of dynamics, which will be centered between the staves surrounding this context.

This context also accepts commands for the following context(s): `Voice` (page 271).

This context creates the following layout object(s): `BarLine` (page 358), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `Hairpin` (page 417), `PianoPedalBracket` (page 464), `Script` (page 470), `SostenutoPedal` (page 474), `SustainPedal` (page 488), `TextScript` (page 496), `TextSpanner` (page 498), `UnaCordaPedal` (page 511), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `font-shape` in `TextScript` (page 496), to `'italic`.
- Set grob property `nonstaff-relatedstaff-spacing` in `VerticalAxisGroup` (page 514), to:

  ```
  '((basic-distance . 5) (padding . 0.5))
  ```
- Set grob property `outside-staff-priority` in `DynamicLineSpanner` (page 401), to `#f`.
- Set grob property `outside-staff-priority` in `DynamicText` (page 402), to `#f`.
- Set grob property `outside-staff-priority` in `Hairpin` (page 417), to `#f`.
- Set grob property `staff-affinity` in `VerticalAxisGroup` (page 514), to `0`.
- Set grob property `Y-offset` in `DynamicLineSpanner` (page 401), to `0`.
- Set translator property `pedalSustainStrings` to:

  ```
  '("Ped." "*Ped." "*")
  ```
- Set translator property `pedalUnaCordaStrings` to:

  ```
  '("una corda" "" "tre corde")
  ```

This is a ‘Bottom’ context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  `keepAliveInterfaces` (list)
  : A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Dynamic_align_engraver` (page 297)
: Align hairpins and dynamic texts on a horizontal line.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
: Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_engraver` (page 317)
: Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(`*`up updown down`*`)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Script_engraver` (page 320)
: Handle note scripted articulations.

  Music types accepted: `articulation-event` (page 45),

  Properties (read)

  `scriptDefinitions` (list)
  : The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

  This engraver creates the following layout object(s): `Script` (page 470).

`Text_engraver` (page 327)
: Create text scripts.

  Music types accepted: `text-script-event` (page 54),

  This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
: Create text spanner from an event.

  Music types accepted: `text-span-event` (page 54),

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `TextSpanner` (page 498).

2.1.8 FiguredBass

A context for printing a figured bass line.

This context creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `StaffSpacing` (page 479), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `nonstaff-nonstaff-spacing.padding` in `VerticalAxisGroup` (page 514), to `0.5`.
- Set grob property `nonstaff-relatedstaff-spacing.padding` in `VerticalAxisGroup` (page 514), to `0.5`.
- Set grob property `remove-empty` in `VerticalAxisGroup` (page 514), to `#t`.
- Set grob property `remove-first` in `VerticalAxisGroup` (page 514), to `#t`.
- Set grob property `staff-affinity` in `VerticalAxisGroup` (page 514), to `1`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

2.1.9 `FretBoards`

A context for displaying fret diagrams.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `FretBoard` (page 412), `InstrumentName` (page 420), `StaffSpacing` (page 479), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set translator property `handleNegativeFrets` to `'recalculate`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `predefinedDiagramTable` to `#<hash-table 0/113>`.
- Set translator property `restrainOpenStrings` to `#f`.
- Set translator property `shortInstrumentName` to `'()`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Fretboard_engraver` (page 301)

Generate fret diagram from one or more events of type `NoteEvent`.

Music types accepted: `fingering-event` (page 47), `note-event` (page 50), and `string-number-event` (page 53),

Properties (read)

`chordChanges` (boolean)
: Only show changes in chords scheme?

`defaultStrings` (list)
: A list of strings to use in calculating frets for tablatures and fretboards if no strings are provided in the notes for the current moment.

`highStringOne` (boolean)
: Whether the first string is the string with highest pitch on the instrument. This used by the automatic string selector for tablature notation.

`maximumFretStretch` (number)
: Don't allocate frets further than this from specified frets.

`minimumFret` (number)
: The tablature auto string-selecting mechanism selects the highest string with a fret at least `minimumFret`.

`noteToFretFunction` (procedure)
: Convert list of notes and list of defined strings to full list of strings and fret numbers. Parameters: The context, a list of note events, a list of tabstring events, and the fretboard grob if a fretboard is desired.

`predefinedDiagramTable` (hash table)
: The hash table of predefined fret diagrams to use in FretBoards.

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

`tablatureFormat` (procedure)
: A function formatting a tablature note head. Called with three arguments: context, string number and, fret number. It returns the text as a markup.

This engraver creates the following layout object(s): `FretBoard` (page 412).

`Instrument_name_engraver` (page 304)

Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
See `instrumentName`.

`shortVocalName` (markup)
Name of a vocal line, short version.

`vocalName` (markup)
Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Output_property_engraver` (page 314)
Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Separating_line_group_engraver` (page 321)
Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

2.1.10 `Global`

Hard coded entry point for LilyPond. Cannot be tuned.

This context creates the following layout object(s): none.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Score` (page 197).

Context `Global` can contain `Score` (page 197).

2.1.11 `GrandStaff`

Groups staves while adding a bracket on the left side, grouping the staves together. The bar lines of the contained staves are connected vertically. `StaffGroup` only consists of a collection of staves, with a bracket in front and spanning bar lines.

This context creates the following layout object(s): `Arpeggio` (page 355), `InstrumentName` (page 420), `SpanBar` (page 477), `SpanBarStub` (page 478), `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), `SystemStartSquare` (page 494), and `VerticalAlignment` (page 514).

This context sets the following properties:

- Set grob property `extra-spacing-width` in `DynamicText` (page 402), to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `#f`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `localAlterations` to `'()`.

- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `systemStartDelimiter` to `'SystemStartBrace`.
- Set translator property `systemStartDelimiter` to `'SystemStartBracket`.
- Set translator property `topLevelAlignment` to `#f`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Staff` (page 220).

Context `GrandStaff` can contain `ChoirStaff` (page 61), `ChordNames` (page 63), `Devnull` (page 76), `DrumStaff` (page 76), `Dynamics` (page 92), `FiguredBass` (page 95), `FretBoards` (page 97), `GrandStaff` (page 99), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `OneStaff` (page 170), `PetrucciStaff` (page 171), `PianoStaff` (page 192), `RhythmicStaff` (page 194), `Staff` (page 220), `StaffGroup` (page 229), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

This context is built from the following engraver(s):

`Instrument_name_engraver` (page 304)

Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Output_property_engraver` (page 314)

Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Span_arpeggio_engraver` (page 322)

Make arpeggios that span multiple staves.

Properties (read)

`connectArpeggios` (boolean)
: If set, connect arpeggios across piano staff.

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Span_bar_engraver` (page 323)

Make cross-staff bar lines: It catches all normal bar lines and draws a single span bar across them.

This engraver creates the following layout object(s): `SpanBar` (page 477).

`Span_bar_stub_engraver` (page 323)
: Make stubs for span bars in all contexts that the span bars cross.

  This engraver creates the following layout object(s): `SpanBarStub` (page 478).

`System_start_delimiter_engraver` (page 325)
: Create a system start delimiter (i.e., a `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket` or `SystemStartSquare` spanner).

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `systemStartDelimiter` (symbol)
  : Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

  `systemStartDelimiterHierarchy` (pair)
  : A nested list, indicating the nesting of a start delimiters.

  This engraver creates the following layout object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

`Vertical_align_engraver` (page 331)
: Catch groups (staves, lyrics lines, etc.) and stack them vertically.

  Properties (read)

  `alignAboveContext` (string)
  : Where to insert newly created context in vertical alignment.

  `alignBelowContext` (string)
  : Where to insert newly created context in vertical alignment.

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  This engraver creates the following layout object(s): `VerticalAlignment` (page 514).

2.1.12 GregorianTranscriptionStaff

Handles clefs, bar lines, keys, accidentals. It can contain `Voice` contexts.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), `AccidentalSuggestion` (page 350), `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `KeyCancellation` (page 424), `KeySignature` (page 427), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `OttavaBracket` (page 457), `PianoPedalBracket`

(page 464), RestCollision (page 470), ScriptRow (page 472), SostenutoPedal (page 474), SostenutoPedalLineSpanner (page 475), StaffSpacing (page 479), StaffSymbol (page 480), SustainPedal (page 488), SustainPedalLineSpanner (page 489), TimeSignature (page 502), UnaCordaPedal (page 511), UnaCordaPedalLineSpanner (page 512), and VerticalAxisGroup (page 514).

This context sets the following properties:

- Set grob property hair-thickness in BarLine (page 358), to 1.9.
- Set grob property thick-thickness in BarLine (page 358), to 1.9.
- Set translator property createSpacing to #t.
- Set translator property defaultBarType to "".
- Set translator property ignoreFiguredBassRest to #f.
- Set translator property instrumentName to '().
- Set translator property localAlterations to '().
- Set translator property ottavationMarkups to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property shortInstrumentName to '().

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type GregorianTranscriptionVoice (page 111).

Context GregorianTranscriptionStaff can contain CueVoice (page 65), GregorianTranscriptionVoice (page 111), and NullVoice (page 168).

This context is built from the following engraver(s):

Accidental_engraver (page 282)

Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at Voice level, so you can \override them at Voice.

Properties (read)

accidentalGrouping (symbol)

If set to 'voice, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

autoAccidentals (list)

List of different ways to typeset an accidental.

For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

Each entry in the list is either a symbol or a procedure.

symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score"

in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

procedure The procedure represents an accidental rule to be applied to the previously specified context.

The procedure takes the following arguments:

`context` The current context to which the rule should be applied.

`pitch` The pitch of the note to be evaluated.

`barnum` The current bar number.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step . alter`*`)` or `((`*`octave . step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can

also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Alteration_glyph_engraver` (page 283)

Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)

Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)

Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)

Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)

True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)

A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)

True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)

Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)

This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)

If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)

Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)

Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)

Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
: Position figured bass alignments over notes.

This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
: Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of (*end-moment* . *grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of (*end-moment* . *grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
See `instrumentName`.

`shortVocalName` (markup)
Name of a vocal line, short version.

`vocalName` (markup)
Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Key_engraver` (page 305)
Engrave a key signature.

Music types accepted: `key-change-event` (page 48),

Properties (read)

`createKeyOnClefChange` (boolean)
Print a key signature whenever the clef is changed.

`explicitKeySignatureVisibility` (vector)
: '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(`*`step`*` . `*`alter`*`)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
: Engraver to merge multi-measure rest numbers in multiple voices.

  This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Ottava_spanner_engraver` (page 314)
: Create a text spanner when the ottavation property changes.

  Music types accepted: `ottava-event` (page 50),

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  `middleCOffset` (number)
  : The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

  `ottavation` (markup)
  : If set, the text for an ottava spanner. Changing this creates a new text spanner.

  This engraver creates the following layout object(s): `OttavaBracket` (page 457).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
: Align piano pedal symbols and brackets.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
: Engrave piano pedal symbols and brackets.

  Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `pedalSostenutoStrings` (list)
  : See `pedalSustainStrings`.

  `pedalSostenutoStyle` (symbol)
  : See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
: Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
: Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
: A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
: Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
: Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)
Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)
break visibility for the initial time signature.

`partialBusy` (boolean)
Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)
A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.13 `GregorianTranscriptionVoice`

Corresponds to a voice on a staff. This context handles the conversion of dynamic signs, stems, beams, super- and subscripts, slurs, ties, and rests.

You have to instantiate this explicitly if you want to have multiple voices on the same staff.

This context also accepts commands for the following context(s): `Voice` (page 271).

This context creates the following layout object(s): `Arpeggio` (page 355), `Beam` (page 366), `BendAfter` (page 368), `BreathingSign` (page 373), `ClusterSpanner` (page 382), `ClusterSpannerBeacon` (page 383), `CombineTextScript` (page 383), `Dots` (page 394), `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `Episema` (page 405), `FingerGlideSpanner` (page 406), `Fingering` (page 408), `Flag` (page 410), `Glissando` (page 414), `Hairpin` (page 417), `InstrumentSwitch` (page 421), `LaissezVibrerTie` (page 430), `LaissezVibrerTieColumn` (page 431), `LigatureBracket` (page 434), `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `NoteColumn` (page 454), `NoteHead` (page 455), `NoteSpacing` (page 456), `PercentRepeat` (page 460), `PercentRepeatCounter` (page 461), `PhrasingSlur` (page 462), `RepeatSlash` (page 467), `RepeatTie` (page 467), `RepeatTieColumn` (page 469), `Rest` (page 469), `Script` (page 470), `ScriptColumn` (page 471), `Slur` (page 472), `Stem` (page 481), `StemStub` (page 483), `StemTremolo` (page 484), `StringNumber` (page 485), `StrokeFinger` (page 487), `TextScript` (page 496), `TextSpanner` (page 498), `Tie` (page 500), `TieColumn` (page 501), `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), `TrillPitchHead` (page 506), `TrillSpanner` (page 507), `TupletBracket` (page 509), `TupletNumber` (page 510), and `VoiceFollower` (page 516).

This context sets the following properties:

- Set grob property `padding` in `Script` (page 470), to `0.5`.
- Set grob property `transparent` in `LigatureBracket` (page 434), to `#t`.
- Set translator property `autoBeaming` to `#f`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Arpeggio_engraver` (page 284)

Generate an Arpeggio symbol.

Music types accepted: `arpeggio-event` (page 45),

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Auto_beam_engraver` (page 284)

Generate beams based on measure characteristics and observed Stems. Uses `baseMoment`, `beatStructure`, `beamExceptions`, `measureLength`, and `measurePosition` to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [`Stem_engraver`], page 324, properties `stemLeftBeamCount` and `stemRightBeamCount`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)

If set to true then beams are generated automatically.

`baseMoment` (moment)

Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)

An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)

Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)

List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)

If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)

Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)
Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)
Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
Generate beams for tremolo repeats.

Music types accepted: `tremolo-span-event` (page 54),

This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
Engrave a cluster using `Spanner` notation.

Music types accepted: `cluster-note-event` (page 46),

This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
Make double measure repeats.

Music types accepted: `double-percent-event` (page 47),

Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`measureLength` (moment)
Length of one measure in the current time signature.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
Align hairpins and dynamic texts on a horizontal line.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Episema_engraver` (page 298)
Create an *Editio Vaticana*-style episema line.

Music types accepted: `episema-event` (page 47),

This engraver creates the following layout object(s): `Episema` (page 405).

`Finger_glide_engraver` (page 300)
Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)
Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)
Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
A queue of (*end-moment . grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
: Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
: Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
: Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
: The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
: Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Ligature_bracket_engraver` (page 307)

Handle `Ligature_events` by engraving `Ligature` brackets.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `LigatureBracket` (page 434).

`Multi_measure_rest_engraver` (page 311)

Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)

Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

  Properties (read)

  `followVoice` (boolean)
  : If set, note heads are tracked across staff switches by a thin line.

  This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
: Generate note heads.

  Music types accepted: `note-event` (page 50),

  Properties (read)

  `middleCPosition` (number)
  : The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

  `staffLineLayoutFunction` (procedure)
  : Layout of staff lines, `traditional`, or `semitone`.

  This engraver creates the following layout object(s): `NoteHead` (page 455).

`Note_spacing_engraver` (page 313)
: Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

  This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings ‘a2’, ‘Solo’, ‘Solo II’, and ‘unisono’.

  Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

  Properties (read)

  `aDueText` (markup)
  : Text to print at a unisono passage.

  `partCombineTextsOnNote` (boolean)
  : Print part-combine texts only on the next note rather than immediately on rests or skips.

  `printPartCombineTexts` (boolean)
  : Set ‘Solo’ and ‘A due’ texts in the part combiner?

  `soloIIText` (markup)
  : The text for the start of a solo for voice ‘two’ when part-combining.

  `soloText` (markup)
  : The text for the start of a solo when part-combining.

This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
Make whole measure repeats.

Music types accepted: `percent-event` (page 51),

Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)
Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)
Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)
The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
: Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
: Handle note scripted articulations.

  Music types accepted: `articulation-event` (page 45),

  Properties (read)

  `scriptDefinitions` (list)
  : The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

  This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
: Make beat repeats.

  Music types accepted: `repeat-slash-event` (page 51),

  This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
: Build slur grobs from slur events.

  Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

  Properties (read)

  `doubleSlurs` (boolean)
  : If set, two slurs are created for every slurred note, one above and one below the chord.

  `slurMelismaBusy` (boolean)
  : Signal if a slur is present.

  This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
: Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
: Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

  Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

  Properties (read)

  `stemLeftBeamCount` (integer)
  : Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

  `stemRightBeamCount` (integer)
  : See `stemLeftBeamCount`.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

  ```
  \set Staff.whichBar = ".|:"
  ```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
: Create text scripts.

  Music types accepted: `text-script-event` (page 54),

  This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
: Create text spanner from an event.

  Music types accepted: `text-span-event` (page 54),

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)
: Generate ties between note heads of equal pitch.

  Music types accepted: `tie-event` (page 54),

  Properties (read)

  `skipTypesetting` (boolean)
  : If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

  `tieWaitForNote` (boolean)
  : If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

  Properties (write)

  `tieMelismaBusy` (boolean)
  : Signal whether a tie is present.

  This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)
: Create trill spanner from an event.

  Music types accepted: `trill-span-event` (page 54),

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)
Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.14 KievanStaff

Same as `Staff` context, except that it is accommodated for typesetting a piece in Kievan style.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), `AccidentalSuggestion` (page 350), `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `KeyCancellation` (page 424), `KeySignature` (page 427), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `OttavaBracket` (page 457), `PianoPedalBracket` (page 464), `RestCollision` (page 470), `ScriptRow` (page 472), `SostenutoPedal` (page 474), `SostenutoPedalLineSpanner` (page 475), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `SustainPedal` (page 488), `SustainPedalLineSpanner` (page 489), `UnaCordaPedal` (page 511), `UnaCordaPedalLineSpanner` (page 512), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set translator property `autoAccidentals` to:

```
'(Staff #<procedure #f (context pitch barnum)>
     #<procedure neo-modern-accidental-rule (context pitch barnum)>)
```

- Set translator property `autoCautionaries` to `'()`.
- Set translator property `clefGlyph` to `"clefs.kievan.do"`.
- Set translator property `clefPosition` to `0`.
- Set translator property `clefTransposition` to `0`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `extraNatural` to `#f`.
- Set translator property `ignoreFiguredBassRest` to `#f`.
- Set translator property `instrumentName` to `'()`.

- Set translator property `localAlterations` to `'()`.
- Set translator property `middleCClefPosition` to `0`.
- Set translator property `middleCPosition` to `0`.
- Set translator property `ottavationMarkups` to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property `printKeyCancellation` to `#f`.
- Set translator property `shortInstrumentName` to `'()`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `KievanVoice` (page 132).

Context `KievanStaff` can contain `CueVoice` (page 65), `KievanVoice` (page 132), and `NullVoice` (page 168).

This context is built from the following engraver(s):

`Accidental_engraver` (page 282)

Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at `Voice` level, so you can `\override` them at `Voice`.

Properties (read)

`accidentalGrouping` (symbol)

If set to `'voice`, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

`autoAccidentals` (list)

List of different ways to typeset an accidental.

For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

Each entry in the list is either a symbol or a procedure.

symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

procedure The procedure represents an accidental rule to be applied to the previously specified context.

The procedure takes the following arguments:

`context` The current context to which the rule should be applied.

`pitch` The pitch of the note to be evaluated.

`barnum` The current bar number.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave`*` . `*`name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave`*` . `*`name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Alteration_glyph_engraver` (page 283)
: Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
: Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

  This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

  Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

  Properties (read)

  `figuredBassAlterationDirection` (direction)
  : Where to put alterations relative to the main figure.

  `figuredBassCenterContinuations` (boolean)
  : Whether to vertically center pairs of extender lines. This does not work with three or more lines.

  `figuredBassFormatter` (procedure)
  : A routine generating a markup for a bass figure.

  `ignoreFiguredBassRest` (boolean)
  : Don't swallow rest events.

  `implicitBassFigures` (list)
  : A list of bass figures that are not printed as numbers, but only as extender lines.

  `useBassFigureExtenders` (boolean)
  : Whether to use extender lines for repeated bass figures.

  This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
: Position figured bass alignments over notes.

  This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
: Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

  Properties (read)

  `fontSize` (number)
  : The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Key_engraver` (page 305)
: Engrave a key signature.

Music types accepted: `key-change-event` (page 48),

Properties (read)

`createKeyOnClefChange` (boolean)
: Print a key signature whenever the clef is changed.

`explicitKeySignatureVisibility` (vector)
: '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(`*`step .`*

alter), where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing (*step* . *alter*) or ((*octave* . *step*) . *alter*), where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing (*step* . *alter*) or ((*octave* . *step*) . *alter*), where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

  This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
: Engraver to merge multi-measure rest numbers in multiple voices.

  This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Ottava_spanner_engraver` (page 314)
: Create a text spanner when the ottavation property changes.

  Music types accepted: `ottava-event` (page 50),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`middleCOffset` (number)
: The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`ottavation` (markup)
: If set, the text for an ottava spanner. Changing this creates a new text spanner.

This engraver creates the following layout object(s): `OttavaBracket` (page 457).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
: Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
: Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
A queue of (*end-moment* . *grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

2.1.15 KievanVoice

Same as Voice context, except that it is accommodated for typesetting a piece in Kievan style.

This context also accepts commands for the following context(s): Voice (page 271).

This context creates the following layout object(s): Arpeggio (page 355), Beam (page 366), BendAfter (page 368), BreathingSign (page 373), ClusterSpanner (page 382), ClusterSpannerBeacon (page 383), CombineTextScript (page 383), Dots (page 394), DoublePercentRepeat (page 395), DoublePercentRepeatCounter (page 396), DoubleRepeatSlash (page 398), DynamicLineSpanner (page 401), DynamicText (page 402), DynamicTextSpanner (page 403), FingerGlideSpanner (page 406), Fingering (page 408), Flag (page 410), Glissando (page 414), Hairpin (page 417), InstrumentSwitch (page 421), KievanLigature (page 430), LaissezVibrerTie (page 430), LaissezVibrerTieColumn (page 431), MultiMeasureRest (page 446), MultiMeasureRestNumber (page 447), MultiMeasureRestScript (page 449), MultiMeasureRestText (page 450), NoteColumn (page 454), NoteHead (page 455), NoteSpacing (page 456), PercentRepeat (page 460), PercentRepeatCounter (page 461), PhrasingSlur (page 462), RepeatSlash (page 467), RepeatTie (page 467), RepeatTieColumn (page 469), Rest (page 469), Script (page 470), ScriptColumn (page 471), Slur (page 472), Stem (page 481), StemStub (page 483), StemTremolo (page 484), StringNumber (page 485), StrokeFinger (page 487), TextScript (page 496), TextSpanner (page 498), Tie (page 500), TieColumn (page 501), TrillPitchAccidental (page 504), TrillPitchGroup (page 505), TrillPitchHead (page 506), TrillSpanner (page 507), TupletBracket (page 509), TupletNumber (page 510), and VoiceFollower (page 516).

This context sets the following properties:

- Set grob property duration-log in NoteHead (page 455), to note-head::calc-kievan-duration-log.
- Set grob property length in Stem (page 481), to 0.0.
- Set grob property positions in Beam (page 366), to beam::get-kievan-positions.
- Set grob property quantized-positions in Beam (page 366), to beam::get-kievan-quantized-positions.
- Set grob property stencil in Flag (page 410), to #f.
- Set grob property stencil in Slur (page 472), to #f.
- Set grob property stencil in Stem (page 481), to #f.
- Set grob property style in Dots (page 394), to 'kievan.
- Set grob property style in NoteHead (page 455), to 'kievan.
- Set grob property style in Rest (page 469), to 'mensural.
- Set grob property X-offset in Stem (page 481), to stem::kievan-offset-callback.
- Set translator property alterationGlyphs to:

```
'((-1/2 . "accidentals.kievanM1")
  (1/2 . "accidentals.kievan1"))
```

- Set translator property autoBeaming to #f.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

Arpeggio_engraver (page 284)

Generate an Arpeggio symbol.

Music types accepted: arpeggio-event (page 45),

This engraver creates the following layout object(s): Arpeggio (page 355).

`Auto_beam_engraver` (page 284)

Generate beams based on measure characteristics and observed Stems. Uses `baseMoment`, `beatStructure`, `beamExceptions`, `measureLength`, and `measurePosition` to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [`Stem_engraver`], page 324, properties `stemLeftBeamCount` and `stemRightBeamCount`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
: If set to true then beams are generated automatically.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)
Create fall spanners.
Music types accepted: `bend-after-event` (page 46),
This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)
Create a breathing sign.
Music types accepted: `breathing-event` (page 46),
This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
Generate beams for tremolo repeats.
Music types accepted: `tremolo-span-event` (page 54),
This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
Engrave a cluster using `Spanner` notation.
Music types accepted: `cluster-note-event` (page 46),
This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.
This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
Make double measure repeats.
Music types accepted: `double-percent-event` (page 47),
Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`measureLength` (moment)
Length of one measure in the current time signature.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
Align hairpins and dynamic texts on a horizontal line.
Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)
Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)
Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)
Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
Forbid line breaks when note heads are still playing at some point.

Properties (read)

busyGrobs (list)
A queue of (*end-moment . grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

forbidBreak (boolean)
If set to #t, prevent a line break at this point.

Glissando_engraver (page 302)
Engrave glissandi.

Music types accepted: glissando-event (page 48),

Properties (read)

glissandoMap (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): Glissando (page 414).

Grace_auto_beam_engraver (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or \noBeam will block autobeaming, just like setting the context property 'autoBeaming' to ##f.

Music types accepted: beam-forbid-event (page 46),

Properties (read)

autoBeaming (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): Beam (page 366).

Grace_beam_engraver (page 302)
Handle Beam events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: beam-event (page 46),

Properties (read)

baseMoment (moment)
Smallest unit of time that will stand on its own as a subdivided section.

beamMelismaBusy (boolean)
Signal if a beam is present.

beatStructure (list)
List of baseMoments that are combined to make beats.

subdivideBeams (boolean)
If set, multiple beams will be subdivided at baseMoment positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): Beam (page 366).

`Grace_engraver` (page 303)
Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Kievan_ligature_engraver` (page 307)
Handle `Kievan_ligature_events` by glueing Kievan heads together.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `KievanLigature` (page 430).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)
: Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
: Generate note heads.

Music types accepted: `note-event` (page 50),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

staffLineLayoutFunction (procedure)
: Layout of staff lines, traditional, or semitone.

This engraver creates the following layout object(s): NoteHead (page 455).

Note_spacing_engraver (page 313)
: Generate NoteSpacing, an object linking horizontal lines for use in spacing.

This engraver creates the following layout object(s): NoteSpacing (page 456).

Output_property_engraver (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: apply-output-event (page 45),

Part_combine_engraver (page 315)
: Part combine engraver for orchestral scores: Print markings 'a2', 'Solo', 'Solo II', and 'unisono'.

Music types accepted: note-event (page 50), and part-combine-event (page 50),

Properties (read)

aDueText (markup)
: Text to print at a unisono passage.

partCombineTextsOnNote (boolean)
: Print part-combine texts only on the next note rather than immediately on rests or skips.

printPartCombineTexts (boolean)
: Set 'Solo' and 'A due' texts in the part combiner?

soloIIText (markup)
: The text for the start of a solo for voice 'two' when part-combining.

soloText (markup)
: The text for the start of a solo when part-combining.

This engraver creates the following layout object(s): CombineTextScript (page 383).

Percent_repeat_engraver (page 316)
: Make whole measure repeats.

Music types accepted: percent-event (page 51),

Properties (read)

countPercentRepeats (boolean)
: If set, produce counters for percent repeats.

currentCommandColumn (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

repeatCountVisibility (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when countPercentRepeats is set.

This engraver creates the following layout object(s): PercentRepeat (page 460), and PercentRepeatCounter (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

  Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

  This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)
: Print the bracketed note head after a note head with trill.

  This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)
: Create repeat ties.

  Music types accepted: `repeat-tie-event` (page 51),

  This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
: Engrave rests.

  Music types accepted: `rest-event` (page 51),

  Properties (read)

  `middleCPosition` (number)
  : The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

  This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
: Generate `NoteColumn`, an object that groups stems, note heads, and rests.

  This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
: Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
: Handle note scripted articulations.

  Music types accepted: `articulation-event` (page 45),

  Properties (read)

  `scriptDefinitions` (list)
  : The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

  This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
: Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
Build slur grobs from slur events.

Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

Properties (read)

`doubleSlurs` (boolean)
If set, two slurs are created for every slurred note, one above and one below the chord.

`slurMelismaBusy` (boolean)
Signal if a slur is present.

This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
See `stemLeftBeamCount`.

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
Create text scripts.

Music types accepted: `text-script-event` (page 54),

This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
Create text spanner from an event.

Music types accepted: `text-span-event` (page 54),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)

Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)

Create trill spanner from an event.

Music types accepted: `trill-span-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)

Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.16 Lyrics

Corresponds to a voice with lyrics. Handles the printing of a single line of lyrics.

This context creates the following layout object(s): `InstrumentName` (page 420), `LyricExtender` (page 436), `LyricHyphen` (page 436), `LyricSpace` (page 437), `LyricText` (page 438), `StanzaNumber` (page 481), `VerticalAxisGroup` (page 514), and `VowelTransition` (page 520).

This context sets the following properties:

- Set grob property `bar-extent` in `BarLine` (page 358), to:

  ```
  '(-0.05 . 0.05)
  ```

- Set grob property `font-size` in `InstrumentName` (page 420), to `1.0`.
- Set grob property `nonstaff-nonstaff-spacing` in `VerticalAxisGroup` (page 514), to:

  ```
  '((basic-distance . 0)
   (minimum-distance . 2.8)
   (padding . 0.2)
   (stretchability . 0))
  ```

- Set grob property `nonstaff-relatedstaff-spacing` in `VerticalAxisGroup` (page 514), to:

  ```
  '((basic-distance . 5.5)
   (padding . 0.5)
   (stretchability . 1))
  ```

- Set grob property `nonstaff-unrelatedstaff-spacing.padding` in `VerticalAxisGroup` (page 514), to `1.5`.
- Set grob property `remove-empty` in `VerticalAxisGroup` (page 514), to `#t`.
- Set grob property `remove-first` in `VerticalAxisGroup` (page 514), to `#t`.
- Set grob property `self-alignment-Y` in `InstrumentName` (page 420), to `#f`.
- Set grob property `staff-affinity` in `VerticalAxisGroup` (page 514), to `1`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `searchForVoice` to `#f`.
- Set translator property `shortInstrumentName` to `'()`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Axis_group_engraver` (page 285)

Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Extender_engraver` (page 299)
: Create lyric extenders.

Music types accepted: `completize-extender-event` (page 47), and `extender-event` (page 47),

Properties (read)

`extendersOverRests` (boolean)
: Whether to continue extenders as they cross a rest.

This engraver creates the following layout object(s): `LyricExtender` (page 436).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Hyphen_engraver` (page 304)
: Create lyric hyphens, vowel transitions and distance constraints between words.

Music types accepted: `hyphen-event` (page 48), and `vowel-transition-event` (page 55),

This engraver creates the following layout object(s): `LyricHyphen` (page 436), `LyricSpace` (page 437), and `VowelTransition` (page 520).

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Lyric_engraver` (page 307)
: Engrave text for lyrics.

  Music types accepted: `lyric-event` (page 48),

  Properties (read)

  `ignoreMelismata` (boolean)
  : Ignore melismata for this Section "Lyrics" in *Internals Reference* line.

  `lyricMelismaAlignment` (number)
  : Alignment to use for a melisma syllable.

  `searchForVoice` (boolean)
  : Signal whether a search should be made of all contexts in the context hierarchy for a voice to provide rhythms for the lyrics.

  This engraver creates the following layout object(s): `LyricText` (page 438).

`Pure_from_neighbor_engraver` (page 318)
: Coordinates items that get their pure heights from their neighbors.

`Stanza_number_engraver` (page 324)
: Engrave stanza numbers.

  Properties (read)

  `stanza` (markup)
  : Stanza 'number' to print before the start of a verse. Use in `Lyrics` context.

  This engraver creates the following layout object(s): `StanzaNumber` (page 481).

2.1.17 MensuralStaff

Same as `Staff` context, except that it is accommodated for typesetting a piece in mensural style.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), `AccidentalSuggestion` (page 350), `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `Custos` (page 392), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `KeyCancellation` (page 424), `KeySignature` (page 427), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `OttavaBracket` (page 457), `PianoPedalBracket` (page 464), `RestCollision` (page 470), `ScriptRow` (page 472), `SostenutoPedal` (page 474), `SostenutoPedalLineSpanner` (page 475), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `SustainPedal` (page 488), `SustainPedalLineSpanner` (page 489), `TimeSignature` (page 502), `UnaCordaPedal` (page 511), `UnaCordaPedalLineSpanner` (page 512), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `hair-thickness` in `BarLine` (page 358), to `0.6`.
- Set grob property `neutral-direction` in `Custos` (page 392), to `-1`.
- Set grob property `neutral-position` in `Custos` (page 392), to `3`.

- Set grob property `style` in `Custos` (page 392), to `'mensural`.
- Set grob property `style` in `TimeSignature` (page 502), to `'mensural`.
- Set grob property `thick-thickness` in `BarLine` (page 358), to `0.6`.
- Set grob property `thickness` in `StaffSymbol` (page 480), to `0.6`.
- Set translator property `alterationGlyphs` to:

```
'((-1/2 . "accidentals.mensuralM1")
  (0 . "accidentals.vaticana0")
  (1/2 . "accidentals.mensural1"))
```

- Set translator property `autoAccidentals` to:

```
'(Staff #<procedure #f (context pitch barnum)>)
```

- Set translator property `autoCautionaries` to `'()`.
- Set translator property `clefGlyph` to `"clefs.mensural.g"`.
- Set translator property `clefPosition` to `-2`.
- Set translator property `clefTransposition` to `0`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `defaultBarType` to `""`.
- Set translator property `extraNatural` to `#f`.
- Set translator property `ignoreFiguredBassRest` to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `middleCClefPosition` to `-6`.
- Set translator property `middleCPosition` to `-6`.
- Set translator property `ottavationMarkups` to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property `printKeyCancellation` to `#f`.
- Set translator property `shortInstrumentName` to `'()`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `MensuralVoice` (page 155).

Context `MensuralStaff` can contain `CueVoice` (page 65), `MensuralVoice` (page 155), and `NullVoice` (page 168).

This context is built from the following engraver(s):

`Accidental_engraver` (page 282)

Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at `Voice` level, so you can `\override` them at `Voice`.

Properties (read)

`accidentalGrouping` (symbol)

If set to `'voice`, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

`autoAccidentals` (list)
: List of different ways to typeset an accidental.

  For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

  Each entry in the list is either a symbol or a procedure.

  symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

  procedure The procedure represents an accidental rule to be applied to the previously specified context.

  The procedure takes the following arguments:

  `context` The current context to which the rule should be applied.

  `pitch` The pitch of the note to be evaluated.

  `barnum` The current bar number.

  The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step . alter`*`)` or `((`*`octave . step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*octave* `.` *name*`) . (`*alter barnumber* `.` *measureposition*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*octave* `.` *name*`) . (`*alter barnumber* `.` *measureposition*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Alteration_glyph_engraver` (page 283)
: Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
: Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Custos_engraver` (page 295)
: Engrave custodes.

This engraver creates the following layout object(s): `Custos` (page 392).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
Position figured bass alignments over notes.

This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
See `instrumentName`.

`shortVocalName` (markup)
Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Key_engraver` (page 305)
: Engrave a key signature.

Music types accepted: `key-change-event` (page 48),

Properties (read)

`createKeyOnClefChange` (boolean)
: Print a key signature whenever the clef is changed.

`explicitKeySignatureVisibility` (vector)
: '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(`*`step`*` . `*`alter`*`)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
: Engraver to merge multi-measure rest numbers in multiple voices.

This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Ottava_spanner_engraver` (page 314)
: Create a text spanner when the ottavation property changes.

Music types accepted: `ottava-event` (page 50),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`middleCOffset` (number)
: The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`ottavation` (markup)
: If set, the text for an ottava spanner. Changing this creates a new text spanner.

This engraver creates the following layout object(s): `OttavaBracket` (page 457).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
: Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
See `pedalSustainStyle`.

`pedalSustainStrings` (list)
A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)
Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)
break visibility for the initial time signature.

`partialBusy` (boolean)
Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)
A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.18 MensuralVoice

Same as `Voice` context, except that it is accommodated for typesetting a piece in mensural style.

This context also accepts commands for the following context(s): `Voice` (page 271).

This context creates the following layout object(s): `Arpeggio` (page 355), `Beam` (page 366), `BendAfter` (page 368), `BreathingSign` (page 373), `ClusterSpanner` (page 382), `ClusterSpannerBeacon` (page 383), `CombineTextScript` (page 383), `Dots` (page 394), `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `FingerGlideSpanner` (page 406), `Fingering` (page 408), `Flag` (page 410), `Glissando` (page 414), `Hairpin` (page 417), `InstrumentSwitch` (page 421), `LaissezVibrerTie` (page 430), `LaissezVibrerTieColumn` (page 431), `MensuralLigature`

(page 444), MultiMeasureRest (page 446), MultiMeasureRestNumber (page 447), MultiMeasureRestScript (page 449), MultiMeasureRestText (page 450), NoteColumn (page 454), NoteHead (page 455), NoteSpacing (page 456), PercentRepeat (page 460), PercentRepeatCounter (page 461), PhrasingSlur (page 462), RepeatSlash (page 467), RepeatTie (page 467), RepeatTieColumn (page 469), Rest (page 469), Script (page 470), ScriptColumn (page 471), Stem (page 481), StemStub (page 483), StemTremolo (page 484), StringNumber (page 485), StrokeFinger (page 487), TextScript (page 496), TextSpanner (page 498), Tie (page 500), TieColumn (page 501), TrillPitchAccidental (page 504), TrillPitchGroup (page 505), TrillPitchHead (page 506), TrillSpanner (page 507), TupletBracket (page 509), TupletNumber (page 510), and VoiceFollower (page 516).

This context sets the following properties:

- Set grob property style in Flag (page 410), to 'mensural.
- Set grob property style in NoteHead (page 455), to 'mensural.
- Set grob property style in Rest (page 469), to 'mensural.
- Set translator property autoBeaming to #f.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

Arpeggio_engraver (page 284)

Generate an Arpeggio symbol.

Music types accepted: arpeggio-event (page 45),

This engraver creates the following layout object(s): Arpeggio (page 355).

Auto_beam_engraver (page 284)

Generate beams based on measure characteristics and observed Stems. Uses baseMoment, beatStructure, beamExceptions, measureLength, and measurePosition to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [Stem_engraver], page 324, properties stemLeftBeamCount and stemRightBeamCount.

Music types accepted: beam-forbid-event (page 46),

Properties (read)

autoBeaming (boolean)

If set to true then beams are generated automatically.

baseMoment (moment)

Smallest unit of time that will stand on its own as a subdivided section.

beamExceptions (list)

An alist of exceptions to autobeam rules that normally end on beats.

beamHalfMeasure (boolean)

Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

beatStructure (list)

List of baseMoments that are combined to make beats.

subdivideBeams (boolean)

If set, multiple beams will be subdivided at baseMoment positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)
Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)
Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
Generate beams for tremolo repeats.

Music types accepted: `tremolo-span-event` (page 54),

This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
Engrave a cluster using `Spanner` notation.

Music types accepted: `cluster-note-event` (page 46),

This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
Make double measure repeats.

Music types accepted: `double-percent-event` (page 47),

Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`measureLength` (moment)
Length of one measure in the current time signature.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
Align hairpins and dynamic texts on a horizontal line.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)
: Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)
: Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
: Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
: A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
: Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
: A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
: Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Mensural_ligature_engraver` (page 310)
Handle `Mensural_ligature_events` by glueing special ligature heads together.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `MensuralLigature` (page 444).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
True at the beginning of a measure.

`restNumberThreshold` (number)
If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)
Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
: Generate note heads.

Music types accepted: `note-event` (page 50),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

This engraver creates the following layout object(s): `NoteHead` (page 455).

`Note_spacing_engraver` (page 313)
: Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings ‘a2’, ‘Solo’, ‘Solo II’, and ‘unisono’.

Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

Properties (read)

`aDueText` (markup)
: Text to print at a unisono passage.

`partCombineTextsOnNote` (boolean)
: Print part-combine texts only on the next note rather than immediately on rests or skips.

`printPartCombineTexts` (boolean)
: Set 'Solo' and 'A due' texts in the part combiner?

`soloIIText` (markup)
: The text for the start of a solo for voice 'two' when part-combining.

`soloText` (markup)
: The text for the start of a solo when part-combining.

This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
: Make whole measure repeats.

Music types accepted: `percent-event` (page 51),

Properties (read)

`countPercentRepeats` (boolean)
: If set, produce counters for percent repeats.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`repeatCountVisibility` (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)
: Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)
: Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
: Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
: Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
: Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
: Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)
: The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
: Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Spanner_break_forbid_engraver` (page 323)
: Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
: Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
: Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
: See `stemLeftBeamCount`.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

  ```
  \set Staff.whichBar = ".|:"
  ```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
: Create text scripts.

  Music types accepted: `text-script-event` (page 54),

  This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
: Create text spanner from an event.

  Music types accepted: `text-span-event` (page 54),

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)
: Generate ties between note heads of equal pitch.

  Music types accepted: `tie-event` (page 54),

  Properties (read)

  `skipTypesetting` (boolean)
  : If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

  `tieWaitForNote` (boolean)
  : If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

  Properties (write)

  `tieMelismaBusy` (boolean)
  : Signal whether a tie is present.

  This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)
: Create trill spanner from an event.

  Music types accepted: `trill-span-event` (page 54),

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)
Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.19 NoteNames

A context for printing the names of notes.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `NoteName` (page 456), `StaffSpacing` (page 479), `Tie` (page 500), `TieColumn` (page 501), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `nonstaff-nonstaff-spacing` in `VerticalAxisGroup` (page 514), to:

```
'((basic-distance . 0)
 (minimum-distance . 2.8)
 (padding . 0.2)
 (stretchability . 0))
```

- Set grob property `nonstaff-relatedstaff-spacing` in `VerticalAxisGroup` (page 514), to:

```
'((basic-distance . 5.5)
 (padding . 0.5)
 (stretchability . 1))
```

- Set grob property `nonstaff-unrelatedstaff-spacing.padding` in `VerticalAxisGroup` (page 514), to `1.5`.
- Set grob property `staff-affinity` in `VerticalAxisGroup` (page 514), to `1`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Alteration_glyph_engraver` (page 283)
Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Note_name_engraver` (page 313)
: Print pitches as words.

Music types accepted: `note-event` (page 50),

Properties (read)

`noteNameFunction` (procedure)
: Function used to convert pitches into strings and markups.

`noteNameSeparator` (string)
: String used to separate simultaneous NoteName objects.

`printAccidentalNames` (boolean or symbol)
: Print accidentals in the `NoteNames` context.

`printNotesLanguage` (string)
: Use a specific language in the `NoteNames` context.

`printOctaveNames` (boolean or symbol)
: Print octave marks in the `NoteNames` context.

This engraver creates the following layout object(s): `NoteName` (page 456).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Tie_engraver` (page 327)
: Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

2.1.20 `NullVoice`

For aligning lyrics without printing notes

This context also accepts commands for the following context(s): `Staff` (page 220), and `Voice` (page 271).

This context creates the following layout object(s): `Beam` (page 366), `NoteHead` (page 455), `Slur` (page 472), `Tie` (page 500), and `TieColumn` (page 501).

This context sets the following properties:

- Set grob property `no-ledgers` in `NoteHead` (page 455), to `#t`.
- Set grob property `stencil` in `Beam` (page 366), to `#f`.
- Set grob property `stencil` in `NoteHead` (page 455), to `#f`.
- Set grob property `stencil` in `Slur` (page 472), to `#f`.
- Set grob property `stencil` in `Tie` (page 500), to `#f`.
- Set grob property `X-extent` in `NoteHead` (page 455), to `#<procedure #f (g)>`.
- Set translator property `nullAccidentals` to `#t`.
- Set translator property `squashedPosition` to `0`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Beam_engraver` (page 287)
: Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Note_heads_engraver` (page 313)
: Generate note heads.

Music types accepted: `note-event` (page 50),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

This engraver creates the following layout object(s): `NoteHead` (page 455).

`Pitch_squash_engraver` (page 317)
: Set the vertical position of note heads to `squashedPosition`, if that property is set. This can be used to make a single-line staff demonstrating the rhythm of a melody.

Properties (read)

`squashedPosition` (integer)
: Vertical position of squashing for Section "Pitch_squash_engraver" in *Internals Reference*.

`Slur_engraver` (page 322)

Build slur grobs from slur events.

Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

Properties (read)

`doubleSlurs` (boolean)

If set, two slurs are created for every slurred note, one above and one below the chord.

`slurMelismaBusy` (boolean)

Signal if a slur is present.

This engraver creates the following layout object(s): `Slur` (page 472).

`Tie_engraver` (page 327)

Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)

If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)

If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)

Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

2.1.21 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.

This context creates the following layout object(s): `VerticalAxisGroup` (page 514).

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Staff` (page 220).

Context `OneStaff` can contain `ChordNames` (page 63), `DrumStaff` (page 76), `Dynamics` (page 92), `FiguredBass` (page 95), `FretBoards` (page 97), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

This context is built from the following engraver(s):

`Axis_group_engraver` (page 285)

Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)

Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

2.1.22 PetrucciStaff

Same as `Staff` context, except that it is accommodated for typesetting a piece in Petrucci style.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), `AccidentalSuggestion` (page 350), `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `Custos` (page 392), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `KeyCancellation` (page 424), `KeySignature` (page 427), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `OttavaBracket` (page 457), `PianoPedalBracket` (page 464), `RestCollision` (page 470), `ScriptRow` (page 472), `SostenutoPedal` (page 474), `SostenutoPedalLineSpanner` (page 475), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `SustainPedal` (page 488), `SustainPedalLineSpanner` (page 489), `TimeSignature` (page 502), `UnaCordaPedal` (page 511), `UnaCordaPedalLineSpanner` (page 512), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `neutral-direction` in `Custos` (page 392), to `-1`.
- Set grob property `neutral-position` in `Custos` (page 392), to `3`.
- Set grob property `style` in `Custos` (page 392), to `'mensural`.
- Set grob property `thickness` in `StaffSymbol` (page 480), to `1.3`.
- Set translator property `autoAccidentals` to:

```
'(Staff #<procedure #f (context pitch barnum)>
     #<procedure neo-modern-accidental-rule (context pitch barnum)>)
```

- Set translator property `autoCautionaries` to `'()`.
- Set translator property `clefGlyph` to `"clefs.petrucci.g"`.
- Set translator property `clefPosition` to `-2`.
- Set translator property `clefTransposition` to `0`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `extraNatural` to `#f`.
- Set translator property `ignoreFiguredBassRest` to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `middleCClefPosition` to `-6`.

- Set translator property `middleCPosition` to `-6`.
- Set translator property `ottavationMarkups` to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property `printKeyCancellation` to `#f`.
- Set translator property `shortInstrumentName` to `'()`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `PetrucciVoice` (page 181).

Context `PetrucciStaff` can contain `CueVoice` (page 65), `NullVoice` (page 168), and `PetrucciVoice` (page 181).

This context is built from the following engraver(s):

`Accidental_engraver` (page 282)

Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at `Voice` level, so you can `\override` them at `Voice`.

Properties (read)

`accidentalGrouping` (symbol)

If set to `'voice`, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

`autoAccidentals` (list)

List of different ways to typeset an accidental.

For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

Each entry in the list is either a symbol or a procedure.

symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

procedure The procedure represents an accidental rule to be applied to the previously specified context.

The procedure takes the following arguments:

`context` The current context to which the rule should be applied.

`pitch` The pitch of the note to be evaluated.

`barnum` The current bar number.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave`*` . `*`name`*`) . (`*`alter barnumber`*` . `*`measureposition`*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave`*` . `*`name`*`) . (`*`alter barnumber`*` . `*`measureposition`*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Alteration_glyph_engraver` (page 283)
: Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  `keepAliveInterfaces` (list)
  : A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

  Properties (write)

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

  Properties (read)

  `whichBar` (string)
  : This property is read to determine what type of bar line to create.

    Example:

    ```
    \set Staff.whichBar = ".|:"
    ```

    This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

  Properties (write)

  `forbidBreak` (boolean)
  : If set to `#t`, prevent a line break at this point.

  This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
: Determine and set reference point for pitches.

  Properties (read)

  `clefGlyph` (string)
  : Name of the symbol within the music font.

  `clefPosition` (number)
  : Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Custos_engraver` (page 295)
: Engrave custodes.

  This engraver creates the following layout object(s): `Custos` (page 392).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

  This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

  Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

  Properties (read)

  `figuredBassAlterationDirection` (direction)
  : Where to put alterations relative to the main figure.

  `figuredBassCenterContinuations` (boolean)
  : Whether to vertically center pairs of extender lines. This does not work with three or more lines.

  `figuredBassFormatter` (procedure)
  : A routine generating a markup for a bass figure.

  `ignoreFiguredBassRest` (boolean)
  : Don't swallow rest events.

  `implicitBassFigures` (list)
  : A list of bass figures that are not printed as numbers, but only as extender lines.

  `useBassFigureExtenders` (boolean)
  : Whether to use extender lines for repeated bass figures.

  This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
: Position figured bass alignments over notes.

  This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
: Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

  Properties (read)

  `fontSize` (number)
  : The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
See `instrumentName`.

`shortVocalName` (markup)
Name of a vocal line, short version.

`vocalName` (markup)
Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Key_engraver` (page 305)
Engrave a key signature.

Music types accepted: `key-change-event` (page 48),

Properties (read)

`createKeyOnClefChange` (boolean)
Print a key signature whenever the clef is changed.

`explicitKeySignatureVisibility` (vector)
'`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(step . alter)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

  This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
: Engraver to merge multi-measure rest numbers in multiple voices.

  This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Ottava_spanner_engraver` (page 314)
: Create a text spanner when the ottavation property changes.

Music types accepted: `ottava-event` (page 50),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`middleCOffset` (number)
: The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`ottavation` (markup)
: If set, the text for an ottava spanner. Changing this creates a new text spanner.

This engraver creates the following layout object(s): `OttavaBracket` (page 457).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
: Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
: Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
: Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
: Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
: Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
: Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
: Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)
: Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

  Music types accepted: `time-signature-event` (page 54),

  Properties (read)

  `initialTimeSignatureVisibility` (vector)
  : break visibility for the initial time signature.

  `partialBusy` (boolean)
  : Signal that \partial acts at the current timestep.

  `timeSignatureFraction` (fraction, as pair)
  : A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

  This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.23 PetrucciVoice

Same as `Voice` context, except that it is accommodated for typesetting a piece in Petrucci style.

This context also accepts commands for the following context(s): `Voice` (page 271).

This context creates the following layout object(s): `Arpeggio` (page 355), `Beam` (page 366), `BendAfter` (page 368), `BreathingSign` (page 373), `ClusterSpanner` (page 382), `ClusterSpannerBeacon` (page 383), `CombineTextScript` (page 383), `Dots` (page 394), `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `FingerGlideSpanner` (page 406), `Fingering` (page 408), `Flag` (page 410), `Glissando` (page 414), `Hairpin` (page 417), `InstrumentSwitch` (page 421), `LaissezVibrerTie` (page 430), `LaissezVibrerTieColumn` (page 431), `MensuralLigature` (page 444), `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `NoteColumn` (page 454), `NoteHead` (page 455), `NoteSpacing` (page 456), `PercentRepeat` (page 460), `PercentRepeatCounter` (page 461), `PhrasingSlur` (page 462), `RepeatSlash` (page 467), `RepeatTie` (page 467), `RepeatTieColumn` (page 469), `Rest` (page 469), `Script` (page 470), `ScriptColumn` (page 471), `Slur` (page 472), `Stem` (page 481), `StemStub` (page 483), `StemTremolo` (page 484), `StringNumber` (page 485), `StrokeFinger` (page 487), `TextScript` (page 496), `TextSpanner` (page 498), `Tie` (page 500), `TieColumn` (page 501), `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), `TrillPitchHead` (page 506), `TrillSpanner` (page 507), `TupletBracket` (page 509), `TupletNumber` (page 510), and `VoiceFollower` (page 516).

This context sets the following properties:

- Set grob property `length` in `Stem` (page 481), to `5`.
- Set grob property `style` in `NoteHead` (page 455), to `'petrucci`.
- Set grob property `style` in `Rest` (page 469), to `'mensural`.
- Set grob property `thickness` in `Stem` (page 481), to `1.7`.
- Set translator property `autoBeaming` to `#f`.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Arpeggio_engraver` (page 284)
: Generate an Arpeggio symbol.

Music types accepted: `arpeggio-event` (page 45),

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Auto_beam_engraver` (page 284)

Generate beams based on measure characteristics and observed Stems. Uses `baseMoment`, `beatStructure`, `beamExceptions`, `measureLength`, and `measurePosition` to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [`Stem_engraver`], page 324, properties `stemLeftBeamCount` and `stemRightBeamCount`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
: If set to true then beams are generated automatically.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)
Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)
Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
Generate beams for tremolo repeats.

Music types accepted: `tremolo-span-event` (page 54),

This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
Engrave a cluster using `Spanner` notation.

Music types accepted: `cluster-note-event` (page 46),

This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
Make double measure repeats.

Music types accepted: `double-percent-event` (page 47),

Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`measureLength` (moment)
Length of one measure in the current time signature.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
Align hairpins and dynamic texts on a horizontal line.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)

Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)

Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)

Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)

Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
A queue of (*end-moment . grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
: Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
: Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
: Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
: The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
: Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Mensural_ligature_engraver` (page 310)
: Handle `Mensural_ligature_events` by glueing special ligature heads together.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `MensuralLigature` (page 444).

`Multi_measure_rest_engraver` (page 311)
: Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)
: Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
: Generate note heads.

  Music types accepted: `note-event` (page 50),

  Properties (read)

  `middleCPosition` (number)
  : The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

  `staffLineLayoutFunction` (procedure)
  : Layout of staff lines, `traditional`, or `semitone`.

  This engraver creates the following layout object(s): `NoteHead` (page 455).

`Note_spacing_engraver` (page 313)
: Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

  This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings ‘a2’, ‘Solo’, ‘Solo II’, and ‘unisono’.

  Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

  Properties (read)

  `aDueText` (markup)
  : Text to print at a unisono passage.

  `partCombineTextsOnNote` (boolean)
  : Print part-combine texts only on the next note rather than immediately on rests or skips.

  `printPartCombineTexts` (boolean)
  : Set ‘Solo’ and ‘A due’ texts in the part combiner?

  `soloIIText` (markup)
  : The text for the start of a solo for voice ‘two’ when part-combining.

  `soloText` (markup)
  : The text for the start of a solo when part-combining.

  This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
: Make whole measure repeats.

  Music types accepted: `percent-event` (page 51),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`repeatCountVisibility` (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)
: Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)
: Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
: Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
: Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
: Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
: Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)
The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
Build slur grobs from slur events.

Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

Properties (read)

`doubleSlurs` (boolean)
If set, two slurs are created for every slurred note, one above and one below the chord.

`slurMelismaBusy` (boolean)
Signal if a slur is present.

This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
See `stemLeftBeamCount`.

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
: Create text scripts.

  Music types accepted: `text-script-event` (page 54),

  This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
: Create text spanner from an event.

  Music types accepted: `text-span-event` (page 54),

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)
: Generate ties between note heads of equal pitch.

  Music types accepted: `tie-event` (page 54),

  Properties (read)

  `skipTypesetting` (boolean)
  : If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

  `tieWaitForNote` (boolean)
  : If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

  Properties (write)

  `tieMelismaBusy` (boolean)
  : Signal whether a tie is present.

  This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)
: Create trill spanner from an event.

  Music types accepted: `trill-span-event` (page 54),

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)
: Catch tuplet events and generate appropriate bracket.

  Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.24 PianoStaff

Just like `GrandStaff`, but the staves are only removed together, never separately.

This context also accepts commands for the following context(s): `GrandStaff` (page 99).

This context creates the following layout object(s): `Arpeggio` (page 355), `InstrumentName` (page 420), `SpanBar` (page 477), `SpanBarStub` (page 478), `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), `SystemStartSquare` (page 494), and `VerticalAlignment` (page 514).

This context sets the following properties:

- Set grob property `extra-spacing-width` in `DynamicText` (page 402), to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `#f`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `systemStartDelimiter` to `'SystemStartBrace`.
- Set translator property `systemStartDelimiter` to `'SystemStartBracket`.
- Set translator property `topLevelAlignment` to `#f`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Staff` (page 220).

Context `PianoStaff` can contain `ChoirStaff` (page 61), `ChordNames` (page 63), `Devnull` (page 76), `DrumStaff` (page 76), `Dynamics` (page 92), `FiguredBass` (page 95), `FretBoards` (page 97), `GrandStaff` (page 99), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `OneStaff` (page 170), `PetrucciStaff` (page 171), `PianoStaff` (page 192), `RhythmicStaff` (page 194), `Staff` (page 220), `StaffGroup` (page 229), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

This context is built from the following engraver(s):

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Keep_alive_together_engraver` (page 305)
: This engraver collects all `Hara_kiri_group_spanner`s that are created in contexts at or below its own. These spanners are then tied together so that one will be removed only if all are removed. For example, if a `StaffGroup` uses this engraver, then the staves in the group will all be visible as long as there is a note in at least one of them.

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Span_arpeggio_engraver` (page 322)
: Make arpeggios that span multiple staves.

Properties (read)

`connectArpeggios` (boolean)
: If set, connect arpeggios across piano staff.

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Span_bar_engraver` (page 323)
: Make cross-staff bar lines: It catches all normal bar lines and draws a single span bar across them.

This engraver creates the following layout object(s): `SpanBar` (page 477).

`Span_bar_stub_engraver` (page 323)
: Make stubs for span bars in all contexts that the span bars cross.

This engraver creates the following layout object(s): `SpanBarStub` (page 478).

`System_start_delimiter_engraver` (page 325)
: Create a system start delimiter (i.e., a `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket` or `SystemStartSquare` spanner).

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`systemStartDelimiter` (symbol)
: Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

`systemStartDelimiterHierarchy` (pair)
A nested list, indicating the nesting of a start delimiters.

This engraver creates the following layout object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

`Vertical_align_engraver` (page 331)
Catch groups (staves, lyrics lines, etc.) and stack them vertically.

Properties (read)

`alignAboveContext` (string)
Where to insert newly created context in vertical alignment.

`alignBelowContext` (string)
Where to insert newly created context in vertical alignment.

`hasAxisGroup` (boolean)
True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAlignment` (page 514).

2.1.25 RhythmicStaff

A context like `Staff` but for printing rhythms. Pitches are ignored; the notes are printed on one line.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `BarLine` (page 358), `DotColumn` (page 394), `InstrumentName` (page 420), `LedgerLineSpanner` (page 432), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `TimeSignature` (page 502), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `line-count` in `StaffSymbol` (page 480), to `1`.
- Set grob property `neutral-direction` in `Beam` (page 366), to `1`.
- Set grob property `neutral-direction` in `Stem` (page 481), to `1`.
- Set grob property `staff-padding` in `VoltaBracket` (page 517), to `3`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `squashedPosition` to `0`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Voice` (page 271).

Context `RhythmicStaff` can contain `CueVoice` (page 65), `NullVoice` (page 168), and `Voice` (page 271).

This context is built from the following engraver(s):

`Axis_group_engraver` (page 285)
Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): `DotColumn` (page 394).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Pitch_squash_engraver` (page 317)
: Set the vertical position of note heads to `squashedPosition`, if that property is set. This can be used to make a single-line staff demonstrating the rhythm of a melody.

Properties (read)

`squashedPosition` (integer)
: Vertical position of squashing for Section "Pitch_squash_engraver" in *Internals Reference*.

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_symbol_engraver` (page 324)
: Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)

Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)
break visibility for the initial time signature.

`partialBusy` (boolean)
Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)
A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.26 `Score`

This is the top level notation context. No other context can contain a `Score` context. This context handles the administration of time signatures. It also makes sure that items such as clefs, time signatures, and key-signatures are aligned across staves.

You cannot explicitly instantiate a `Score` context (since it is not contained in any other context). It is instantiated automatically when an output definition (a `\score` or `\layout` block) is processed.

An alias for `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.

This context also accepts commands for the following context(s): `Timing` (page 197).

This context creates the following layout object(s): `BarNumber` (page 361), `BreakAlignGroup` (page 371), `BreakAlignment` (page 372), `CenteredBarNumber` (page 375), `CenteredBarNumberLineSpanner` (page 376), `ControlPoint` (page 385), `ControlPolygon` (page 386), `Footnote` (page 411), `GraceSpacing` (page 415), `JumpScript` (page 422), `LeftEdge` (page 432), `MetronomeMark` (page 444), `NonMusicalPaperColumn` (page 452), `PaperColumn` (page 458), `Parentheses` (page 459), `RehearsalMark` (page 465), `SpacingSpanner` (page 476), `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), `SystemStartSquare` (page 494), `VerticalAlignment` (page 514), `VoltaBracket` (page 517), and `VoltaBracketSpanner` (page 518).

This context sets the following properties:

- Set translator property `additionalPitchPrefix` to `""`.
- Set translator property `aDueText` to `"a2"`.
- Set translator property `alterationGlyphs` to `#f`.
- Set translator property `alternativeRestores` to:

  ```
  '(measurePosition measureLength lastChord)
  ```

- Set translator property `associatedVoiceType` to `'Voice`.
- Set translator property `autoAccidentals` to:

  ```
  '(Staff #<procedure #f (context pitch barnum)>)
  ```

- Set translator property `autoBeamCheck` to `default-auto-beam-check`.

- Set translator property `autoBeaming` to `#t`.
- Set translator property `autoCautionaries` to `'()`.
- Set translator property `automaticBars` to `#t`.
- Set translator property `barCheckSynchronize` to `#f`.
- Set translator property `barNumberFormatter` to `robust-bar-number-function`.
- Set translator property `barNumberVisibility` to `first-bar-number-invisible-and-no-parenthesized-bar-numbers`.
- Set translator property `beamHalfMeasure` to `#t`.
- Set translator property `centerBarNumbers` to `#f`.
- Set translator property `chordNameExceptions` to:

```
'(((#<Pitch e' > #<Pitch gis' >)
  #<procedure line-markup (layout props args)>
  ("+"))
 ((#<Pitch ees' > #<Pitch ges' >)
  #<procedure line-markup (layout props args)>
  ((#<procedure line-markup (layout props args)>
    ((#<procedure fontsize-markup (layout props increment arg)>
     2
     "˚")))))
 ((#<Pitch ees' > #<Pitch ges' > #<Pitch bes' >)
  #<procedure line-markup (layout props args)>
  ((#<procedure super-markup (layout props arg)>
    "ø")))
 ((#<Pitch ees' > #<Pitch ges' > #<Pitch beses' >)
  #<procedure concat-markup (layout props args)>
  ((#<procedure line-markup (layout props args)>
    ((#<procedure fontsize-markup (layout props increment arg)>
     2
     "˚")))
   (#<procedure super-markup (layout props arg)>
    "7")))
 ((#<Pitch e' >
   #<Pitch g' >
   #<Pitch b' >
   #<Pitch fis'' >)
  #<procedure line-markup (layout props args)>
  ((#<procedure super-markup (layout props arg)>
    "lyd")))
 ((#<Pitch e' >
   #<Pitch g' >
   #<Pitch bes' >
   #<Pitch des'' >
   #<Pitch ees'' >
   #<Pitch fis'' >
   #<Pitch aes'' >)
  #<procedure line-markup (layout props args)>
  ((#<procedure super-markup (layout props arg)>
    "alt")))
 ((#<Pitch g' >)
  #<procedure line-markup (layout props args)>
```

```
 ((#<procedure super-markup (layout props arg)>
  "5")))
((#<Pitch g' > #<Pitch c'' >)
 #<procedure line-markup (layout props args)>
 ((#<procedure super-markup (layout props arg)>
  "5"))))
```

- Set translator property `chordNameFunction` to `ignatzek-chord-names`.
- Set translator property `chordNameLowercaseMinor` to `#f`.
- Set translator property `chordNameSeparator` to:

  ```
  '(#<procedure hspace-markup (layout props amount)>
   0.5)
  ```
- Set translator property `chordNoteNamer` to `'()`.
- Set translator property `chordPrefixSpacer` to `0`.
- Set translator property `chordRootNamer` to `note-name->markup`.
- Set translator property `clefGlyph` to `"clefs.G"`.
- Set translator property `clefPosition` to `-2`.
- Set translator property `clefTranspositionFormatter` to `clef-transposition-markup`.
- Set translator property `completionFactor` to `unity-if-multimeasure`.
- Set translator property `crescendoSpanner` to `'hairpin`.
- Set translator property `cueClefTranspositionFormatter` to `clef-transposition-markup`.
- Set translator property `decrescendoSpanner` to `'hairpin`.
- Set translator property `defaultBarType` to `"|"`.
- Set translator property `doubleRepeatType` to `":..:"`.
- Set translator property `drumStyleTable` to `#<hash-table 29/61>`.
- Set translator property `endRepeatType` to `":|."`.
- Set translator property `explicitClefVisibility` to:

  ```
  #(#t #t #t)
  ```
- Set translator property `explicitCueClefVisibility` to:

  ```
  #(#f #t #t)
  ```
- Set translator property `explicitKeySignatureVisibility` to:

  ```
  #(#t #t #t)
  ```
- Set translator property `extendersOverRests` to `#t`.
- Set translator property `extraNatural` to `#t`.
- Set translator property `figuredBassFormatter` to `format-bass-figure`.
- Set translator property `fineBarType` to `"|."`.
- Set translator property `fineText` to `"Fine"`.
- Set translator property `fingeringOrientations` to:

  ```
  '(up down)
  ```
- Set translator property `firstClef` to `#t`.
- Set translator property `graceSettings` to:

  ```
  '((Voice Stem direction 1)
   (Voice Slur direction -1)
   (Voice Stem font-size -3)
  ```

```
(Voice Flag font-size -3)
(Voice NoteHead font-size -3)
(Voice TabNoteHead font-size -4)
(Voice Dots font-size -3)
(Voice Stem length-fraction 0.8)
(Voice Stem no-stem-extend #t)
(Voice Beam beam-thickness 0.384)
(Voice Beam length-fraction 0.8)
(Voice Accidental font-size -4)
(Voice AccidentalCautionary font-size -4)
(Voice Script font-size -3)
(Voice Fingering font-size -8)
(Voice StringNumber font-size -8))
```

- Set translator property `harmonicAccidentals` to `#t`.
- Set translator property `highStringOne` to `#t`.
- Set translator property `initialTimeSignatureVisibility` to:

```
#(#f #t #t)
```

- Set translator property `instrumentTransposition` to `#<Pitch c' >`.
- Set translator property `keepAliveInterfaces` to:

```
'(bass-figure-interface
  chord-name-interface
  cluster-beacon-interface
  dynamic-interface
  fret-diagram-interface
  lyric-syllable-interface
  note-head-interface
  tab-note-head-interface
  lyric-interface
  percent-repeat-item-interface
  percent-repeat-interface
  stanza-number-interface)
```

- Set translator property `keyAlterationOrder` to:

```
'((6 . -1/2)
  (2 . -1/2)
  (5 . -1/2)
  (1 . -1/2)
  (4 . -1/2)
  (0 . -1/2)
  (3 . -1/2)
  (3 . 1/2)
  (0 . 1/2)
  (4 . 1/2)
  (1 . 1/2)
  (5 . 1/2)
  (2 . 1/2)
  (6 . 1/2)
  (6 . -1)
  (2 . -1)
  (5 . -1)
  (1 . -1)
```

```
(4 . -1)
(0 . -1)
(3 . -1)
(3 . 1)
(0 . 1)
(4 . 1)
(1 . 1)
(5 . 1)
(2 . 1)
(6 . 1))
```

- Set translator property `lyricMelismaAlignment` to `-1`.
- Set translator property `majorSevenSymbol` to:

```
'(#<procedure line-markup (layout props args)>
 ((#<procedure fontsize-markup (layout props increment arg)>
  -3
  (#<procedure triangle-markup (layout props filled)>
   #f))))
```

- Set translator property `markFormatter` to `format-mark-letters`.
- Set translator property `melismaBusyProperties` to:

```
'(melismaBusy
  slurMelismaBusy
  tieMelismaBusy
  beamMelismaBusy
  completionBusy)
```

- Set translator property `metronomeMarkFormatter` to `format-metronome-markup`.
- Set translator property `middleCClefPosition` to `-6`.
- Set translator property `middleCPosition` to `-6`.
- Set translator property `minorChordModifier` to:

```
'(#<procedure simple-markup (layout props str)>
 "m")
```

- Set translator property `noChordSymbol` to:

```
'(#<procedure simple-markup (layout props str)>
 "N.C.")
```

- Set translator property `noteNameFunction` to `note-name-markup`.
- Set translator property `noteNameSeparator` to `"/"`.
- Set translator property `noteToFretFunction` to `determine-frets`.
- Set translator property `partCombineTextsOnNote` to `#t`.
- Set translator property `pedalSostenutoStrings` to:

```
'("Sost. Ped." "*Sost. Ped." "*")
```

- Set translator property `pedalSostenutoStyle` to `'mixed`.
- Set translator property `pedalSustainStrings` to:

```
'("Ped." "*Ped." "*")
```

- Set translator property `pedalSustainStyle` to `'text`.
- Set translator property `pedalUnaCordaStrings` to:

```
'("una corda" "" "tre corde")
```

- Set translator property `pedalUnaCordaStyle` to `'text`.

- Set translator property `predefinedDiagramTable` to `#f`.
- Set translator property `printAccidentalNames` to `#t`.
- Set translator property `printKeyCancellation` to `#t`.
- Set translator property `printOctaveNames` to `#f`.
- Set translator property `printPartCombineTexts` to `#t`.
- Set translator property `quotedCueEventTypes` to:

```
'(note-event
  rest-event
  tie-event
  beam-event
  tuplet-span-event
  tremolo-event)
```

- Set translator property `quotedEventTypes` to:

```
'(StreamEvent)
```

- Set translator property `rehearsalMark` to `1`.
- Set translator property `repeatCountVisibility` to `all-repeat-counts-visible`.
- Set translator property `restNumberThreshold` to `1`.
- Set translator property `scriptDefinitions` to:

```
'(("accent"
  (avoid-slur . around)
  (padding . 0.2)
  (script-stencil feta "sforzato" . "sforzato")
  (side-relative-direction . -1))
 ("accentus"
  (script-stencil feta "uaccentus" . "uaccentus")
  (side-relative-direction . -1)
  (avoid-slur . ignore)
  (padding . 0.2)
  (quantize-position . #t)
  (script-priority . -100)
  (direction . 1))
 ("circulus"
  (script-stencil feta "circulus" . "circulus")
  (side-relative-direction . -1)
  (avoid-slur . ignore)
  (padding . 0.2)
  (quantize-position . #t)
  (script-priority . -100)
  (direction . 1))
 ("coda"
  (script-stencil feta "coda" . "coda")
  (padding . 0.2)
  (avoid-slur . outside)
  (direction . 1))
 ("comma"
  (script-stencil feta "lcomma" . "rcomma")
  (quantize-position . #t)
  (padding . 0.2)
  (avoid-slur . ignore)
```

```
 (direction . 1))
("downbow"
 (script-stencil feta "downbow" . "downbow")
 (padding . 0.2)
 (skyline-horizontal-padding . 0.2)
 (avoid-slur . around)
 (direction . 1)
 (script-priority . 150))
("downmordent"
 (script-stencil
  feta
  "downmordent"
  .
  "downmordent")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("downprall"
 (script-stencil feta "downprall" . "downprall")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("espressivo"
 (avoid-slur . around)
 (padding . 0.2)
 (script-stencil feta "espr" . "espr")
 (side-relative-direction . -1))
("fermata"
 (script-stencil feta "dfermata" . "ufermata")
 (padding . 0.2)
 (avoid-slur . around)
 (script-priority . 4000)
 (direction . 1))
("flageolet"
 (script-stencil feta "flageolet" . "flageolet")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("halfopen"
 (avoid-slur . outside)
 (padding . 0.2)
 (script-stencil feta "halfopen" . "halfopen")
 (direction . 1))
("halfopenvertical"
 (avoid-slur . outside)
 (padding . 0.2)
 (script-stencil
  feta
  "halfopenvertical"
  .
  "halfopenvertical")
 (direction . 1))
```

```
("haydnturn"
 (script-stencil feta "haydnturn" . "haydnturn")
 (padding . 0.2)
 (avoid-slur . inside)
 (direction . 1))
("henzelongfermata"
 (script-stencil
  feta
  "dhenzelongfermata"
  .
  "uhenzelongfermata")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("henzeshortfermata"
 (script-stencil
  feta
  "dhenzeshortfermata"
  .
  "uhenzeshortfermata")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("ictus"
 (script-stencil feta "ictus" . "ictus")
 (side-relative-direction . -1)
 (quantize-position . #t)
 (avoid-slur . ignore)
 (padding . 0.2)
 (script-priority . -100)
 (direction . -1))
("lheel"
 (script-stencil feta "upedalheel" . "upedalheel")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . -1))
("lineprall"
 (script-stencil feta "lineprall" . "lineprall")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("longfermata"
 (script-stencil
  feta
  "dlongfermata"
  .
  "ulongfermata")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("ltoe"
 (script-stencil feta "upedaltoe" . "upedaltoe")
```

```
 (padding . 0.2)
 (avoid-slur . around)
 (direction . -1))
("marcato"
 (script-stencil feta "dmarcato" . "umarcato")
 (padding . 0.2)
 (avoid-slur . inside)
 (quantize-position . #t)
 (side-relative-direction . -1))
("mordent"
 (script-stencil feta "mordent" . "mordent")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("open"
 (avoid-slur . outside)
 (padding . 0.2)
 (script-stencil feta "open" . "open")
 (direction . 1))
("portato"
 (script-stencil feta "uportato" . "dportato")
 (avoid-slur . around)
 (padding . 0.45)
 (side-relative-direction . -1))
("prall"
 (script-stencil feta "prall" . "prall")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("pralldown"
 (script-stencil feta "pralldown" . "pralldown")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("prallmordent"
 (script-stencil
  feta
  "prallmordent"
  .
  "prallmordent")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("prallprall"
 (script-stencil feta "prallprall" . "prallprall")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("prallup"
 (script-stencil feta "prallup" . "prallup")
 (padding . 0.2)
 (avoid-slur . around)
```

```
 (direction . 1))
("reverseturn"
 (script-stencil
  feta
  "reverseturn"
  .
  "reverseturn")
 (padding . 0.2)
 (avoid-slur . inside)
 (direction . 1))
("rheel"
 (script-stencil feta "dpedalheel" . "dpedalheel")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("rtoe"
 (script-stencil feta "dpedaltoe" . "dpedaltoe")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("segno"
 (script-stencil feta "segno" . "segno")
 (padding . 0.2)
 (avoid-slur . outside)
 (direction . 1))
("semicirculus"
 (script-stencil
  feta
  "dsemicirculus"
  .
  "dsemicirculus")
 (side-relative-direction . -1)
 (quantize-position . #t)
 (avoid-slur . ignore)
 (padding . 0.2)
 (script-priority . -100)
 (direction . 1))
("shortfermata"
 (script-stencil
  feta
  "dshortfermata"
  .
  "ushortfermata")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("signumcongruentiae"
 (script-stencil
  feta
  "dsignumcongruentiae"
  .
  "usignumcongruentiae")
```

```
 (padding . 0.2)
 (avoid-slur . outside)
 (direction . 1))
("slashturn"
 (script-stencil feta "slashturn" . "slashturn")
 (padding . 0.2)
 (avoid-slur . inside)
 (direction . 1))
("snappizzicato"
 (script-stencil
  feta
  "snappizzicato"
  .
  "snappizzicato")
 (padding . 0.2)
 (avoid-slur . outside)
 (direction . 1))
("staccatissimo"
 (avoid-slur . inside)
 (quantize-position . #t)
 (script-stencil
  feta
  "dstaccatissimo"
  .
  "ustaccatissimo")
 (padding . 0.2)
 (skyline-horizontal-padding . 0.1)
 (side-relative-direction . -1)
 (toward-stem-shift . 1.0)
 (toward-stem-shift-in-column . 0.0))
("staccato"
 (script-stencil feta "staccato" . "staccato")
 (side-relative-direction . -1)
 (quantize-position . #t)
 (avoid-slur . inside)
 (toward-stem-shift . 1.0)
 (toward-stem-shift-in-column . 0.0)
 (padding . 0.2)
 (skyline-horizontal-padding . 0.1)
 (script-priority . -100))
("stopped"
 (script-stencil feta "stopped" . "stopped")
 (avoid-slur . inside)
 (padding . 0.2)
 (direction . 1))
("tenuto"
 (script-stencil feta "tenuto" . "tenuto")
 (quantize-position . #t)
 (avoid-slur . inside)
 (padding . 0.2)
 (side-relative-direction . -1))
("trill"
```

```
 (script-stencil feta "trill" . "trill")
 (direction . 1)
 (padding . 0.2)
 (avoid-slur . outside)
 (script-priority . 2000))
("turn"
 (script-stencil feta "turn" . "turn")
 (avoid-slur . inside)
 (padding . 0.2)
 (direction . 1))
("upbow"
 (script-stencil feta "upbow" . "upbow")
 (avoid-slur . around)
 (padding . 0.2)
 (direction . 1)
 (script-priority . 150))
("upmordent"
 (script-stencil feta "upmordent" . "upmordent")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("upprall"
 (script-stencil feta "upprall" . "upprall")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("varcoda"
 (script-stencil feta "varcoda" . "varcoda")
 (padding . 0.2)
 (avoid-slur . outside)
 (direction . 1))
("varcomma"
 (script-stencil feta "lvarcomma" . "rvarcomma")
 (quantize-position . #t)
 (padding . 0.2)
 (avoid-slur . ignore)
 (direction . 1))
("verylongfermata"
 (script-stencil
  feta
  "dverylongfermata"
  .
  "uverylongfermata")
 (padding . 0.2)
 (avoid-slur . around)
 (direction . 1))
("veryshortfermata"
 (script-stencil
  feta
  "dveryshortfermata"
  .
  "uveryshortfermata")
```

```
(padding . 0.2)
(avoid-slur . around)
(direction . 1)))
```

- Set translator property `sectionBarType` to `"||"`.
- Set translator property `slashChordSeparator` to:

```
'(#<procedure simple-markup (layout props str)>
 "/")
```

- Set translator property `soloIIText` to `"Solo II"`.
- Set translator property `soloText` to `"Solo"`.
- Set translator property `startRepeatType` to `".|:"`.
- Set translator property `stringNumberOrientations` to:

```
'(up down)
```

- Set translator property `stringOneTopmost` to `#t`.
- Set translator property `stringTunings` to:

```
'(#<Pitch e' >
 #<Pitch b >
 #<Pitch g >
 #<Pitch d >
 #<Pitch a, >
 #<Pitch e, >)
```

- Set translator property `strokeFingerOrientations` to:

```
'(right)
```

- Set translator property `subdivideBeams` to `#f`.
- Set translator property `suspendMelodyDecisions` to `#f`.
- Set translator property `systemStartDelimiter` to `'SystemStartBar`.
- Set translator property `tablatureFormat` to `fret-number-tablature-format`.
- Set translator property `tabStaffLineLayoutFunction` to `tablature-position-on-lines`.
- Set translator property `tieWaitForNote` to `#f`.
- Set translator property `timeSignatureFraction` to:

```
'(4 . 4)
```

- Set translator property `timeSignatureSettings` to:

```
'(((2 . 2) (beamExceptions (end (1/32 8 8 8 8))))
 ((3 . 2)
  (beamExceptions (end (1/32 8 8 8 8 8 8))))
 ((3 . 4)
  (beamExceptions (end (1/8 6) (1/12 3 3 3))))
 ((3 . 8) (beamExceptions (end (1/8 3))))
 ((4 . 2)
  (beamExceptions (end (1/16 4 4 4 4 4 4 4 4))))
 ((4 . 4)
  (beamExceptions (end (1/8 4 4) (1/12 3 3 3 3))))
 ((4 . 8) (beatStructure 2 2))
 ((6 . 4)
  (beamExceptions (end (1/16 4 4 4 4 4 4))))
 ((9 . 4)
  (beamExceptions (end (1/32 8 8 8 8 8 8 8 8))))
```

```
((12 . 4)
 (beamExceptions
  (end (1/32 8 8 8 8 8 8 8 8 8 8 8 8))))
((5 . 8) (beatStructure 3 2))
((8 . 8) (beatStructure 3 3 2)))
```

- Set translator property `timing` to `#t`.
- Set translator property `topLevelAlignment` to `#t`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Staff` (page 220).

Context `Score` can contain `ChoirStaff` (page 61), `ChordNames` (page 63), `Devnull` (page 76), `DrumStaff` (page 76), `Dynamics` (page 92), `FiguredBass` (page 95), `FretBoards` (page 97), `GrandStaff` (page 99), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `OneStaff` (page 170), `PetrucciStaff` (page 171), `PianoStaff` (page 192), `RhythmicStaff` (page 194), `Staff` (page 220), `StaffGroup` (page 229), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

This context is built from the following engraver(s):

`Bar_number_engraver` (page 286)

A bar number may be created at any bar line, subject to the `barNumberVisibility` callback. By default, it is put on top of all staves and appears only at the left side of the staff. The staves are taken from `stavesFound`, which is maintained by Section 2.2.126 [`Staff_collecting_engraver`], page 323. This engraver usually creates `BarNumber` grobs, but when `centerBarNumbers` is true, it makes `CenteredBarNumber` grobs instead.

Properties (read)

`alternativeNumber` (integer)

When set, the index of the current `\alternative` element, starting from one. Not set outside of alternatives. Note the distinction from volta number: an alternative may pertain to multiple volte.

`alternativeNumberingStyle` (symbol)

The scheme and style for numbering bars in repeat alternatives. If not set (the default), bar numbers continue through alternatives. Can be set to `numbers` to reset the bar number at each alternative, or set to `numbers-with-letters` to reset and also include letter suffixes.

`barNumberFormatter` (procedure)

A procedure that takes a bar number, measure position, and alternative number and returns a markup of the bar number to print.

`barNumberVisibility` (procedure)

A procedure that takes a bar number and a measure position and returns whether the corresponding bar number should be printed. Note that the actual print-out of bar numbers is controlled with the `break-visibility` property.

The following procedures are predefined:

`all-bar-numbers-visible`
: Enable bar numbers for all bars, including the first one and broken bars (which get bar numbers in parentheses).

`first-bar-number-invisible`
: Enable bar numbers for all bars (including broken bars) except the first one. If the first bar is broken, it doesn't get a bar number either.

`first-bar-number-invisible-save-broken-bars`
: Enable bar numbers for all bars (including broken bars) except the first one. A broken first bar gets a bar number.

`first-bar-number-invisible-and-no-parenthesized-bar-numbers`
: Enable bar numbers for all bars except the first bar and broken bars. This is the default.

`(every-nth-bar-number-visible` *n*`)`
: Assuming *n* is value 2, for example, this enables bar numbers for bars 2, 4, 6, etc.

`(modulo-bar-number-visible` *n m*`)`
: If bar numbers 1, 4, 7, etc., should be enabled, *n* (the modulo) must be set to 3 and *m* (the division remainder) to 1.

`centerBarNumbers` (boolean)
: Whether to center bar numbers in their measure instead of aligning them on the bar line.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

This engraver creates the following layout object(s): `BarNumber` (page 361), and `CenteredBarNumber` (page 375).

`Beam_collision_engraver` (page 287)
: Help beams avoid colliding with notes and clefs in other voices.

`Break_align_engraver` (page 290)
: Align grobs with corresponding `break-align-symbols` into groups, and order the groups according to `breakAlignOrder`. The left edge of the alignment gets a separate group, with a symbol `left-edge`.

This engraver creates the following layout object(s): `BreakAlignGroup` (page 371), `BreakAlignment` (page 372), and `LeftEdge` (page 432).

`Centered_bar_number_align_engraver` (page 290)
Group measure-centered bar numbers in a `CenteredBarNumberLineSpanner` so they end up on the same vertical position.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `CenteredBarNumberLineSpanner` (page 376).

`Concurrent_hairpin_engraver` (page 294)
Collect concurrent hairpins.

`Default_bar_line_engraver` (page 295)
This engraver determines what kind of automatic bar lines should be produced, and sets `whichBar` accordingly. It should be at the same level as Section 2.2.143 [`Timing_translator`], page 329.

Properties (read)

`automaticBars` (boolean)
If set to false then bar lines will not be printed automatically; they must be explicitly created with a `\bar` command. Unlike the `\cadenzaOn` keyword, measures are still counted. Bar line generation will resume according to that count if this property is unset.

`barAlways` (boolean)
If set to true a bar line is drawn after each note.

`defaultBarType` (string)
Set the default type of bar line. See `whichBar` for information on available bar types.

This variable is read by Section "Timing_translator" in *Internals Reference* at Section "Score" in *Internals Reference* level.

`measureStartNow` (boolean)
True at the beginning of a measure.

Properties (write)

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

`Footnote_engraver` (page 300)
Create footnote texts.

This engraver creates the following layout object(s): `Footnote` (page 411).

`Grace_spacing_engraver` (page 303)
: Bookkeeping of shortest starting and playing notes in grace note runs.

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `GraceSpacing` (page 415).

`Jump_engraver` (page 305)
: Create `JumpScript` objects. It puts them outside all staves (which is taken from the property `stavesFound`). If moving this engraver to a different context, Section 2.2.126 [`Staff_collecting_engraver`], page 323, must move along, otherwise all marks end up on the same Y location.

  Music types accepted: `fine-event` (page 47),

  Properties (read)

  `stavesFound` (list of grobs)
  : A list of all staff-symbols found.

  This engraver creates the following layout object(s): `JumpScript` (page 422).

`Mark_engraver` (page 308)
: This engraver creates rehearsal marks.

  `Mark_engraver` creates marks formatted according to the `markFormatter` context property and places them vertically outside the set of staves given in the `stavesFound` context property.

  If `Mark_engraver` is added or moved to another context, `Staff_collecting_engraver` (page 323), also needs to be there so that marks appear at the intended Y location.

  By default, `Mark_engravers` in multiple contexts create a common sequence of marks chosen by the `Score`-level `Mark_tracking_translator` (page 308). If independent sequences are desired, multiple `Mark_tracking_translators` must be used.

  Properties (read)

  `currentMarkEvent` (stream event)
  : The event selected by `Mark_tracking_translator` for engraving by `Mark_engraver`.

  `markFormatter` (procedure)
  : A procedure taking as arguments the context and the sequence number of the rehearsal mark. It should return the formatted mark as a markup object.

  `stavesFound` (list of grobs)
  : A list of all staff-symbols found.

  This engraver creates the following layout object(s): `RehearsalMark` (page 465).

`Mark_tracking_translator` (page 308)
: This translator chooses which mark `Mark_engraver` should engrave.

Music types accepted: `ad-hoc-mark-event` (page 45), and `rehearsal-mark-event` (page 51),

Properties (read)

`rehearsalMark` (integer)
The last rehearsal mark printed.

Properties (write)

`currentMarkEvent` (stream event)
The event selected by `Mark_tracking_translator` for engraving by `Mark_engraver`.

`rehearsalMark` (integer)
The last rehearsal mark printed.

`Metronome_mark_engraver` (page 310)
Engrave metronome marking. This delegates the formatting work to the function in the `metronomeMarkFormatter` property. The mark is put over all staves. The staves are taken from the `stavesFound` property, which is maintained by Section 2.2.126 [`Staff_collecting_engraver`], page 323.

Music types accepted: `tempo-change-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`metronomeMarkFormatter` (procedure)
How to produce a metronome markup. Called with two arguments: a `TempoChangeEvent` and context.

`stavesFound` (list of grobs)
A list of all staff-symbols found.

`tempoHideNote` (boolean)
Hide the note = count in tempo marks.

This engraver creates the following layout object(s): `MetronomeMark` (page 444).

`Output_property_engraver` (page 314)
Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Paper_column_engraver` (page 315)
Take care of generating columns.

This engraver decides whether a column is breakable. The default is that a column is always breakable. However, every `Bar_engraver` that does not have a barline at a certain point will set `forbidBreaks` in the score context to stop line breaks. In practice, this means that you can make a break point by creating a bar line (assuming that there are no beams or notes that prevent a break point).

Music types accepted: `break-event` (page 46), and `label-event` (page 48),

Properties (read)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

Properties (write)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `NonMusicalPaperColumn` (page 452), and `PaperColumn` (page 458).

`Parenthesis_engraver` (page 315)
Parenthesize objects whose `parenthesize` property is `#t`.

This engraver creates the following layout object(s): `Parentheses` (page 459).

`Repeat_acknowledge_engraver` (page 318)
Acknowledge repeated music, and convert the contents of `repeatCommands` into an appropriate setting for `whichBar`.

Music types accepted: `fine-event` (page 47), `section-event` (page 52), `segno-event` (page 52), and `volta-span-event` (page 55),

Properties (read)

`defaultBarType` (string)
Set the default type of bar line. See `whichBar` for information on available bar types.

This variable is read by Section "Timing_translator" in *Internals Reference* at Section "Score" in *Internals Reference* level.

`doubleRepeatSegnoType` (string)
Set the default bar line for the combinations double repeat with segno. Default is '`:|.S.|:`'.

`doubleRepeatType` (string)
Set the default bar line for double repeats.

`endRepeatSegnoType` (string)
Set the default bar line for the combinations ending of repeat with segno. Default is '`:|.S`'.

`endRepeatType` (string)
Set the default bar line for the ending of repeats.

`fineBarType` (string)
The bar line for `\fine`. See `whichBar` for information on available bar types.

`fineSegnoType` (string)
Set the default bar line for a requested segno with fine. Default is '`|.S`'.

`fineStartRepeatSegnoType` (string)
: Set the default bar line for the combinations beginning of repeat with segno and fine. Default is '`|.S.|:`'.

`repeatCommands` (list)
: This property is a list of commands of the form `(list 'volta x)`, where *x* is a string or `#f`. `'end-repeat` is also accepted as a command.

`sectionBarType` (string)
: The bar line for `\section`. See `whichBar` for information on available bar types.

`segnoType` (string)
: Set the default bar line for a requested segno. Default is '`S`'.

`startRepeatSegnoType` (string)
: Set the default bar line for the combinations beginning of repeat with segno. Default is '`S.|:`'.

`startRepeatType` (string)
: Set the default bar line for the beginning of repeats.

`underlyingRepeatType` (string)
: Set the bar line to use at points of repetition or departure where no bar line would normally appear, for example at the end of a system broken in mid measure where the next system begins with a segno.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

  ```
  \set Staff.whichBar = ".|:"
  ```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

  ```
  \set Staff.whichBar = ".|:"
  ```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

`Show_control_points_engraver` (page 321)
: Create grobs to visualize control points of Bézier curves (ties and slurs) for ease of tweaking.

  This engraver creates the following layout object(s): `ControlPoint` (page 385), and `ControlPolygon` (page 386).

`Spacing_engraver` (page 322)
: Make a `SpacingSpanner` and do bookkeeping of shortest starting and playing notes.

Music types accepted: `spacing-section-event` (page 52),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`proportionalNotationDuration` (moment)
: Global override for shortest-playing duration. This is used for switching on proportional notation.

This engraver creates the following layout object(s): `SpacingSpanner` (page 476).

`Spanner_tracking_engraver` (page 323)
: Helper for creating spanners attached to other spanners. If a spanner has the `sticky-grob-interface`, the engraver tracks the spanner contained in its `sticky-host` object. When the host ends, the sticky spanner attached to it has its end announced too.

`Staff_collecting_engraver` (page 323)
: Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`Stanza_number_align_engraver` (page 324)
: This engraver ensures that stanza numbers are neatly aligned.

`System_start_delimiter_engraver` (page 325)
: Create a system start delimiter (i.e., a `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket` or `SystemStartSquare` spanner).

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`systemStartDelimiter` (symbol)
: Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

`systemStartDelimiterHierarchy` (pair)
: A nested list, indicating the nesting of a start delimiters.

This engraver creates the following layout object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

`Timing_translator` (page 329)

This engraver adds the alias `Timing` to its containing context. Responsible for synchronizing timing information from staves. Normally in `Score`. In order to create polyrhythmic music, this engraver should be removed from `Score` and placed in `Staff`.

Music types accepted: `alternative-event` (page 45),

Properties (read)

`alternativeNumberingStyle` (symbol)
: The scheme and style for numbering bars in repeat alternatives. If not set (the default), bar numbers continue through alternatives. Can be set to `numbers` to reset the bar number at each alternative, or set to `numbers-with-letters` to reset and also include letter suffixes.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

Properties (write)

`alternativeNumber` (integer)
: When set, the index of the current `\alternative` element, starting from one. Not set outside of alternatives. Note the distinction from volta number: an alternative may pertain to multiple volte.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

`Tweak_engraver` (page 330)
: Read the `tweaks` property from the originating event, and set properties.

`Vertical_align_engraver` (page 331)
: Catch groups (staves, lyrics lines, etc.) and stack them vertically.

  Properties (read)

  `alignAboveContext` (string)
  : Where to insert newly created context in vertical alignment.

  `alignBelowContext` (string)
  : Where to insert newly created context in vertical alignment.

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  This engraver creates the following layout object(s): `VerticalAlignment` (page 514).

`Volta_engraver` (page 331)
: Make volta brackets.

  Music types accepted: `volta-span-event` (page 55),

  Properties (read)

  `repeatCommands` (list)
  : This property is a list of commands of the form `(list 'volta` *x*`)`, where *x* is a string or `#f`. `'end-repeat` is also accepted as a command.

  `stavesFound` (list of grobs)
  : A list of all staff-symbols found.

  `voltaSpannerDuration` (moment)
  : This specifies the maximum duration to use for the brackets printed for `\alternative`. This can be used to shrink the length of brackets in the situation where one alternative is very large.

  This engraver creates the following layout object(s): `VoltaBracket` (page 517), and `VoltaBracketSpanner` (page 518).

2.1.27 Staff

Handles clefs, bar lines, keys, accidentals. It can contain Voice contexts.

This context creates the following layout object(s): Accidental (page 348), AccidentalCautionary (page 349), AccidentalPlacement (page 350), AccidentalSuggestion (page 350), BarLine (page 358), BassFigure (page 363), BassFigureAlignment (page 363), BassFigureAlignmentPositioning (page 364), BassFigureBracket (page 365), BassFigureContinuation (page 365), BassFigureLine (page 366), Clef (page 378), ClefModifier (page 381), CueClef (page 387), CueEndClef (page 389), DotColumn (page 394), FingeringColumn (page 410), InstrumentName (page 420), KeyCancellation (page 424), KeySignature (page 427), LedgerLineSpanner (page 432), NoteCollision (page 453), OttavaBracket (page 457), PianoPedalBracket (page 464), RestCollision (page 470), ScriptRow (page 472), SostenutoPedal (page 474), SostenutoPedalLineSpanner (page 475), StaffSpacing (page 479), StaffSymbol (page 480), SustainPedal (page 488), SustainPedalLineSpanner (page 489), TimeSignature (page 502), UnaCordaPedal (page 511), UnaCordaPedalLineSpanner (page 512), and VerticalAxisGroup (page 514).

This context sets the following properties:

- Set translator property createSpacing to #t.
- Set translator property ignoreFiguredBassRest to #f.
- Set translator property instrumentName to '().
- Set translator property localAlterations to '().
- Set translator property ottavationMarkups to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property shortInstrumentName to '().

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type Voice (page 271).

Context Staff can contain CueVoice (page 65), NullVoice (page 168), and Voice (page 271).

This context is built from the following engraver(s):

Accidental_engraver (page 282)
: Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at Voice level, so you can \override them at Voice.

  Properties (read)

  accidentalGrouping (symbol)
  : If set to 'voice, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

  autoAccidentals (list)
  : List of different ways to typeset an accidental.

For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

Each entry in the list is either a symbol or a procedure.

symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

procedure The procedure represents an accidental rule to be applied to the previously specified context.

The procedure takes the following arguments:

`context` The current context to which the rule should be applied.

`pitch` The pitch of the note to be evaluated.

`barnum` The current bar number.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
The current key signature. This is an alist containing `(`*`step . alter`*`)` or `((`*`octave . step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Alteration_glyph_engraver` (page 283)
: Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
: Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
: Position figured bass alignments over notes.

  This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
: Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

  Properties (read)

  `fontSize` (number)
  : The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

  Properties (read)

  `busyGrobs` (list)
  : A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

  Properties (write)

  `busyGrobs` (list)
  : A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `instrumentName` (markup)
  : The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

  `shortInstrumentName` (markup)
  : See `instrumentName`.

  `shortVocalName` (markup)
  : Name of a vocal line, short version.

  `vocalName` (markup)
  : Name of a vocal line.

  This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Key_engraver` (page 305)

Engrave a key signature.

Music types accepted: `key-change-event` (page 48),

Properties (read)

`createKeyOnClefChange` (boolean)
: Print a key signature whenever the clef is changed.

`explicitKeySignatureVisibility` (vector)
: '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(`*`step`*` . `*`alter`*`)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Ledger_line_engraver` (page 307)
Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
Engraver to merge multi-measure rest numbers in multiple voices.

This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Ottava_spanner_engraver` (page 314)
Create a text spanner when the ottavation property changes.

Music types accepted: `ottava-event` (page 50),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`middleCOffset` (number)
The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`ottavation` (markup)
If set, the text for an ottava spanner. Changing this creates a new text spanner.

This engraver creates the following layout object(s): `OttavaBracket` (page 457).

`Output_property_engraver` (page 314)
Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
: Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
: Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
: A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
: Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)
Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)
break visibility for the initial time signature.

`partialBusy` (boolean)
Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)
A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.28 StaffGroup

Groups staves while adding a bracket on the left side, grouping the staves together. The bar lines of the contained staves are connected vertically. `StaffGroup` only consists of a collection of staves, with a bracket in front and spanning bar lines.

This context creates the following layout object(s): `Arpeggio` (page 355), `InstrumentName` (page 420), `SpanBar` (page 477), `SpanBarStub` (page 478), `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), `SystemStartSquare` (page 494), and `VerticalAlignment` (page 514).

This context sets the following properties:

- Set grob property `extra-spacing-width` in `DynamicText` (page 402), to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `#f`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `shortInstrumentName` to `'()`.
- Set translator property `systemStartDelimiter` to `'SystemStartBracket`.

- Set translator property `topLevelAlignment` to `#f`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `Staff` (page 220).

Context `StaffGroup` can contain `ChoirStaff` (page 61), `ChordNames` (page 63), `Devnull` (page 76), `DrumStaff` (page 76), `Dynamics` (page 92), `FiguredBass` (page 95), `FretBoards` (page 97), `GrandStaff` (page 99), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `OneStaff` (page 170), `PetrucciStaff` (page 171), `PianoStaff` (page 192), `RhythmicStaff` (page 194), `Staff` (page 220), `StaffGroup` (page 229), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

This context is built from the following engraver(s):

`Instrument_name_engraver` (page 304)

Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
See `instrumentName`.

`shortVocalName` (markup)
Name of a vocal line, short version.

`vocalName` (markup)
Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Output_property_engraver` (page 314)

Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Span_arpeggio_engraver` (page 322)

Make arpeggios that span multiple staves.

Properties (read)

`connectArpeggios` (boolean)
If set, connect arpeggios across piano staff.

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Span_bar_engraver` (page 323)

Make cross-staff bar lines: It catches all normal bar lines and draws a single span bar across them.

This engraver creates the following layout object(s): `SpanBar` (page 477).

`Span_bar_stub_engraver` (page 323)

Make stubs for span bars in all contexts that the span bars cross.

This engraver creates the following layout object(s): `SpanBarStub` (page 478).

`System_start_delimiter_engraver` (page 325)
: Create a system start delimiter (i.e., a `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket` or `SystemStartSquare` spanner).

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `systemStartDelimiter` (symbol)
  : Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

  `systemStartDelimiterHierarchy` (pair)
  : A nested list, indicating the nesting of a start delimiters.

  This engraver creates the following layout object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

`Vertical_align_engraver` (page 331)
: Catch groups (staves, lyrics lines, etc.) and stack them vertically.

  Properties (read)

  `alignAboveContext` (string)
  : Where to insert newly created context in vertical alignment.

  `alignBelowContext` (string)
  : Where to insert newly created context in vertical alignment.

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  This engraver creates the following layout object(s): `VerticalAlignment` (page 514).

2.1.29 TabStaff

Context for generating tablature. It accepts only `TabVoice` contexts and handles the line spacing, the tablature clef etc. properly.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `PianoPedalBracket` (page 464), `RestCollision` (page 470), `ScriptRow` (page 472), `SostenutoPedal` (page 474), `SostenutoPedalLineSpanner` (page 475), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `SustainPedal` (page 488), `SustainPedalLineSpanner` (page 489), `TimeSignature` (page 502), `UnaCordaPedal` (page 511), `UnaCordaPedalLineSpanner` (page 512), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `after-line-breaking` in `RepeatTie` (page 467), to `repeat-tie::handle-tab-note-head`.

- Set grob property `after-line-breaking` in `Tie` (page 500), to `tie::handle-tab-note-head`.
- Set grob property `avoid-note-head` in `Stem` (page 481), to `#t`.
- Set grob property `beam-thickness` in `Beam` (page 366), to `0.32`.
- Set grob property `beam-thickness` in `StemTremolo` (page 484), to `0.32`.
- Set grob property `beam-width` in `StemTremolo` (page 484), to `stem-tremolo::calc-tab-width`.
- Set grob property `bound-details.left` in `Glissando` (page 414), to:

  ```
  '((attach-dir . 1) (padding . 0.3))
  ```

- Set grob property `bound-details.right` in `Glissando` (page 414), to:

  ```
  '((attach-dir . -1) (padding . 0.3))
  ```

- Set grob property `details` in `Stem` (page 481), to:

  ```
  '((lengths 0 0 0 0 0 0)
    (beamed-lengths 0 0 0)
    (beamed-minimum-free-lengths 0 0 0)
    (beamed-extreme-minimum-free-lengths 0 0)
    (stem-shorten 0 0))
  ```

- Set grob property `extra-dy` in `Glissando` (page 414), to `glissando::calc-tab-extra-dy`.
- Set grob property `glyph-name` in `TabNoteHead` (page 494), to `tab-note-head::calc-glyph-name`.
- Set grob property `ignore-collision` in `NoteColumn` (page 454), to `#t`.
- Set grob property `length-fraction` in `Beam` (page 366), to `0.62`.
- Set grob property `length-fraction` in `StemTremolo` (page 484), to `#<procedure #f (grob)>`.
- Set grob property `no-stem-extend` in `Stem` (page 481), to `#t`.
- Set grob property `staff-space` in `StaffSymbol` (page 480), to `1.5`.
- Set grob property `stencil` in `Arpeggio` (page 355), to `#f`.
- Set grob property `stencil` in `Beam` (page 366), to `#f`.
- Set grob property `stencil` in `Clef` (page 378), to `clef::print-modern-tab-if-set`.
- Set grob property `stencil` in `Dots` (page 394), to `#f`.
- Set grob property `stencil` in `DynamicTextSpanner` (page 403), to `#f`.
- Set grob property `stencil` in `DynamicText` (page 402), to `#f`.
- Set grob property `stencil` in `Flag` (page 410), to `#f`.
- Set grob property `stencil` in `Glissando` (page 414), to `glissando::draw-tab-glissando`.
- Set grob property `stencil` in `Hairpin` (page 417), to `#f`.
- Set grob property `stencil` in `LaissezVibrerTie` (page 430), to `#f`.
- Set grob property `stencil` in `MultiMeasureRestNumber` (page 447), to `#f`.
- Set grob property `stencil` in `MultiMeasureRestScript` (page 449), to `#f`.
- Set grob property `stencil` in `MultiMeasureRestText` (page 450), to `#f`.
- Set grob property `stencil` in `MultiMeasureRest` (page 446), to `#f`.
- Set grob property `stencil` in `PhrasingSlur` (page 462), to `#f`.
- Set grob property `stencil` in `RepeatTie` (page 467), to `#f`.

- Set grob property `stencil` in `Rest` (page 469), to `#f`.
- Set grob property `stencil` in `Script` (page 470), to `#f`.
- Set grob property `stencil` in `Slur` (page 472), to `slur::draw-tab-slur`.
- Set grob property `stencil` in `StemTremolo` (page 484), to `#f`.
- Set grob property `stencil` in `Stem` (page 481), to `#f`.
- Set grob property `stencil` in `TabNoteHead` (page 494), to `tab-note-head::whiteout-if-style-set`.
- Set grob property `stencil` in `TextScript` (page 496), to `#f`.
- Set grob property `stencil` in `TextSpanner` (page 498), to `#f`.
- Set grob property `stencil` in `Tie` (page 500), to `#f`.
- Set grob property `stencil` in `TimeSignature` (page 502), to `#f`.
- Set grob property `stencil` in `TupletBracket` (page 509), to `#f`.
- Set grob property `stencil` in `TupletNumber` (page 510), to `#f`.
- Set grob property `style` in `Flag` (page 410), to `'no-flag`.
- Set translator property `autoBeaming` to `#f`.
- Set translator property `clefGlyph` to `"clefs.tab"`.
- Set translator property `clefPosition` to `0`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `handleNegativeFrets` to `'recalculate`.
- Set translator property `ignoreFiguredBassRest` to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `ottavationMarkups` to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property `restrainOpenStrings` to `#f`.
- Set translator property `shortInstrumentName` to `'()`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `TabVoice` (page 240).

Context `TabStaff` can contain `CueVoice` (page 65), `NullVoice` (page 168), and `TabVoice` (page 240).

This context is built from the following engraver(s):

`Alteration_glyph_engraver` (page 283)

Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)

Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
Name of the symbol within the music font.

`clefPosition` (number)
Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
Don't swallow rest events.

`implicitBassFigures` (list)
A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
Position figured bass alignments over notes.

This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
See `instrumentName`.

`shortVocalName` (markup)
Name of a vocal line, short version.

`vocalName` (markup)
Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Ledger_line_engraver` (page 307)
Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
Engraver to merge multi-measure rest numbers in multiple voices.

This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Output_property_engraver` (page 314)
Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
See `pedalSustainStyle`.

`pedalSustainStrings` (list)
A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
: Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
: Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Tab_staff_symbol_engraver` (page 326)
: Create a tablature staff symbol, but look at `stringTunings` for the number of lines.

Properties (read)

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Time_signature_engraver` (page 328)
: Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)
: break visibility for the initial time signature.

`partialBusy` (boolean)
: Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

2.1.30 TabVoice

Context for drawing notes in a Tab staff.

This context also accepts commands for the following context(s): Voice (page 271).

This context creates the following layout object(s): Arpeggio (page 355), Beam (page 366), BendAfter (page 368), BendSpanner (page 369), BreathingSign (page 373), ClusterSpanner (page 382), ClusterSpannerBeacon (page 383), CombineTextScript (page 383), Dots (page 394), DoublePercentRepeat (page 395), DoublePercentRepeatCounter (page 396), DoubleRepeatSlash (page 398), DynamicLineSpanner (page 401), DynamicText (page 402), DynamicTextSpanner (page 403), FingerGlideSpanner (page 406), Flag (page 410), Glissando (page 414), Hairpin (page 417), InstrumentSwitch (page 421), LaissezVibrerTie (page 430), LaissezVibrerTieColumn (page 431), LigatureBracket (page 434), MultiMeasureRest (page 446), MultiMeasureRestNumber (page 447), MultiMeasureRestScript (page 449), MultiMeasureRestText (page 450), NoteColumn (page 454), NoteSpacing (page 456), PercentRepeat (page 460), PercentRepeatCounter (page 461), PhrasingSlur (page 462), RepeatSlash (page 467), RepeatTie (page 467), RepeatTieColumn (page 469), Rest (page 469), Script (page 470), ScriptColumn (page 471), Slur (page 472), Stem (page 481), StemStub (page 483), StemTremolo (page 484), TabNoteHead (page 494), TextScript (page 496), TextSpanner (page 498), Tie (page 500), TieColumn (page 501), TrillSpanner (page 507), TupletBracket (page 509), TupletNumber (page 510), and VoiceFollower (page 516).

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

Arpeggio_engraver (page 284)
: Generate an Arpeggio symbol.

  Music types accepted: arpeggio-event (page 45),

  This engraver creates the following layout object(s): Arpeggio (page 355).

Auto_beam_engraver (page 284)
: Generate beams based on measure characteristics and observed Stems. Uses baseMoment, beatStructure, beamExceptions, measureLength, and measurePosition to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [Stem_engraver], page 324, properties stemLeftBeamCount and stemRightBeamCount.

  Music types accepted: beam-forbid-event (page 46),

  Properties (read)

  autoBeaming (boolean)
  : If set to true then beams are generated automatically.

  baseMoment (moment)
  : Smallest unit of time that will stand on its own as a subdivided section.

  beamExceptions (list)
  : An alist of exceptions to autobeam rules that normally end on beats.

  beamHalfMeasure (boolean)
  : Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)
Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Bend_spanner_engraver` (page 289)
Engraver to print a BendSpanner.

Music types accepted: `bend-span-event` (page 46), `note-event` (page 50), and `string-number-event` (page 53),

Properties (read)

`stringFretFingerList` (list)
A list containg three entries. In `TabVoice` and `FretBoards` they determine the string, fret and finger to use

`supportNonIntegerFret` (boolean)
If set in `Score` the `TabStaff` will print micro-tones as '$2\frac{1}{2}$'

Properties (write)

`stringFretFingerList` (list)
A list containg three entries. In `TabVoice` and `FretBoards` they determine the string, fret and finger to use

`supportNonIntegerFret` (boolean)
If set in `Score` the `TabStaff` will print micro-tones as '$2\frac{1}{2}$'

This engraver creates the following layout object(s): `BendSpanner` (page 369).

`Breathing_sign_engraver` (page 290)
Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
Generate beams for tremolo repeats.

Music types accepted: `tremolo-span-event` (page 54),

This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
Engrave a cluster using `Spanner` notation.

Music types accepted: `cluster-note-event` (page 46),

This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
Make double measure repeats.

Music types accepted: `double-percent-event` (page 47),

Properties (read)

`countPercentRepeats` (boolean)
If set, produce counters for percent repeats.

`measureLength` (moment)
Length of one measure in the current time signature.

`repeatCountVisibility` (procedure)
A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
Align hairpins and dynamic texts on a horizontal line.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)

Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)

Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Font_size_engraver` (page 300)

Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)

Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Ligature_bracket_engraver` (page 307)
Handle `Ligature_events` by engraving `Ligature` brackets.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `LigatureBracket` (page 434).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_spacing_engraver` (page 313)
: Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings ‘a2’, ‘Solo’, ‘Solo II’, and ‘unisono’.

Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

Properties (read)

`aDueText` (markup)
: Text to print at a unisono passage.

`partCombineTextsOnNote` (boolean)
: Print part-combine texts only on the next note rather than immediately on rests or skips.

`printPartCombineTexts` (boolean)
: Set ‘Solo’ and ‘A due’ texts in the part combiner?

`soloIIText` (markup)
: The text for the start of a solo for voice ‘two’ when part-combining.

`soloText` (markup)
: The text for the start of a solo when part-combining.

This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
: Make whole measure repeats.

Music types accepted: `percent-event` (page 51),

Properties (read)

`countPercentRepeats` (boolean)
: If set, produce counters for percent repeats.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`repeatCountVisibility` (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Repeat_tie_engraver` (page 319)
: Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
: Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
: Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
: Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
: Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)
The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
Build slur grobs from slur events.

Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

Properties (read)

`doubleSlurs` (boolean)
If set, two slurs are created for every slurred note, one above and one below the chord.

`slurMelismaBusy` (boolean)
Signal if a slur is present.

This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
See `stemLeftBeamCount`.

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Tab_note_heads_engraver` (page 325)

Generate one or more tablature note heads from event of type `NoteEvent`.

Music types accepted: `fingering-event` (page 47), `note-event` (page 50), and `string-number-event` (page 53),

Properties (read)

`defaultStrings` (list)
: A list of strings to use in calculating frets for tablatures and fretboards if no strings are provided in the notes for the current moment.

`fretLabels` (list)
: A list of strings or Scheme-formatted markups containing, in the correct order, the labels to be used for lettered frets in tablature.

`highStringOne` (boolean)
: Whether the first string is the string with highest pitch on the instrument. This used by the automatic string selector for tablature notation.

`maximumFretStretch` (number)
: Don't allocate frets further than this from specified frets.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`minimumFret` (number)
: The tablature auto string-selecting mechanism selects the highest string with a fret at least `minimumFret`.

`noteToFretFunction` (procedure)
: Convert list of notes and list of defined strings to full list of strings and fret numbers. Parameters: The context, a list of note events, a list of tabstring events, and the fretboard grob if a fretboard is desired.

`stringOneTopmost` (boolean)
: Whether the first string is printed on the top line of the tablature.

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

`tablatureFormat` (procedure)
: A function formatting a tablature note head. Called with three arguments: context, string number and, fret number. It returns the text as a markup.

`tabStaffLineLayoutFunction` (procedure)
: A function determining the staff position of a tablature note head. Called with two arguments: the context and the string.

This engraver creates the following layout object(s): `TabNoteHead` (page 494).

`Tab_tie_follow_engraver` (page 326)
Adjust TabNoteHead properties when a tie is followed by a slur or glissando.

`Text_engraver` (page 327)
Create text scripts.

Music types accepted: `text-script-event` (page 54),

This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
Create text spanner from an event.

Music types accepted: `text-span-event` (page 54),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)
Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)
Create trill spanner from an event.

Music types accepted: `trill-span-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)

Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)

If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)

If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.1.31 VaticanaStaff

Same as `Staff` context, except that it is accommodated for typesetting Gregorian Chant in the notational style of Editio Vaticana.

This context also accepts commands for the following context(s): `Staff` (page 220).

This context creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), `AccidentalSuggestion` (page 350), `BarLine` (page 358), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), `BassFigureLine` (page 366), `Clef` (page 378), `ClefModifier` (page 381), `CueClef` (page 387), `CueEndClef` (page 389), `Custos` (page 392), `DotColumn` (page 394), `FingeringColumn` (page 410), `InstrumentName` (page 420), `KeyCancellation` (page 424), `KeySignature` (page 427), `LedgerLineSpanner` (page 432), `NoteCollision` (page 453), `OttavaBracket` (page 457), `PianoPedalBracket` (page 464), `RestCollision` (page 470), `ScriptRow` (page 472), `SostenutoPedal` (page 474), `SostenutoPedalLineSpanner` (page 475), `StaffSpacing` (page 479), `StaffSymbol` (page 480), `SustainPedal` (page 488), `SustainPedalLineSpanner` (page 489), `UnaCordaPedal` (page 511), `UnaCordaPedalLineSpanner` (page 512), and `VerticalAxisGroup` (page 514).

This context sets the following properties:

- Set grob property `hair-thickness` in `BarLine` (page 358), to `0.6`.
- Set grob property `line-count` in `StaffSymbol` (page 480), to `4`.
- Set grob property `neutral-direction` in `Custos` (page 392), to `-1`.
- Set grob property `neutral-position` in `Custos` (page 392), to `3`.
- Set grob property `style` in `Custos` (page 392), to `'vaticana`.
- Set grob property `style` in `Dots` (page 394), to `'vaticana`.
- Set grob property `thick-thickness` in `BarLine` (page 358), to `0.6`.
- Set grob property `thickness` in `StaffSymbol` (page 480), to `0.6`.
- Set translator property `alterationGlyphs` to:

```
'((-1/2 . "accidentals.vaticanaM1")
 (0 . "accidentals.vaticana0")
 (1/2 . "accidentals.mensural1"))
```

- Set translator property `clefGlyph` to `"clefs.vaticana.do"`.
- Set translator property `clefPosition` to `1`.

- Set translator property `clefTransposition` to `0`.
- Set translator property `createSpacing` to `#t`.
- Set translator property `defaultBarType` to `""`.
- Set translator property `ignoreFiguredBassRest` to `#f`.
- Set translator property `instrumentName` to `'()`.
- Set translator property `localAlterations` to `'()`.
- Set translator property `middleCClefPosition` to `1`.
- Set translator property `middleCPosition` to `1`.
- Set translator property `ottavationMarkups` to:

```
'((4 . "29")
  (3 . "22")
  (2 . "15")
  (1 . "8")
  (-1 . "8")
  (-2 . "15")
  (-3 . "22")
  (-4 . "29"))
```

- Set translator property `shortInstrumentName` to `'()`.

This is not a 'Bottom' context; search for such a one will commence after creating an implicit context of type `VaticanaVoice` (page 261).

Context `VaticanaStaff` can contain `CueVoice` (page 65), `NullVoice` (page 168), and `VaticanaVoice` (page 261).

This context is built from the following engraver(s):

`Accidental_engraver` (page 282)

Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at `Voice` level, so you can `\override` them at `Voice`.

Properties (read)

`accidentalGrouping` (symbol)

If set to `'voice`, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

`autoAccidentals` (list)

List of different ways to typeset an accidental.

For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

Each entry in the list is either a symbol or a procedure.

symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

procedure The procedure represents an accidental rule to be applied to the previously specified context.

The procedure takes the following arguments:

`context` The current context to which the rule should be applied.

`pitch` The pitch of the note to be evaluated.

`barnum` The current bar number.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step`*` . `*`alter`*`)` or `((`*`octave`*` . `*`step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave`*` . `*`name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave`*` . `*`name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Alteration_glyph_engraver` (page 283)
: Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

  Properties (read)

  `alterationGlyphs` (list)
  : Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Axis_group_engraver` (page 285)
: Group all objects created in this context in a `VerticalAxisGroup` spanner.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  `keepAliveInterfaces` (list)
  : A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

  Properties (write)

  `hasAxisGroup` (boolean)
  : True if the current context is contained in an axis group.

  This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Bar_engraver` (page 285)
: Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

  Properties (read)

  `whichBar` (string)
  : This property is read to determine what type of bar line to create.

    Example:

    ```
    \set Staff.whichBar = ".|:"
    ```

    This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

  Properties (write)

  `forbidBreak` (boolean)
  : If set to `#t`, prevent a line break at this point.

  This engraver creates the following layout object(s): `BarLine` (page 358).

`Clef_engraver` (page 291)
: Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Collision_engraver` (page 292)
: Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Cue_clef_engraver` (page 294)
: Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Custos_engraver` (page 295)
: Engrave custodes.

This engraver creates the following layout object(s): `Custos` (page 392).

`Dot_column_engraver` (page 295)
: Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): `DotColumn` (page 394).

`Figured_bass_engraver` (page 299)
: Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_position_engraver` (page 299)
: Position figured bass alignments over notes.

This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Fingering_column_engraver` (page 300)
: Find potentially colliding scripts and put them into a `FingeringColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `FingeringColumn` (page 410).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

  Properties (read)

  `fontSize` (number)
  : The relative size of all grobs in a context.

`Grob_pq_engraver` (page 303)
: Administrate when certain grobs (e.g., note heads) stop playing.

  Properties (read)

  `busyGrobs` (list)
  : A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

  Properties (write)

  `busyGrobs` (list)
  : A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_name_engraver` (page 304)
: Create a system start text for instrument or vocal names.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `instrumentName` (markup)
  : The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

  `shortInstrumentName` (markup)
  : See `instrumentName`.

  `shortVocalName` (markup)
  : Name of a vocal line, short version.

  `vocalName` (markup)
  : Name of a vocal line.

  This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Key_engraver` (page 305)
: Engrave a key signature.

  Music types accepted: `key-change-event` (page 48),

  Properties (read)

  `createKeyOnClefChange` (boolean)
  : Print a key signature whenever the clef is changed.

  `explicitKeySignatureVisibility` (vector)
  : '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(step . alter)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Ledger_line_engraver` (page 307)
: Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Merge_mmrest_numbers_engraver` (page 310)
: Engraver to merge multi-measure rest numbers in multiple voices.

This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Ottava_spanner_engraver` (page 314)
: Create a text spanner when the ottavation property changes.

  Music types accepted: `ottava-event` (page 50),

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  `middleCOffset` (number)
  : The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

  `ottavation` (markup)
  : If set, the text for an ottava spanner. Changing this creates a new text spanner.

  This engraver creates the following layout object(s): `OttavaBracket` (page 457).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Piano_pedal_align_engraver` (page 316)
: Align piano pedal symbols and brackets.

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_engraver` (page 317)
: Engrave piano pedal symbols and brackets.

  Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `pedalSostenutoStrings` (list)
  : See `pedalSustainStrings`.

  `pedalSostenutoStyle` (symbol)
  : See `pedalSustainStyle`.

  `pedalSustainStrings` (list)
  : A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

  `pedalSustainStyle` (symbol)
  : A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Pure_from_neighbor_engraver` (page 318)
: Coordinates items that get their pure heights from their neighbors.

`Rest_collision_engraver` (page 320)
: Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Script_row_engraver` (page 321)
: Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Separating_line_group_engraver` (page 321)
: Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Staff_collecting_engraver` (page 323)
: Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`Staff_symbol_engraver` (page 324)
: Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

2.1.32 VaticanaVoice

Same as Voice context, except that it is accommodated for typesetting Gregorian Chant in the notational style of Editio Vaticana.

This context also accepts commands for the following context(s): Voice (page 271).

This context creates the following layout object(s): Arpeggio (page 355), Beam (page 366), BendAfter (page 368), BreathingSign (page 373), ClusterSpanner (page 382), ClusterSpannerBeacon (page 383), CombineTextScript (page 383), DotColumn (page 394), Dots (page 394), DoublePercentRepeat (page 395), DoublePercentRepeatCounter (page 396), DoubleRepeatSlash (page 398), DynamicLineSpanner (page 401), DynamicText (page 402), DynamicTextSpanner (page 403), Episema (page 405), FingerGlideSpanner (page 406), Fingering (page 408), Glissando (page 414), Hairpin (page 417), InstrumentSwitch (page 421), LaissezVibrerTie (page 430), LaissezVibrerTieColumn (page 431), MultiMeasureRest (page 446), MultiMeasureRestNumber (page 447), MultiMeasureRestScript (page 449), MultiMeasureRestText (page 450), NoteColumn (page 454), NoteHead (page 455), NoteSpacing (page 456), PercentRepeat (page 460), PercentRepeatCounter (page 461), PhrasingSlur (page 462), RepeatSlash (page 467), RepeatTie (page 467), RepeatTieColumn (page 469), Rest (page 469), Script (page 470), ScriptColumn (page 471), StringNumber (page 485), StrokeFinger (page 487), TextScript (page 496), Tie (page 500), TieColumn (page 501), TrillPitchAccidental (page 504), TrillPitchGroup (page 505), TrillPitchHead (page 506), TrillSpanner (page 507), TupletBracket (page 509), TupletNumber (page 510), VaticanaLigature (page 513), and VoiceFollower (page 516).

This context sets the following properties:

- Set grob property padding in Script (page 470), to 0.5.
- Set grob property style in NoteHead (page 455), to 'vaticana.punctum.
- Set translator property autoBeaming to #f.

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

Arpeggio_engraver (page 284)

Generate an Arpeggio symbol.

Music types accepted: arpeggio-event (page 45),

This engraver creates the following layout object(s): Arpeggio (page 355).

Auto_beam_engraver (page 284)

Generate beams based on measure characteristics and observed Stems. Uses baseMoment, beatStructure, beamExceptions, measureLength, and measurePosition to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [Stem_engraver], page 324, properties stemLeftBeamCount and stemRightBeamCount.

Music types accepted: beam-forbid-event (page 46),

Properties (read)

autoBeaming (boolean)

If set to true then beams are generated automatically.

baseMoment (moment)

Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)
: Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)
: Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)
: Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
: Generate beams for tremolo repeats.

Music types accepted: `tremolo-span-event` (page 54),

This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
: Engrave a cluster using `Spanner` notation.

  Music types accepted: `cluster-note-event` (page 46),

  This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
: Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

  This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
: Make double measure repeats.

  Music types accepted: `double-percent-event` (page 47),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `measureLength` (moment)
  : Length of one measure in the current time signature.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  Properties (write)

  `forbidBreak` (boolean)
  : If set to `#t`, prevent a line break at this point.

  This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
: Align hairpins and dynamic texts on a horizontal line.

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
: Create hairpins, dynamic texts and dynamic text spanners.

  Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

  Properties (read)

  `crescendoSpanner` (symbol)
  : The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Episema_engraver` (page 298)
Create an *Editio Vaticana*-style episema line.

Music types accepted: `episema-event` (page 47),

This engraver creates the following layout object(s): `Episema` (page 405).

`Finger_glide_engraver` (page 300)
Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)
Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)
Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
True at the beginning of a measure.

`restNumberThreshold` (number)
If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)
Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
See `fingeringOrientations`.

`strokeFingerOrientations` (list)
See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
Generate note heads.

Music types accepted: `note-event` (page 50),

Properties (read)

`middleCPosition` (number)
The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`staffLineLayoutFunction` (procedure)
Layout of staff lines, `traditional`, or `semitone`.

This engraver creates the following layout object(s): `NoteHead` (page 455).

`Note_spacing_engraver` (page 313)
Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings 'a2', 'Solo', 'Solo II', and 'unisono'.

  Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

  Properties (read)

  `aDueText` (markup)
  : Text to print at a unisono passage.

  `partCombineTextsOnNote` (boolean)
  : Print part-combine texts only on the next note rather than immediately on rests or skips.

  `printPartCombineTexts` (boolean)
  : Set 'Solo' and 'A due' texts in the part combiner?

  `soloIIText` (markup)
  : The text for the start of a solo for voice 'two' when part-combining.

  `soloText` (markup)
  : The text for the start of a solo when part-combining.

  This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
: Make whole measure repeats.

  Music types accepted: `percent-event` (page 51),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

  Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

  This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)
: Print the bracketed note head after a note head with trill.

  This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)
: Create repeat ties.

  Music types accepted: `repeat-tie-event` (page 51),

  This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)
: Engrave rests.

  Music types accepted: `rest-event` (page 51),

  Properties (read)

  `middleCPosition` (number)
  : The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

  This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)
: Generate `NoteColumn`, an object that groups stems, note heads, and rests.

  This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)
: Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

  This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)
: Handle note scripted articulations.

  Music types accepted: `articulation-event` (page 45),

  Properties (read)

  `scriptDefinitions` (list)
  : The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

  This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)
: Make beat repeats.

  Music types accepted: `repeat-slash-event` (page 51),

  This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Spanner_break_forbid_engraver` (page 323)
: Forbid breaks in certain spanners.

`Text_engraver` (page 327)
: Create text scripts.

  Music types accepted: `text-script-event` (page 54),

  This engraver creates the following layout object(s): `TextScript` (page 496).

`Tie_engraver` (page 327)
: Generate ties between note heads of equal pitch.

  Music types accepted: `tie-event` (page 54),

  Properties (read)

  `skipTypesetting` (boolean)
  : If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

  `tieWaitForNote` (boolean)
  : If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

  Properties (write)

  `tieMelismaBusy` (boolean)
  : Signal whether a tie is present.

  This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)
: Create trill spanner from an event.

  Music types accepted: `trill-span-event` (page 54),

  Properties (read)

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)
: Catch tuplet events and generate appropriate bracket.

  Music types accepted: `tuplet-span-event` (page 54),

  Properties (read)

  `tupletFullLength` (boolean)
  : If set, the tuplet is printed up to the start of the next note.

  `tupletFullLengthNote` (boolean)
  : If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

  This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

`Vaticana_ligature_engraver` (page 331)
: Handle ligatures by glueing special ligature heads together.

  Music types accepted: `ligature-event` (page 48), and `pes-or-flexa-event` (page 51),

  This engraver creates the following layout object(s): `DotColumn` (page 394), and `VaticanaLigature` (page 513).

2.1.33 `Voice`

Corresponds to a voice on a staff. This context handles the conversion of dynamic signs, stems, beams, super- and subscripts, slurs, ties, and rests.

You have to instantiate this explicitly if you want to have multiple voices on the same staff.

This context creates the following layout object(s): `Arpeggio` (page 355), `Beam` (page 366), `BendAfter` (page 368), `BreathingSign` (page 373), `ClusterSpanner` (page 382), `ClusterSpannerBeacon` (page 383), `CombineTextScript` (page 383), `Dots` (page 394), `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `FingerGlideSpanner` (page 406), `Fingering` (page 408), `Flag` (page 410), `Glissando` (page 414), `Hairpin` (page 417), `InstrumentSwitch` (page 421), `LaissezVibrerTie` (page 430), `LaissezVibrerTieColumn` (page 431), `LigatureBracket` (page 434), `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `NoteColumn` (page 454), `NoteHead` (page 455), `NoteSpacing` (page 456), `PercentRepeat` (page 460), `PercentRepeatCounter` (page 461), `PhrasingSlur` (page 462), `RepeatSlash` (page 467), `RepeatTie` (page 467), `RepeatTieColumn` (page 469), `Rest` (page 469), `Script` (page 470), `ScriptColumn` (page 471), `Slur` (page 472), `Stem` (page 481), `StemStub` (page 483), `StemTremolo` (page 484), `StringNumber` (page 485), `StrokeFinger` (page 487), `TextScript` (page 496), `TextSpanner` (page 498), `Tie` (page 500), `TieColumn` (page 501), `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), `TrillPitchHead` (page 506), `TrillSpanner` (page 507), `TupletBracket` (page 509), `TupletNumber` (page 510), and `VoiceFollower` (page 516).

This is a 'Bottom' context; no contexts will be created implicitly from it.

This context cannot contain other contexts.

This context is built from the following engraver(s):

`Arpeggio_engraver` (page 284)
: Generate an Arpeggio symbol.

  Music types accepted: `arpeggio-event` (page 45),

  This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Auto_beam_engraver` (page 284)
: Generate beams based on measure characteristics and observed Stems. Uses `baseMoment`, `beatStructure`, `beamExceptions`, `measureLength`, and `measurePosition` to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [`Stem_engraver`], page 324, properties `stemLeftBeamCount` and `stemRightBeamCount`.

  Music types accepted: `beam-forbid-event` (page 46),

  Properties (read)

  `autoBeaming` (boolean)
  : If set to true then beams are generated automatically.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` (page 287)

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Bend_engraver` (page 289)

Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Breathing_sign_engraver` (page 290)

Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Chord_tremolo_engraver` (page 291)
: Generate beams for tremolo repeats.

  Music types accepted: `tremolo-span-event` (page 54),

  This engraver creates the following layout object(s): `Beam` (page 366).

`Cluster_spanner_engraver` (page 292)
: Engrave a cluster using `Spanner` notation.

  Music types accepted: `cluster-note-event` (page 46),

  This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Dots_engraver` (page 296)
: Create Section 3.1.39 [`Dots`], page 394, objects for Section 3.2.111 [`rhythmic-head-interface`], page 577s.

  This engraver creates the following layout object(s): `Dots` (page 394).

`Double_percent_repeat_engraver` (page 296)
: Make double measure repeats.

  Music types accepted: `double-percent-event` (page 47),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `measureLength` (moment)
  : Length of one measure in the current time signature.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  Properties (write)

  `forbidBreak` (boolean)
  : If set to `#t`, prevent a line break at this point.

  This engraver creates the following layout object(s): `DoublePercentRepeat` (page 395), and `DoublePercentRepeatCounter` (page 396).

`Dynamic_align_engraver` (page 297)
: Align hairpins and dynamic texts on a horizontal line.

  Properties (read)

  `currentMusicalColumn` (graphical (layout) object)
  : Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

  This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_engraver` (page 297)
: Create hairpins, dynamic texts and dynamic text spanners.

  Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Finger_glide_engraver` (page 300)
: Engraver to print a line between two `Fingering` grobs.

Music types accepted: `note-event` (page 50),

This engraver creates the following layout object(s): `FingerGlideSpanner` (page 406).

`Fingering_engraver` (page 300)
: Create fingering scripts.

Music types accepted: `fingering-event` (page 47),

This engraver creates the following layout object(s): `Fingering` (page 408).

`Font_size_engraver` (page 300)
: Put `fontSize` into `font-size` grob property.

Properties (read)

`fontSize` (number)
: The relative size of all grobs in a context.

`Forbid_line_break_engraver` (page 301)
: Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
: A queue of (*`end-moment . grob`*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`Glissando_engraver` (page 302)
Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Grace_auto_beam_engraver` (page 302)
Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` (page 302)
Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
Signal if a beam is present.

`beatStructure` (list)
List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_engraver` (page 303)
Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grob_pq_engraver` (page 303)
Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Instrument_switch_engraver` (page 305)
Create a cue text for taking instrument.

Properties (read)

`instrumentCueName` (markup)
The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): `InstrumentSwitch` (page 421).

`Laissez_vibrer_engraver` (page 307)
Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Ligature_bracket_engraver` (page 307)
Handle `Ligature_events` by engraving `Ligature` brackets.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `LigatureBracket` (page 434).

`Multi_measure_rest_engraver` (page 311)
Engrave multi-measure rests that are produced with '`R`'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`New_fingering_engraver` (page 312)
: Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`Note_head_line_engraver` (page 312)
: Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_heads_engraver` (page 313)
: Generate note heads.

Music types accepted: `note-event` (page 50),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

This engraver creates the following layout object(s): `NoteHead` (page 455).

`Note_spacing_engraver` (page 313)
: Generate `NoteSpacing`, an object linking horizontal lines for use in spacing.

  This engraver creates the following layout object(s): `NoteSpacing` (page 456).

`Output_property_engraver` (page 314)
: Apply a procedure to any grob acknowledged.

  Music types accepted: `apply-output-event` (page 45),

`Part_combine_engraver` (page 315)
: Part combine engraver for orchestral scores: Print markings 'a2', 'Solo', 'Solo II', and 'unisono'.

  Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

  Properties (read)

  `aDueText` (markup)
  : Text to print at a unisono passage.

  `partCombineTextsOnNote` (boolean)
  : Print part-combine texts only on the next note rather than immediately on rests or skips.

  `printPartCombineTexts` (boolean)
  : Set 'Solo' and 'A due' texts in the part combiner?

  `soloIIText` (markup)
  : The text for the start of a solo for voice 'two' when part-combining.

  `soloText` (markup)
  : The text for the start of a solo when part-combining.

  This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Percent_repeat_engraver` (page 316)
: Make whole measure repeats.

  Music types accepted: `percent-event` (page 51),

  Properties (read)

  `countPercentRepeats` (boolean)
  : If set, produce counters for percent repeats.

  `currentCommandColumn` (graphical (layout) object)
  : Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

  `repeatCountVisibility` (procedure)
  : A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

  This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Phrasing_slur_engraver` (page 316)
: Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Pitched_trill_engraver` (page 318)

Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Repeat_tie_engraver` (page 319)

Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Rest_engraver` (page 320)

Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)

The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rhythmic_column_engraver` (page 320)

Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Script_column_engraver` (page 320)

Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_engraver` (page 320)

Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)

The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Slash_repeat_engraver` (page 321)

Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

`Slur_engraver` (page 322)
Build slur grobs from slur events.

Music types accepted: `note-event` (page 50), and `slur-event` (page 52),

Properties (read)

`doubleSlurs` (boolean)
If set, two slurs are created for every slurred note, one above and one below the chord.

`slurMelismaBusy` (boolean)
Signal if a slur is present.

This engraver creates the following layout object(s): `Slur` (page 472).

`Spanner_break_forbid_engraver` (page 323)
Forbid breaks in certain spanners.

`Stem_engraver` (page 324)
Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
See `stemLeftBeamCount`.

`whichBar` (string)
This property is read to determine what type of bar line to create.

Example:

```
\set Staff.whichBar = ".|:"
```

This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Text_engraver` (page 327)
Create text scripts.

Music types accepted: `text-script-event` (page 54),

This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_spanner_engraver` (page 327)
Create text spanner from an event.

Music types accepted: `text-span-event` (page 54),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Tie_engraver` (page 327)

Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Trill_spanner_engraver` (page 330)

Create trill spanner from an event.

Music types accepted: `trill-span-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Tuplet_engraver` (page 330)

Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

2.2 Engravers and Performers

See Section "Modifying context plug-ins" in *Notation Reference*.

2.2.1 `Accidental_engraver`

Make accidentals. Catch note heads, ties and notices key-change events. This engraver usually lives at Staff level, but reads the settings for Accidental at `Voice` level, so you can `\override` them at `Voice`.

Properties (read)

`accidentalGrouping` (symbol)
: If set to `'voice`, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

`autoAccidentals` (list)
: List of different ways to typeset an accidental.

  For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

  Each entry in the list is either a symbol or a procedure.

  symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

  procedure The procedure represents an accidental rule to be applied to the previously specified context.

  The procedure takes the following arguments:

  `context` The current context to which the rule should be applied.

  `pitch` The pitch of the note to be evaluated.

  `barnum` The current bar number.

  The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step . alter`*`)` or `((`*`octave . step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

Properties (write)

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

This engraver creates the following layout object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), and `AccidentalSuggestion` (page 350).

`Accidental_engraver` is part of the following context(s) in `\layout`: `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), and `VaticanaStaff` (page 251).

2.2.2 Alteration_glyph_engraver

Set the `glyph-name-alist` of all grobs having the `accidental-switch-interface` to the value of the context's `alterationGlyphs` property, when defined.

Properties (read)

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`Alteration_glyph_engraver` is part of the following context(s) in `\layout`: `ChordNames` (page 63), `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `NoteNames` (page 166), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.3 Ambitus_engraver

Create an ambitus.

Properties (read)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(`*`step . alter`*`)` or `((`*`octave . step`*`) . `*`alter`*`)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

`middleCOffset` (number)
: The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

This engraver creates the following layout object(s): `AccidentalPlacement` (page 350), `Ambitus` (page 352), `AmbitusAccidental` (page 354), `AmbitusLine` (page 354), and `AmbitusNoteHead` (page 355).

`Ambitus_engraver` is not part of any context

2.2.4 Arpeggio_engraver

Generate an Arpeggio symbol.

Music types accepted: `arpeggio-event` (page 45),

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Arpeggio_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.5 Auto_beam_engraver

Generate beams based on measure characteristics and observed Stems. Uses `baseMoment`, `beatStructure`, `beamExceptions`, `measureLength`, and `measurePosition` to decide when to start and stop a beam. Overriding beaming is done through Section 2.2.131 [`Stem_engraver`], page 324, properties `stemLeftBeamCount` and `stemRightBeamCount`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
: If set to true then beams are generated automatically.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatStructure` (list)
: List of `baseMoments` that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Auto_beam_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.6 Axis_group_engraver

Group all objects created in this context in a `VerticalAxisGroup` spanner.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

Properties (write)

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAxisGroup` (page 514).

`Axis_group_engraver` is part of the following context(s) in `\layout`: `ChordNames` (page 63), `DrumStaff` (page 76), `Dynamics` (page 92), `FiguredBass` (page 95), `FretBoards` (page 97), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `OneStaff` (page 170), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.7 Balloon_engraver

Create balloon texts.

Music types accepted: `annotate-output-event` (page 45),

This engraver creates the following layout object(s): `BalloonText` (page 357).

`Balloon_engraver` is not part of any context

2.2.8 Bar_engraver

Create barlines. This engraver is controlled through the `whichBar` property. If it has no bar line to create, it will forbid a linebreak at this point. This engraver is required to trigger the creation of clefs at the start of systems.

Properties (read)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

```
\set Staff.whichBar = ".|:"
```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `BarLine` (page 358).

`Bar_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `Dynamics` (page 92), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.9 Bar_number_engraver

A bar number may be created at any bar line, subject to the `barNumberVisibility` callback. By default, it is put on top of all staves and appears only at the left side of the staff. The staves are taken from `stavesFound`, which is maintained by Section 2.2.126 [`Staff_collecting_engraver`], page 323. This engraver usually creates `BarNumber` grobs, but when `centerBarNumbers` is true, it makes `CenteredBarNumber` grobs instead.

Properties (read)

`alternativeNumber` (integer)
: When set, the index of the current `\alternative` element, starting from one. Not set outside of alternatives. Note the distinction from volta number: an alternative may pertain to multiple volte.

`alternativeNumberingStyle` (symbol)
: The scheme and style for numbering bars in repeat alternatives. If not set (the default), bar numbers continue through alternatives. Can be set to `numbers` to reset the bar number at each alternative, or set to `numbers-with-letters` to reset and also include letter suffixes.

`barNumberFormatter` (procedure)
: A procedure that takes a bar number, measure position, and alternative number and returns a markup of the bar number to print.

`barNumberVisibility` (procedure)
: A procedure that takes a bar number and a measure position and returns whether the corresponding bar number should be printed. Note that the actual print-out of bar numbers is controlled with the `break-visibility` property.

  The following procedures are predefined:

  `all-bar-numbers-visible`
  : Enable bar numbers for all bars, including the first one and broken bars (which get bar numbers in parentheses).

  `first-bar-number-invisible`
  : Enable bar numbers for all bars (including broken bars) except the first one. If the first bar is broken, it doesn't get a bar number either.

  `first-bar-number-invisible-save-broken-bars`
  : Enable bar numbers for all bars (including broken bars) except the first one. A broken first bar gets a bar number.

  `first-bar-number-invisible-and-no-parenthesized-bar-numbers`
  : Enable bar numbers for all bars except the first bar and broken bars. This is the default.

`(every-nth-bar-number-visible n)`
: Assuming n is value 2, for example, this enables bar numbers for bars 2, 4, 6, etc.

`(modulo-bar-number-visible n m)`
: If bar numbers 1, 4, 7, etc., should be enabled, n (the modulo) must be set to 3 and m (the division remainder) to 1.

`centerBarNumbers` (boolean)
: Whether to center bar numbers in their measure instead of aligning them on the bar line.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

This engraver creates the following layout object(s): `BarNumber` (page 361), and `CenteredBarNumber` (page 375).

`Bar_number_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.10 Beam_collision_engraver

Help beams avoid colliding with notes and clefs in other voices.

`Beam_collision_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.11 Beam_engraver

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `Beam` (page 366).

`Beam_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `NullVoice` (page 168), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.12 Beam_performer

Music types accepted: `beam-event` (page 46),

`Beam_performer` is part of the following context(s) in `\midi`: `ChordNames` (page 63), `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `NullVoice` (page 168), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.13 Beat_engraver

This engraver is just a functionally identical copy of Section 2.2.14 [`Beat_performer`], page 288, used for visualising its effects. You can also use it for showcasing the effects of the current `beatStructure`.

Music types accepted: `articulation-event` (page 45), and `note-event` (page 50),

Properties (read)

`barExtraVelocity` (integer)
: Extra MIDI velocity added by the '`Beat_performer`' at the start of each measure.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beatExtraVelocity` (integer)
: Extra MIDI velocity added by the '`Beat_performer`' at the start of each beat.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

`timing` (boolean)
: Keep administration of measure length, position, bar number, etc.? Switch off for cadenzas.

`Beat_engraver` is not part of any context

2.2.14 Beat_performer

This performer is intended for instantiation in '`Voice`'-like contexts. The context variable `beatExtraVelocity` is used for adding extra MIDI velocity at each beat (default 15) in accordance with `beatStructure` and an additional `barExtraVelocity` (default 10) at the start of each bar.

This is done by adding corresponding `\accent` and `\marcato` events when such note events are encountered.

Off-beat manual use of `\accent` or `\marcato` causes autogeneration of the next on-beat accent to be skipped.

Music types accepted: `articulation-event` (page 45), and `note-event` (page 50),

Properties (read)

`barExtraVelocity` (integer)
: Extra MIDI velocity added by the '`Beat_performer`' at the start of each measure.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beatExtraVelocity` (integer)
: Extra MIDI velocity added by the '`Beat_performer`' at the start of each beat.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

`timing` (boolean)
: Keep administration of measure length, position, bar number, etc.? Switch off for cadenzas.

`Beat_performer` is not part of any context

2.2.15 Bend_engraver

Create fall spanners.

Music types accepted: `bend-after-event` (page 46),

This engraver creates the following layout object(s): `BendAfter` (page 368).

`Bend_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.16 Bend_spanner_engraver

Engraver to print a BendSpanner.

Music types accepted: `bend-span-event` (page 46), `note-event` (page 50), and `string-number-event` (page 53),

Properties (read)

`stringFretFingerList` (list)
: A list containg three entries. In `TabVoice` and `FretBoards` they determine the string, fret and finger to use

`supportNonIntegerFret` (boolean)
: If set in `Score` the `TabStaff` will print micro-tones as '$2\frac{1}{2}$'

Properties (write)

`stringFretFingerList` (list)
: A list containg three entries. In `TabVoice` and `FretBoards` they determine the string, fret and finger to use

`supportNonIntegerFret` (boolean)
: If set in `Score` the `TabStaff` will print micro-tones as '$2\frac{1}{2}$'

This engraver creates the following layout object(s): `BendSpanner` (page 369).

`Bend_spanner_engraver` is part of the following context(s) in `\layout`: `TabVoice` (page 240).

2.2.17 Break_align_engraver

Align grobs with corresponding `break-align-symbols` into groups, and order the groups according to `breakAlignOrder`. The left edge of the alignment gets a separate group, with a symbol `left-edge`.

This engraver creates the following layout object(s): `BreakAlignGroup` (page 371), `BreakAlignment` (page 372), and `LeftEdge` (page 432).

`Break_align_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.18 Breathing_sign_engraver

Create a breathing sign.

Music types accepted: `breathing-event` (page 46),

This engraver creates the following layout object(s): `BreathingSign` (page 373).

`Breathing_sign_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.19 Centered_bar_number_align_engraver

Group measure-centered bar numbers in a `CenteredBarNumberLineSpanner` so they end up on the same vertical position.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `CenteredBarNumberLineSpanner` (page 376).

`Centered_bar_number_align_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.20 Chord_name_engraver

Catch note and rest events and generate the appropriate chordname.

Music types accepted: `note-event` (page 50), and `rest-event` (page 51),

Properties (read)

`chordChanges` (boolean)
: Only show changes in chords scheme?

`chordNameExceptions` (list)
: An alist of chord exceptions. Contains (*`chord . markup`*) entries.

`chordNameExceptions` (list)
: An alist of chord exceptions. Contains (*`chord . markup`*) entries.

`chordNameFunction` (procedure)
: The function that converts lists of pitches to chord names.

`chordNoteNamer` (procedure)
: A function that converts from a pitch object to a text markup. Used for single pitches.

`chordRootNamer` (procedure)
: A function that converts from a pitch object to a text markup. Used for chords.

`lastChord` (markup)
: Last chord, used for detecting chord changes.

`majorSevenSymbol` (markup)
: How should the major 7th be formatted in a chord name?

`noChordSymbol` (markup)
: Markup to be displayed for rests in a ChordNames context.

Properties (write)

`lastChord` (markup)
: Last chord, used for detecting chord changes.

This engraver creates the following layout object(s): `ChordName` (page 377).

`Chord_name_engraver` is part of the following context(s) in `\layout`: `ChordNames` (page 63).

2.2.21 `Chord_tremolo_engraver`

Generate beams for tremolo repeats.

Music types accepted: `tremolo-span-event` (page 54),

This engraver creates the following layout object(s): `Beam` (page 366).

`Chord_tremolo_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.22 `Clef_engraver`

Determine and set reference point for pitches.

Properties (read)

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

This engraver creates the following layout object(s): `Clef` (page 378), and `ClefModifier` (page 381).

`Clef_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.23 `Cluster_spanner_engraver`

Engrave a cluster using `Spanner` notation.

Music types accepted: `cluster-note-event` (page 46),

This engraver creates the following layout object(s): `ClusterSpanner` (page 382), and `ClusterSpannerBeacon` (page 383).

`Cluster_spanner_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.24 `Collision_engraver`

Collect `NoteColumns`, and as soon as there are two or more, put them in a `NoteCollision` object.

This engraver creates the following layout object(s): `NoteCollision` (page 453).

`Collision_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.25 `Completion_heads_engraver`

This engraver replaces `Note_heads_engraver`. It plays some trickery to break long notes and automatically tie them into the next measure.

Music types accepted: `note-event` (page 50),

Properties (read)

`completionFactor` (an exact rational or procedure)
: When `Completion_heads_engraver` and `Completion_rest_engraver` need to split a note or rest with a scaled duration, such as `c2*3`, this specifies the scale factor to use for the newly-split notes and rests created by the engraver.

  If `#f`, the completion engraver uses the scale-factor of each duration being split.

  If set to a callback procedure, that procedure is called with the context of the completion engraver, and the duration to be split.

`completionUnit` (moment)
: Sub-bar unit of completion.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`timing` (boolean)
: Keep administration of measure length, position, bar number, etc.? Switch off for cadenzas.

Properties (write)

`completionBusy` (boolean)
: Whether a completion-note head is playing.

This engraver creates the following layout object(s): `NoteHead` (page 455), `Tie` (page 500), and `TieColumn` (page 501).

`Completion_heads_engraver` is not part of any context

2.2.26 `Completion_rest_engraver`

This engraver replaces `Rest_engraver`. It plays some trickery to break long rests into the next measure.

Music types accepted: `rest-event` (page 51),

Properties (read)

`completionFactor` (an exact rational or procedure)
: When `Completion_heads_engraver` and `Completion_rest_engraver` need to split a note or rest with a scaled duration, such as `c2*3`, this specifies the scale factor to use for the newly-split notes and rests created by the engraver.

  If `#f`, the completion engraver uses the scale-factor of each duration being split.

  If set to a callback procedure, that procedure is called with the context of the completion engraver, and the duration to be split.

`completionUnit` (moment)
: Sub-bar unit of completion.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

Properties (write)

`restCompletionBusy` (boolean)
: Signal whether a completion-rest is active.

This engraver creates the following layout object(s): `Rest` (page 469).

`Completion_rest_engraver` is not part of any context

2.2.27 `Concurrent_hairpin_engraver`

Collect concurrent hairpins.

`Concurrent_hairpin_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.28 `Control_track_performer`

Properties (read)

`midiSkipOffset` (moment)
: This is the accrued MIDI offset to account for time skipped via `skipTypesetting`.

`Control_track_performer` is part of the following context(s) in `\midi`: `Score` (page 197).

2.2.29 `Cue_clef_engraver`

Determine and set reference point for pitches in cued voices.

Properties (read)

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

This engraver creates the following layout object(s): `ClefModifier` (page 381), `CueClef` (page 387), and `CueEndClef` (page 389).

`Cue_clef_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.30 Custos_engraver

Engrave custodes.

This engraver creates the following layout object(s): Custos (page 392).

Custos_engraver is part of the following context(s) in \layout: MensuralStaff (page 145), PetrucciStaff (page 171), and VaticanaStaff (page 251).

2.2.31 Default_bar_line_engraver

This engraver determines what kind of automatic bar lines should be produced, and sets whichBar accordingly. It should be at the same level as Section 2.2.143 [Timing_translator], page 329.

Properties (read)

automaticBars (boolean)
: If set to false then bar lines will not be printed automatically; they must be explicitly created with a \bar command. Unlike the \cadenzaOn keyword, measures are still counted. Bar line generation will resume according to that count if this property is unset.

barAlways (boolean)
: If set to true a bar line is drawn after each note.

defaultBarType (string)
: Set the default type of bar line. See whichBar for information on available bar types.

  This variable is read by Section "Timing_translator" in *Internals Reference* at Section "Score" in *Internals Reference* level.

measureStartNow (boolean)
: True at the beginning of a measure.

Properties (write)

whichBar (string)
: This property is read to determine what type of bar line to create.

  Example:

```
\set Staff.whichBar = ".|:"
```

  This will create a start-repeat bar in this staff only. Valid values are described in scm/bar-line.scm.

Default_bar_line_engraver is part of the following context(s) in \layout: Score (page 197).

2.2.32 Dot_column_engraver

Engrave dots on dotted notes shifted to the right of the note. If omitted, then dots appear on top of the notes.

This engraver creates the following layout object(s): DotColumn (page 394).

Dot_column_engraver is part of the following context(s) in \layout: DrumStaff (page 76), GregorianTranscriptionStaff (page 101), KievanStaff (page 122), MensuralStaff (page 145), PetrucciStaff (page 171), RhythmicStaff (page 194), Staff (page 220), TabStaff (page 231), and VaticanaStaff (page 251).

2.2.33 Dots_engraver

Create Section 3.1.39 [Dots], page 394, objects for Section 3.2.111 [rhythmic-head-interface], page 577s.

This engraver creates the following layout object(s): Dots (page 394).

Dots_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.34 Double_percent_repeat_engraver

Make double measure repeats.

Music types accepted: double-percent-event (page 47),

Properties (read)

countPercentRepeats (boolean)
: If set, produce counters for percent repeats.

measureLength (moment)
: Length of one measure in the current time signature.

repeatCountVisibility (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when countPercentRepeats is set.

Properties (write)

forbidBreak (boolean)
: If set to #t, prevent a line break at this point.

This engraver creates the following layout object(s): DoublePercentRepeat (page 395), and DoublePercentRepeatCounter (page 396).

Double_percent_repeat_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.35 Drum_note_performer

Play drum notes.

Music types accepted: articulation-event (page 45), note-event (page 50), and tie-event (page 54),

Drum_note_performer is part of the following context(s) in \midi: DrumVoice (page 82).

2.2.36 Drum_notes_engraver

Generate drum note heads.

Music types accepted: note-event (page 50),

Properties (read)

drumStyleTable (hash table)
: A hash table which maps drums to layout settings. Predefined values: ‘drums-style’, ‘agostini-drums-style’, ‘timbales-style’, ‘congas-style’, ‘bongos-style’, and ‘percussion-style’.

The layout style is a hash table, containing the drum-pitches (e.g., the symbol '`hihat`') as keys, and a list (*`notehead-style script vertical-position`*) as values.

This engraver creates the following layout object(s): `NoteHead` (page 455), and `Script` (page 470).

`Drum_notes_engraver` is part of the following context(s) in `\layout`: `DrumVoice` (page 82).

2.2.37 Duration_line_engraver

Engraver to print a line representing the duration of a rhythmic event like `NoteHead`, `NoteColumn` or `Rest`.

Music types accepted: `duration-line-event` (page 47),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`endAtSkip` (boolean)
: End `DurationLine` grob on `skip-event`

`startAtNoteColumn` (boolean)
: Start `DurationLine` grob at entire `NoteColumn`.

`startAtSkip` (boolean)
: Start `DurationLine` grob at `skip-event`.

This engraver creates the following layout object(s): `DurationLine` (page 399).

`Duration_line_engraver` is not part of any context

2.2.38 Dynamic_align_engraver

Align hairpins and dynamic texts on a horizontal line.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `DynamicLineSpanner` (page 401).

`Dynamic_align_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `Dynamics` (page 92), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.39 Dynamic_engraver

Create hairpins, dynamic texts and dynamic text spanners.

Music types accepted: `absolute-dynamic-event` (page 45), `break-span-event` (page 46), and `span-dynamic-event` (page 52),

Properties (read)

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

This engraver creates the following layout object(s): `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

`Dynamic_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `Dynamics` (page 92), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.40 Dynamic_performer

Music types accepted: `absolute-dynamic-event` (page 45), `crescendo-event` (page 47), and `decrescendo-event` (page 47),

Properties (read)

`dynamicAbsoluteVolumeFunction` (procedure)
: A procedure that takes one argument, the text value of a dynamic event, and returns the absolute volume of that dynamic event.

`instrumentEqualizer` (procedure)
: A function taking a string (instrument name), and returning a `(`*`min . max`*`)` pair of numbers for the loudness range of the instrument.

`midiInstrument` (string)
: Name of the MIDI instrument to use.

`midiMaximumVolume` (number)
: Analogous to `midiMinimumVolume`.

`midiMinimumVolume` (number)
: Set the minimum loudness for MIDI. Ranges from 0 to 1.

`Dynamic_performer` is part of the following context(s) in `\midi`: `ChordNames` (page 63), `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.41 Episema_engraver

Create an *Editio Vaticana*-style episema line.

Music types accepted: `episema-event` (page 47),

This engraver creates the following layout object(s): `Episema` (page 405).

`Episema_engraver` is part of the following context(s) in `\layout`: `GregorianTranscriptionVoice` (page 111), and `VaticanaVoice` (page 261).

2.2.42 Extender_engraver

Create lyric extenders.

Music types accepted: `completize-extender-event` (page 47), and `extender-event` (page 47),

Properties (read)

`extendersOverRests` (boolean)
: Whether to continue extenders as they cross a rest.

This engraver creates the following layout object(s): `LyricExtender` (page 436).

`Extender_engraver` is part of the following context(s) in `\layout`: `Lyrics` (page 143).

2.2.43 Figured_bass_engraver

Make figured bass numbers.

Music types accepted: `bass-figure-event` (page 46), and `rest-event` (page 51),

Properties (read)

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

This engraver creates the following layout object(s): `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureBracket` (page 365), `BassFigureContinuation` (page 365), and `BassFigureLine` (page 366).

`Figured_bass_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `FiguredBass` (page 95), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.44 Figured_bass_position_engraver

Position figured bass alignments over notes.

This engraver creates the following layout object(s): `BassFigureAlignmentPositioning` (page 364).

`Figured_bass_position_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.45 Finger_glide_engraver

Engraver to print a line between two Fingering grobs.

Music types accepted: note-event (page 50),

This engraver creates the following layout object(s): FingerGlideSpanner (page 406).

Finger_glide_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.46 Fingering_column_engraver

Find potentially colliding scripts and put them into a FingeringColumn object; that will fix the collisions.

This engraver creates the following layout object(s): FingeringColumn (page 410).

Fingering_column_engraver is part of the following context(s) in \layout: DrumStaff (page 76), GregorianTranscriptionStaff (page 101), KievanStaff (page 122), MensuralStaff (page 145), PetrucciStaff (page 171), Staff (page 220), TabStaff (page 231), and VaticanaStaff (page 251).

2.2.47 Fingering_engraver

Create fingering scripts.

Music types accepted: fingering-event (page 47),

This engraver creates the following layout object(s): Fingering (page 408).

Fingering_engraver is part of the following context(s) in \layout: CueVoice (page 65), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), VaticanaVoice (page 261), and Voice (page 271).

2.2.48 Font_size_engraver

Put fontSize into font-size grob property.

Properties (read)

fontSize (number)
: The relative size of all grobs in a context.

Font_size_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumStaff (page 76), DrumVoice (page 82), Dynamics (page 92), FretBoards (page 97), GregorianTranscriptionStaff (page 101), GregorianTranscriptionVoice (page 111), KievanStaff (page 122), KievanVoice (page 132), Lyrics (page 143), MensuralStaff (page 145), MensuralVoice (page 155), PetrucciStaff (page 171), PetrucciVoice (page 181), RhythmicStaff (page 194), Staff (page 220), TabStaff (page 231), TabVoice (page 240), VaticanaStaff (page 251), VaticanaVoice (page 261), and Voice (page 271).

2.2.49 Footnote_engraver

Create footnote texts.

This engraver creates the following layout object(s): Footnote (page 411).

Footnote_engraver is part of the following context(s) in \layout: Score (page 197).

2.2.50 Forbid_line_break_engraver

Forbid line breaks when note heads are still playing at some point.

Properties (read)

`busyGrobs` (list)
: A queue of (*end-moment* . *grob*) cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`Forbid_line_break_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.51 Fretboard_engraver

Generate fret diagram from one or more events of type `NoteEvent`.

Music types accepted: `fingering-event` (page 47), `note-event` (page 50), and `string-number-event` (page 53),

Properties (read)

`chordChanges` (boolean)
: Only show changes in chords scheme?

`defaultStrings` (list)
: A list of strings to use in calculating frets for tablatures and fretboards if no strings are provided in the notes for the current moment.

`highStringOne` (boolean)
: Whether the first string is the string with highest pitch on the instrument. This used by the automatic string selector for tablature notation.

`maximumFretStretch` (number)
: Don't allocate frets further than this from specified frets.

`minimumFret` (number)
: The tablature auto string-selecting mechanism selects the highest string with a fret at least `minimumFret`.

`noteToFretFunction` (procedure)
: Convert list of notes and list of defined strings to full list of strings and fret numbers. Parameters: The context, a list of note events, a list of tabstring events, and the fretboard grob if a fretboard is desired.

`predefinedDiagramTable` (hash table)
: The hash table of predefined fret diagrams to use in FretBoards.

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

`tablatureFormat` (procedure)
: A function formatting a tablature note head. Called with three arguments: context, string number and, fret number. It returns the text as a markup.

This engraver creates the following layout object(s): `FretBoard` (page 412).

`Fretboard_engraver` is part of the following context(s) in `\layout`: `FretBoards` (page 97).

2.2.52 Glissando_engraver

Engrave glissandi.

Music types accepted: `glissando-event` (page 48),

Properties (read)

`glissandoMap` (list)
: A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

This engraver creates the following layout object(s): `Glissando` (page 414).

`Glissando_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.53 Grace_auto_beam_engraver

Generates one autobeam group across an entire grace phrase. As usual, any manual beaming or `\noBeam` will block autobeaming, just like setting the context property '`autoBeaming`' to `##f`.

Music types accepted: `beam-forbid-event` (page 46),

Properties (read)

`autoBeaming` (boolean)
: If set to true then beams are generated automatically.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_auto_beam_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.54 Grace_beam_engraver

Handle `Beam` events by engraving beams. If omitted, then notes are printed with flags instead of beams. Only engraves beams when we are at grace points in time.

Music types accepted: `beam-event` (page 46),

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`beatStructure` (list)
: List of `baseMoments` that are combined to make beats.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

This engraver creates the following layout object(s): `Beam` (page 366).

`Grace_beam_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.55 Grace_engraver

Set font size and other properties for grace notes.

Properties (read)

`graceSettings` (list)
: Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`Grace_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.56 Grace_spacing_engraver

Bookkeeping of shortest starting and playing notes in grace note runs.

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `GraceSpacing` (page 415).

`Grace_spacing_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.57 Grid_line_span_engraver

This engraver makes cross-staff lines: It catches all normal lines and draws a single span line across them.

This engraver creates the following layout object(s): `GridLine` (page 416).

`Grid_line_span_engraver` is not part of any context

2.2.58 Grid_point_engraver

Generate grid points.

Properties (read)

`gridInterval` (moment)
: Interval for which to generate `GridPoint`s.

This engraver creates the following layout object(s): `GridPoint` (page 416).

`Grid_point_engraver` is not part of any context

2.2.59 Grob_pq_engraver

Administrate when certain grobs (e.g., note heads) stop playing.

Properties (read)

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

Properties (write)

`busyGrobs` (list)
: A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`Grob_pq_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumStaff` (page 76), `DrumVoice` (page 82), `GregorianTranscriptionStaff` (page 101), `GregorianTranscriptionVoice` (page 111), `KievanStaff` (page 122), `KievanVoice` (page 132), `MensuralStaff` (page 145), `MensuralVoice` (page 155), `NullVoice` (page 168), `PetrucciStaff` (page 171), `PetrucciVoice` (page 181), `Staff` (page 220), `TabStaff` (page 231), `TabVoice` (page 240), `VaticanaStaff` (page 251), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.60 Horizontal_bracket_engraver

Create horizontal brackets over notes for musical analysis purposes.

Music types accepted: `note-grouping-event` (page 50),

This engraver creates the following layout object(s): `HorizontalBracket` (page 418), and `HorizontalBracketText` (page 419).

`Horizontal_bracket_engraver` is not part of any context

2.2.61 Hyphen_engraver

Create lyric hyphens, vowel transitions and distance constraints between words.

Music types accepted: `hyphen-event` (page 48), and `vowel-transition-event` (page 55),

This engraver creates the following layout object(s): `LyricHyphen` (page 436), `LyricSpace` (page 437), and `VowelTransition` (page 520).

`Hyphen_engraver` is part of the following context(s) in `\layout`: `Lyrics` (page 143).

2.2.62 Instrument_name_engraver

Create a system start text for instrument or vocal names.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`vocalName` (markup)
: Name of a vocal line.

This engraver creates the following layout object(s): `InstrumentName` (page 420).

`Instrument_name_engraver` is part of the following context(s) in `\layout`: `ChoirStaff` (page 61), `DrumStaff` (page 76), `FretBoards` (page 97), `GrandStaff` (page 99),

GregorianTranscriptionStaff (page 101), KievanStaff (page 122), Lyrics (page 143), MensuralStaff (page 145), PetrucciStaff (page 171), PianoStaff (page 192), RhythmicStaff (page 194), Staff (page 220), StaffGroup (page 229), TabStaff (page 231), and VaticanaStaff (page 251).

2.2.63 Instrument_switch_engraver

Create a cue text for taking instrument.

Properties (read)

instrumentCueName (markup)
: The name to print if another instrument is to be taken.

This engraver creates the following layout object(s): InstrumentSwitch (page 421).

Instrument_switch_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.64 Jump_engraver

Create JumpScript objects. It puts them outside all staves (which is taken from the property stavesFound). If moving this engraver to a different context, Section 2.2.126 [Staff_collecting_engraver], page 323, must move along, otherwise all marks end up on the same Y location.

Music types accepted: fine-event (page 47),

Properties (read)

stavesFound (list of grobs)
: A list of all staff-symbols found.

This engraver creates the following layout object(s): JumpScript (page 422).

Jump_engraver is part of the following context(s) in \layout: Score (page 197).

2.2.65 Keep_alive_together_engraver

This engraver collects all Hara_kiri_group_spanners that are created in contexts at or below its own. These spanners are then tied together so that one will be removed only if all are removed. For example, if a StaffGroup uses this engraver, then the staves in the group will all be visible as long as there is a note in at least one of them.

Keep_alive_together_engraver is part of the following context(s) in \layout: PianoStaff (page 192).

2.2.66 Key_engraver

Engrave a key signature.

Music types accepted: key-change-event (page 48),

Properties (read)

createKeyOnClefChange (boolean)
: Print a key signature whenever the clef is changed.

explicitKeySignatureVisibility (vector)
: 'break-visibility' function for explicit key changes. '\override' of the break-visibility property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(step . alter)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

Properties (write)

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`tonic` (pitch)
: The tonic of the current scale.

This engraver creates the following layout object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

`Key_engraver` is part of the following context(s) in `\layout`: `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), and `VaticanaStaff` (page 251).

2.2.67 Key_performer

Music types accepted: `key-change-event` (page 48),

Properties (read)

`instrumentTransposition` (pitch)
: Define the transposition of the instrument. Its value is the pitch that sounds when the instrument plays written middle C. This is used to transpose the MIDI output, and `\quote`s.

`Key_performer` is part of the following context(s) in `\midi`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.68 Kievan_ligature_engraver

Handle `Kievan_ligature_events` by glueing Kievan heads together.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `KievanLigature` (page 430).

`Kievan_ligature_engraver` is part of the following context(s) in `\layout`: `KievanVoice` (page 132).

2.2.69 Laissez_vibrer_engraver

Create laissez vibrer items.

Music types accepted: `laissez-vibrer-event` (page 48),

This engraver creates the following layout object(s): `LaissezVibrerTie` (page 430), and `LaissezVibrerTieColumn` (page 431).

`Laissez_vibrer_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.70 Ledger_line_engraver

Create the spanner to draw ledger lines, and notices objects that need ledger lines.

This engraver creates the following layout object(s): `LedgerLineSpanner` (page 432).

`Ledger_line_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.71 Ligature_bracket_engraver

Handle `Ligature_events` by engraving `Ligature` brackets.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `LigatureBracket` (page 434).

`Ligature_bracket_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `GregorianTranscriptionVoice` (page 111), `TabVoice` (page 240), and `Voice` (page 271).

2.2.72 Lyric_engraver

Engrave text for lyrics.

Music types accepted: `lyric-event` (page 48),

Properties (read)

`ignoreMelismata` (boolean)
: Ignore melismata for this Section "Lyrics" in *Internals Reference* line.

`lyricMelismaAlignment` (number)
: Alignment to use for a melisma syllable.

`searchForVoice` (boolean)
: Signal whether a search should be made of all contexts in the context hierarchy for a voice to provide rhythms for the lyrics.

This engraver creates the following layout object(s): `LyricText` (page 438).

`Lyric_engraver` is part of the following context(s) in `\layout`: `Lyrics` (page 143).

2.2.73 Lyric_performer

Music types accepted: `lyric-event` (page 48),

`Lyric_performer` is part of the following context(s) in `\midi`: `Lyrics` (page 143).

2.2.74 Mark_engraver

This engraver creates rehearsal marks.

`Mark_engraver` creates marks formatted according to the `markFormatter` context property and places them vertically outside the set of staves given in the `stavesFound` context property.

If `Mark_engraver` is added or moved to another context, `Staff_collecting_engraver` (page 323), also needs to be there so that marks appear at the intended Y location.

By default, `Mark_engravers` in multiple contexts create a common sequence of marks chosen by the `Score`-level `Mark_tracking_translator` (page 308). If independent sequences are desired, multiple `Mark_tracking_translators` must be used.

Properties (read)

`currentMarkEvent` (stream event)
: The event selected by `Mark_tracking_translator` for engraving by `Mark_engraver`.

`markFormatter` (procedure)
: A procedure taking as arguments the context and the sequence number of the rehearsal mark. It should return the formatted mark as a markup object.

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

This engraver creates the following layout object(s): `RehearsalMark` (page 465).

`Mark_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.75 Mark_tracking_translator

This translator chooses which mark `Mark_engraver` should engrave.

Music types accepted: `ad-hoc-mark-event` (page 45), and `rehearsal-mark-event` (page 51),

Properties (read)

`rehearsalMark` (integer)
: The last rehearsal mark printed.

Properties (write)

`currentMarkEvent` (stream event)
: The event selected by `Mark_tracking_translator` for engraving by `Mark_engraver`.

`rehearsalMark` (integer)
: The last rehearsal mark printed.

`Mark_tracking_translator` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.76 Measure_counter_engraver

This engraver numbers ranges of measures, which is useful in parts as an aid for counting repeated measures. There is no requirement that the affected measures be repeated, however. The user delimits the area to receive a count with `\startMeasureCount` and `\stopMeasureCount`.

Music types accepted: `measure-counter-event` (page 49),

Properties (read)

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

This engraver creates the following layout object(s): `MeasureCounter` (page 439).

`Measure_counter_engraver` is not part of any context

2.2.77 Measure_grouping_engraver

Create `MeasureGrouping` to indicate beat subdivision.

Properties (read)

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

This engraver creates the following layout object(s): `MeasureGrouping` (page 441).

`Measure_grouping_engraver` is not part of any context

2.2.78 Measure_spanner_engraver

This engraver creates spanners bounded by the columns that start and end measures in response to `\startMeasureSpanner` and `\stopMeasureSpanner`.

Music types accepted: `measure-spanner-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

This engraver creates the following layout object(s): `MeasureSpanner` (page 442).

`Measure_spanner_engraver` is not part of any context

2.2.79 Melody_engraver

Create information for context dependent typesetting decisions.

This engraver creates the following layout object(s): `MelodyItem` (page 443).

`Melody_engraver` is not part of any context

2.2.80 Mensural_ligature_engraver

Handle `Mensural_ligature_events` by glueing special ligature heads together.

Music types accepted: `ligature-event` (page 48),

This engraver creates the following layout object(s): `MensuralLigature` (page 444).

`Mensural_ligature_engraver` is part of the following context(s) in `\layout`: `MensuralVoice` (page 155), and `PetrucciVoice` (page 181).

2.2.81 Merge_mmrest_numbers_engraver

Engraver to merge multi-measure rest numbers in multiple voices.

This works by gathering all multi-measure rest numbers at a time step. If they all have the same text and there are at least two only the first one is retained and the others are hidden.

`Merge_mmrest_numbers_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.82 Merge_rests_engraver

Engraver to merge rests in multiple voices on the same staff. This works by gathering all rests at a time step. If they are all of the same length and there are at least two they are moved to the correct location as if there were one voice.

Properties (read)

`suspendRestMerging` (boolean)
: When using the Merge_rest_engraver do not merge rests when this is set to true.

`Merge_rests_engraver` is not part of any context

2.2.83 Metronome_mark_engraver

Engrave metronome marking. This delegates the formatting work to the function in the `metronomeMarkFormatter` property. The mark is put over all staves. The staves are taken from the `stavesFound` property, which is maintained by Section 2.2.126 [`Staff_collecting_engraver`], page 323.

Music types accepted: `tempo-change-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`metronomeMarkFormatter` (procedure)
: How to produce a metronome markup. Called with two arguments: a `TempoChangeEvent` and context.

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`tempoHideNote` (boolean)
: Hide the note = count in tempo marks.

This engraver creates the following layout object(s): `MetronomeMark` (page 444).

`Metronome_mark_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.84 Midi_control_change_performer

This performer listens to SetProperty events on context properties for generating MIDI control changes and prepares them for MIDI output.

Properties (read)

`midiBalance` (number)
: Stereo balance for the MIDI channel associated with the current context. Ranges from -1 to 1, where the values -1 (`#LEFT`), 0 (`#CENTER`) and 1 (`#RIGHT`) correspond to leftmost emphasis, center balance, and rightmost emphasis, respectively.

`midiChorusLevel` (number)
: Chorus effect level for the MIDI channel associated with the current context. Ranges from 0 to 1 (0=off, 1=full effect).

`midiExpression` (number)
: Expression control for the MIDI channel associated with the current context. Ranges from 0 to 1 (0=off, 1=full effect).

`midiPanPosition` (number)
: Pan position for the MIDI channel associated with the current context. Ranges from -1 to 1, where the values -1 (`#LEFT`), 0 (`#CENTER`) and 1 (`#RIGHT`) correspond to hard left, center, and hard right, respectively.

`midiReverbLevel` (number)
: Reverb effect level for the MIDI channel associated with the current context. Ranges from 0 to 1 (0=off, 1=full effect).

`Midi_control_change_performer` is part of the following context(s) in `\midi`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.85 Multi_measure_rest_engraver

Engrave multi-measure rests that are produced with 'R'. It reads `measureStartNow` and `internalBarNumber` to determine what number to print over the Section 3.1.82 [`MultiMeasureRest`], page 446.

Music types accepted: `multi-measure-articulation-event` (page 49), `multi-measure-rest-event` (page 49), and `multi-measure-text-event` (page 49),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

This engraver creates the following layout object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

`Multi_measure_rest_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.86 New_fingering_engraver

Create fingering scripts for notes in a new chord. This engraver is ill-named, since it also takes care of articulations and harmonic note heads.

Properties (read)

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

This engraver creates the following layout object(s): `Fingering` (page 408), `Script` (page 470), `StringNumber` (page 485), and `StrokeFinger` (page 487).

`New_fingering_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.87 Note_head_line_engraver

Engrave a line between two note heads in a staff switch if `followVoice` is set.

Properties (read)

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

This engraver creates the following layout object(s): `VoiceFollower` (page 516).

`Note_head_line_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.88 Note_heads_engraver

Generate note heads.

Music types accepted: note-event (page 50),

Properties (read)

middleCPosition (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at middleCClefPosition and middleCOffset.

staffLineLayoutFunction (procedure)
: Layout of staff lines, traditional, or semitone.

This engraver creates the following layout object(s): NoteHead (page 455).

Note_heads_engraver is part of the following context(s) in \layout: CueVoice (page 65), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), NullVoice (page 168), PetrucciVoice (page 181), VaticanaVoice (page 261), and Voice (page 271).

2.2.89 Note_name_engraver

Print pitches as words.

Music types accepted: note-event (page 50),

Properties (read)

noteNameFunction (procedure)
: Function used to convert pitches into strings and markups.

noteNameSeparator (string)
: String used to separate simultaneous NoteName objects.

printAccidentalNames (boolean or symbol)
: Print accidentals in the NoteNames context.

printNotesLanguage (string)
: Use a specific language in the NoteNames context.

printOctaveNames (boolean or symbol)
: Print octave marks in the NoteNames context.

This engraver creates the following layout object(s): NoteName (page 456).

Note_name_engraver is part of the following context(s) in \layout: NoteNames (page 166).

2.2.90 Note_performer

Music types accepted: articulation-event (page 45), breathing-event (page 46), note-event (page 50), and tie-event (page 54),

Note_performer is part of the following context(s) in \midi: ChordNames (page 63), CueVoice (page 65), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.91 Note_spacing_engraver

Generate NoteSpacing, an object linking horizontal lines for use in spacing.

This engraver creates the following layout object(s): NoteSpacing (page 456).

Note_spacing_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice

(page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.92 Ottava_spanner_engraver

Create a text spanner when the ottavation property changes.

Music types accepted: ottava-event (page 50),

Properties (read)

currentMusicalColumn (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

middleCOffset (number)
: The offset of middle C from the position given by middleCClefPosition This is used for ottava brackets.

ottavation (markup)
: If set, the text for an ottava spanner. Changing this creates a new text spanner.

This engraver creates the following layout object(s): OttavaBracket (page 457).

Ottava_spanner_engraver is part of the following context(s) in \layout: GregorianTranscriptionStaff (page 101), KievanStaff (page 122), MensuralStaff (page 145), PetrucciStaff (page 171), Staff (page 220), and VaticanaStaff (page 251).

2.2.93 Output_property_engraver

Apply a procedure to any grob acknowledged.

Music types accepted: apply-output-event (page 45),

Output_property_engraver is part of the following context(s) in \layout: ChoirStaff (page 61), ChordNames (page 63), CueVoice (page 65), DrumStaff (page 76), DrumVoice (page 82), Dynamics (page 92), FretBoards (page 97), GrandStaff (page 99), GregorianTranscriptionStaff (page 101), GregorianTranscriptionVoice (page 111), KievanStaff (page 122), KievanVoice (page 132), MensuralStaff (page 145), MensuralVoice (page 155), PetrucciStaff (page 171), PetrucciVoice (page 181), PianoStaff (page 192), RhythmicStaff (page 194), Score (page 197), Staff (page 220), StaffGroup (page 229), TabStaff (page 231), TabVoice (page 240), VaticanaStaff (page 251), VaticanaVoice (page 261), and Voice (page 271).

2.2.94 Page_turn_engraver

Decide where page turns are allowed to go.

Music types accepted: break-event (page 46),

Properties (read)

minimumPageTurnLength (moment)
: Minimum length of a rest for a page turn to be allowed.

minimumRepeatLengthForPageTurn (moment)
: Minimum length of a repeated section for a page turn to be allowed within that section.

Page_turn_engraver is not part of any context

2.2.95 Paper_column_engraver

Take care of generating columns.

This engraver decides whether a column is breakable. The default is that a column is always breakable. However, every `Bar_engraver` that does not have a barline at a certain point will set `forbidBreaks` in the score context to stop line breaks. In practice, this means that you can make a break point by creating a bar line (assuming that there are no beams or notes that prevent a break point).

Music types accepted: `break-event` (page 46), and `label-event` (page 48),

Properties (read)

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

Properties (write)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

This engraver creates the following layout object(s): `NonMusicalPaperColumn` (page 452), and `PaperColumn` (page 458).

`Paper_column_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.96 Parenthesis_engraver

Parenthesize objects whose `parenthesize` property is `#t`.

This engraver creates the following layout object(s): `Parentheses` (page 459).

`Parenthesis_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.97 Part_combine_engraver

Part combine engraver for orchestral scores: Print markings ‘a2’, ‘Solo’, ‘Solo II’, and ‘unisono’.

Music types accepted: `note-event` (page 50), and `part-combine-event` (page 50),

Properties (read)

`aDueText` (markup)
: Text to print at a unisono passage.

`partCombineTextsOnNote` (boolean)
: Print part-combine texts only on the next note rather than immediately on rests or skips.

`printPartCombineTexts` (boolean)
: Set ‘Solo’ and ‘A due’ texts in the part combiner?

`soloIIText` (markup)
: The text for the start of a solo for voice ‘two’ when part-combining.

`soloText` (markup)
: The text for the start of a solo when part-combining.

This engraver creates the following layout object(s): `CombineTextScript` (page 383).

`Part_combine_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.98 `Percent_repeat_engraver`

Make whole measure repeats.

Music types accepted: `percent-event` (page 51),

Properties (read)

`countPercentRepeats` (boolean)
: If set, produce counters for percent repeats.

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`repeatCountVisibility` (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

This engraver creates the following layout object(s): `PercentRepeat` (page 460), and `PercentRepeatCounter` (page 461).

`Percent_repeat_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.99 `Phrasing_slur_engraver`

Print phrasing slurs. Similar to Section 2.2.117 [`Slur_engraver`], page 322.

Music types accepted: `note-event` (page 50), and `phrasing-slur-event` (page 51),

This engraver creates the following layout object(s): `PhrasingSlur` (page 462).

`Phrasing_slur_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.100 `Piano_pedal_align_engraver`

Align piano pedal symbols and brackets.

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

This engraver creates the following layout object(s): `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

`Piano_pedal_align_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.101 Piano_pedal_engraver

Engrave piano pedal symbols and brackets.

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(up updown down)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

This engraver creates the following layout object(s): `PianoPedalBracket` (page 464), `SostenutoPedal` (page 474), `SustainPedal` (page 488), and `UnaCordaPedal` (page 511).

`Piano_pedal_engraver` is part of the following context(s) in `\layout`: `Dynamics` (page 92), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.102 Piano_pedal_performer

Music types accepted: `sostenuto-event` (page 52), `sustain-event` (page 53), and `una-corda-event` (page 54),

`Piano_pedal_performer` is part of the following context(s) in `\midi`: `ChordNames` (page 63), `CueVoice` (page 65), `DrumVoice` (page 82), `Dynamics` (page 92), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.103 Pitch_squash_engraver

Set the vertical position of note heads to `squashedPosition`, if that property is set. This can be used to make a single-line staff demonstrating the rhythm of a melody.

Properties (read)

`squashedPosition` (integer)
: Vertical position of squashing for Section "Pitch_squash_engraver" in *Internals Reference*.

`Pitch_squash_engraver` is part of the following context(s) in `\layout`: `NullVoice` (page 168), and `RhythmicStaff` (page 194).

2.2.104 `Pitched_trill_engraver`

Print the bracketed note head after a note head with trill.

This engraver creates the following layout object(s): `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), and `TrillPitchHead` (page 506).

`Pitched_trill_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.105 `Pure_from_neighbor_engraver`

Coordinates items that get their pure heights from their neighbors.

`Pure_from_neighbor_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.106 `Repeat_acknowledge_engraver`

Acknowledge repeated music, and convert the contents of `repeatCommands` into an appropriate setting for `whichBar`.

Music types accepted: `fine-event` (page 47), `section-event` (page 52), `segno-event` (page 52), and `volta-span-event` (page 55),

Properties (read)

`defaultBarType` (string)
: Set the default type of bar line. See `whichBar` for information on available bar types.

  This variable is read by Section "Timing_translator" in *Internals Reference* at Section "Score" in *Internals Reference* level.

`doubleRepeatSegnoType` (string)
: Set the default bar line for the combinations double repeat with segno. Default is '`:|.S.|:`'.

`doubleRepeatType` (string)
: Set the default bar line for double repeats.

`endRepeatSegnoType` (string)
: Set the default bar line for the combinations ending of repeat with segno. Default is '`:|.S`'.

`endRepeatType` (string)
: Set the default bar line for the ending of repeats.

`fineBarType` (string)
: The bar line for `\fine`. See `whichBar` for information on available bar types.

`fineSegnoType` (string)
: Set the default bar line for a requested segno with fine. Default is '`|.S`'.

`fineStartRepeatSegnoType` (string)
: Set the default bar line for the combinations beginning of repeat with segno and fine. Default is '`|.S.|:`'.

`repeatCommands` (list)
: This property is a list of commands of the form `(list 'volta x)`, where *x* is a string or `#f`. `'end-repeat` is also accepted as a command.

`sectionBarType` (string)
: The bar line for `\section`. See `whichBar` for information on available bar types.

`segnoType` (string)
: Set the default bar line for a requested segno. Default is '`S`'.

`startRepeatSegnoType` (string)
: Set the default bar line for the combinations beginning of repeat with segno. Default is '`S.|:`'.

`startRepeatType` (string)
: Set the default bar line for the beginning of repeats.

`underlyingRepeatType` (string)
: Set the bar line to use at points of repetition or departure where no bar line would normally appear, for example at the end of a system broken in mid measure where the next system begins with a segno.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

```
\set Staff.whichBar = ".|:"
```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

Properties (write)

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

```
\set Staff.whichBar = ".|:"
```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

`Repeat_acknowledge_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.107 Repeat_tie_engraver

Create repeat ties.

Music types accepted: `repeat-tie-event` (page 51),

This engraver creates the following layout object(s): `RepeatTie` (page 467), and `RepeatTieColumn` (page 469).

`Repeat_tie_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.108 Rest_collision_engraver

Handle collisions of rests.

Properties (read)

`busyGrobs` (list)
: A queue of `(end-moment . grob)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

This engraver creates the following layout object(s): `RestCollision` (page 470).

`Rest_collision_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.109 Rest_engraver

Engrave rests.

Music types accepted: `rest-event` (page 51),

Properties (read)

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

This engraver creates the following layout object(s): `Rest` (page 469).

`Rest_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.110 Rhythmic_column_engraver

Generate `NoteColumn`, an object that groups stems, note heads, and rests.

This engraver creates the following layout object(s): `NoteColumn` (page 454).

`Rhythmic_column_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.111 Script_column_engraver

Find potentially colliding scripts and put them into a `ScriptColumn` object; that will fix the collisions.

This engraver creates the following layout object(s): `ScriptColumn` (page 471).

`Script_column_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.112 Script_engraver

Handle note scripted articulations.

Music types accepted: `articulation-event` (page 45),

Properties (read)

`scriptDefinitions` (list)
: The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

This engraver creates the following layout object(s): `Script` (page 470).

`Script_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `Dynamics` (page 92), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.113 Script_row_engraver

Determine order in horizontal side position elements.

This engraver creates the following layout object(s): `ScriptRow` (page 472).

`Script_row_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.114 Separating_line_group_engraver

Generate objects for computing spacing parameters.

Properties (read)

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

Properties (write)

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

This engraver creates the following layout object(s): `StaffSpacing` (page 479).

`Separating_line_group_engraver` is part of the following context(s) in `\layout`: `ChordNames` (page 63), `DrumStaff` (page 76), `FiguredBass` (page 95), `FretBoards` (page 97), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `NoteNames` (page 166), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.115 Show_control_points_engraver

Create grobs to visualize control points of Bézier curves (ties and slurs) for ease of tweaking.

This engraver creates the following layout object(s): `ControlPoint` (page 385), and `ControlPolygon` (page 386).

`Show_control_points_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.116 Slash_repeat_engraver

Make beat repeats.

Music types accepted: `repeat-slash-event` (page 51),

This engraver creates the following layout object(s): `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

Slash_repeat_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.117 Slur_engraver

Build slur grobs from slur events.

Music types accepted: note-event (page 50), and slur-event (page 52),

Properties (read)

doubleSlurs (boolean)
: If set, two slurs are created for every slurred note, one above and one below the chord.

slurMelismaBusy (boolean)
: Signal if a slur is present.

This engraver creates the following layout object(s): Slur (page 472).

Slur_engraver is part of the following context(s) in \layout: CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), NullVoice (page 168), PetrucciVoice (page 181), TabVoice (page 240), and Voice (page 271).

2.2.118 Slur_performer

Music types accepted: slur-event (page 52),

Slur_performer is part of the following context(s) in \midi: ChordNames (page 63), CueVoice (page 65), DrumVoice (page 82), GregorianTranscriptionVoice (page 111), KievanVoice (page 132), MensuralVoice (page 155), NullVoice (page 168), PetrucciVoice (page 181), TabVoice (page 240), VaticanaVoice (page 261), and Voice (page 271).

2.2.119 Spacing_engraver

Make a SpacingSpanner and do bookkeeping of shortest starting and playing notes.

Music types accepted: spacing-section-event (page 52),

Properties (read)

currentCommandColumn (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

currentMusicalColumn (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

proportionalNotationDuration (moment)
: Global override for shortest-playing duration. This is used for switching on proportional notation.

This engraver creates the following layout object(s): SpacingSpanner (page 476).

Spacing_engraver is part of the following context(s) in \layout: Score (page 197).

2.2.120 Span_arpeggio_engraver

Make arpeggios that span multiple staves.

Properties (read)

connectArpeggios (boolean)
: If set, connect arpeggios across piano staff.

This engraver creates the following layout object(s): `Arpeggio` (page 355).

`Span_arpeggio_engraver` is part of the following context(s) in `\layout`: `ChoirStaff` (page 61), `GrandStaff` (page 99), `PianoStaff` (page 192), and `StaffGroup` (page 229).

2.2.121 Span_bar_engraver

Make cross-staff bar lines: It catches all normal bar lines and draws a single span bar across them.

This engraver creates the following layout object(s): `SpanBar` (page 477).

`Span_bar_engraver` is part of the following context(s) in `\layout`: `GrandStaff` (page 99), `PianoStaff` (page 192), and `StaffGroup` (page 229).

2.2.122 Span_bar_stub_engraver

Make stubs for span bars in all contexts that the span bars cross.

This engraver creates the following layout object(s): `SpanBarStub` (page 478).

`Span_bar_stub_engraver` is part of the following context(s) in `\layout`: `ChoirStaff` (page 61), `GrandStaff` (page 99), `PianoStaff` (page 192), and `StaffGroup` (page 229).

2.2.123 Span_stem_engraver

Connect cross-staff stems to the stems above in the system

This engraver creates the following layout object(s): `Stem` (page 481).

`Span_stem_engraver` is not part of any context

2.2.124 Spanner_break_forbid_engraver

Forbid breaks in certain spanners.

`Spanner_break_forbid_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.125 Spanner_tracking_engraver

Helper for creating spanners attached to other spanners. If a spanner has the `sticky-grob-interface`, the engraver tracks the spanner contained in its `sticky-host` object. When the host ends, the sticky spanner attached to it has its end announced too.

`Spanner_tracking_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.126 Staff_collecting_engraver

Maintain the `stavesFound` variable.

Properties (read)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

Properties (write)

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`Staff_collecting_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `Score` (page 197), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.127 Staff_performer

Properties (read)

`midiChannelMapping` (symbol)
: How to map MIDI channels: per `staff` (default), `instrument` or `voice`.

`midiMergeUnisons` (boolean)
: If true, output only one MIDI note-on event when notes with the same pitch, in the same MIDI-file track, overlap.

`midiSkipOffset` (moment)
: This is the accrued MIDI offset to account for time skipped via `skipTypesetting`.

`Staff_performer` is part of the following context(s) in `\midi`: `ChordNames` (page 63), `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `Lyrics` (page 143), `MensuralStaff` (page 145), `NoteNames` (page 166), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.128 Staff_symbol_engraver

Create the constellation of five (default) staff lines.

Music types accepted: `staff-span-event` (page 53),

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Staff_symbol_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `KievanStaff` (page 122), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), `TabStaff` (page 231), and `VaticanaStaff` (page 251).

2.2.129 Stanza_number_align_engraver

This engraver ensures that stanza numbers are neatly aligned.

`Stanza_number_align_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.130 Stanza_number_engraver

Engrave stanza numbers.

Properties (read)

`stanza` (markup)
: Stanza 'number' to print before the start of a verse. Use in `Lyrics` context.

This engraver creates the following layout object(s): `StanzaNumber` (page 481).

`Stanza_number_engraver` is part of the following context(s) in `\layout`: `Lyrics` (page 143).

2.2.131 Stem_engraver

Create stems, flags and single-stem tremolos. It also works together with the beam engraver for overriding beaming.

Music types accepted: `tremolo-event` (page 54), and `tuplet-span-event` (page 54),

Properties (read)

`stemLeftBeamCount` (integer)
: Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
: See `stemLeftBeamCount`.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

  ```
  \set Staff.whichBar = ".|:"
  ```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

This engraver creates the following layout object(s): `Flag` (page 410), `Stem` (page 481), `StemStub` (page 483), and `StemTremolo` (page 484).

`Stem_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), and `Voice` (page 271).

2.2.132 System_start_delimiter_engraver

Create a system start delimiter (i.e., a `SystemStartBar`, `SystemStartBrace`, `SystemStartBracket` or `SystemStartSquare` spanner).

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`systemStartDelimiter` (symbol)
: Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

`systemStartDelimiterHierarchy` (pair)
: A nested list, indicating the nesting of a start delimiters.

This engraver creates the following layout object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

`System_start_delimiter_engraver` is part of the following context(s) in `\layout`: `ChoirStaff` (page 61), `GrandStaff` (page 99), `PianoStaff` (page 192), `Score` (page 197), and `StaffGroup` (page 229).

2.2.133 Tab_note_heads_engraver

Generate one or more tablature note heads from event of type `NoteEvent`.

Music types accepted: `fingering-event` (page 47), `note-event` (page 50), and `string-number-event` (page 53),

Properties (read)

`defaultStrings` (list)
: A list of strings to use in calculating frets for tablatures and fretboards if no strings are provided in the notes for the current moment.

`fretLabels` (list)
: A list of strings or Scheme-formatted markups containing, in the correct order, the labels to be used for lettered frets in tablature.

`highStringOne` (boolean)
: Whether the first string is the string with highest pitch on the instrument. This used by the automatic string selector for tablature notation.

`maximumFretStretch` (number)
: Don't allocate frets further than this from specified frets.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`minimumFret` (number)
: The tablature auto string-selecting mechanism selects the highest string with a fret at least `minimumFret`.

`noteToFretFunction` (procedure)
: Convert list of notes and list of defined strings to full list of strings and fret numbers. Parameters: The context, a list of note events, a list of tabstring events, and the fretboard grob if a fretboard is desired.

`stringOneTopmost` (boolean)
: Whether the first string is printed on the top line of the tablature.

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

`tablatureFormat` (procedure)
: A function formatting a tablature note head. Called with three arguments: context, string number and, fret number. It returns the text as a markup.

`tabStaffLineLayoutFunction` (procedure)
: A function determining the staff position of a tablature note head. Called with two arguments: the context and the string.

This engraver creates the following layout object(s): `TabNoteHead` (page 494).

`Tab_note_heads_engraver` is part of the following context(s) in `\layout`: `TabVoice` (page 240).

2.2.134 `Tab_staff_symbol_engraver`

Create a tablature staff symbol, but look at `stringTunings` for the number of lines.

Properties (read)

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

This engraver creates the following layout object(s): `StaffSymbol` (page 480).

`Tab_staff_symbol_engraver` is part of the following context(s) in `\layout`: `TabStaff` (page 231).

2.2.135 `Tab_tie_follow_engraver`

Adjust TabNoteHead properties when a tie is followed by a slur or glissando.

`Tab_tie_follow_engraver` is part of the following context(s) in `\layout`: `TabVoice` (page 240).

2.2.136 Tempo_performer

Properties (read)

`tempoWholesPerMinute` (moment)
: The tempo in whole notes per minute.

`Tempo_performer` is part of the following context(s) in `\midi`: `Score` (page 197).

2.2.137 Text_engraver

Create text scripts.

Music types accepted: `text-script-event` (page 54),

This engraver creates the following layout object(s): `TextScript` (page 496).

`Text_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `Dynamics` (page 92), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.138 Text_spanner_engraver

Create text spanner from an event.

Music types accepted: `text-span-event` (page 54),

Properties (read)

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TextSpanner` (page 498).

`Text_spanner_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `Dynamics` (page 92), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), and `Voice` (page 271).

2.2.139 Tie_engraver

Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

This engraver creates the following layout object(s): `Tie` (page 500), and `TieColumn` (page 501).

`Tie_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `NoteNames` (page 166), `NullVoice` (page 168), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.140 Tie_performer

Generate ties between note heads of equal pitch.

Music types accepted: `tie-event` (page 54),

Properties (read)

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

Properties (write)

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

`Tie_performer` is part of the following context(s) in `\midi`: `ChordNames` (page 63), `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `NullVoice` (page 168), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.141 Time_signature_engraver

Create a Section 3.1.135 [`TimeSignature`], page 502, whenever `timeSignatureFraction` changes.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`initialTimeSignatureVisibility` (vector)
: break visibility for the initial time signature.

`partialBusy` (boolean)
: Signal that \partial acts at the current timestep.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

This engraver creates the following layout object(s): `TimeSignature` (page 502).

`Time_signature_engraver` is part of the following context(s) in `\layout`: `DrumStaff` (page 76), `GregorianTranscriptionStaff` (page 101), `MensuralStaff` (page 145), `PetrucciStaff` (page 171), `RhythmicStaff` (page 194), `Staff` (page 220), and `TabStaff` (page 231).

2.2.142 Time_signature_performer

Creates a MIDI time signature whenever `timeSignatureFraction` changes or a `\time` command is issued.

Music types accepted: `time-signature-event` (page 54),

Properties (read)

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

`Time_signature_performer` is part of the following context(s) in `\midi`: `Score` (page 197).

2.2.143 Timing_translator

This engraver adds the alias `Timing` to its containing context. Responsible for synchronizing timing information from staves. Normally in `Score`. In order to create polyrhythmic music, this engraver should be removed from `Score` and placed in `Staff`.

Music types accepted: `alternative-event` (page 45),

Properties (read)

`alternativeNumberingStyle` (symbol)
: The scheme and style for numbering bars in repeat alternatives. If not set (the default), bar numbers continue through alternatives. Can be set to `numbers` to reset the bar number at each alternative, or set to `numbers-with-letters` to reset and also include letter suffixes.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

Properties (write)

`alternativeNumber` (integer)
: When set, the index of the current `\alternative` element, starting from one. Not set outside of alternatives. Note the distinction from volta number: an alternative may pertain to multiple volte.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

`Timing_translator` is part of the following context(s) in `\layout`: `Score` (page 197); in `\midi`: `Score` (page 197).

2.2.144 `Trill_spanner_engraver`

Create trill spanner from an event.

Music types accepted: `trill-span-event` (page 54),

Properties (read)

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

This engraver creates the following layout object(s): `TrillSpanner` (page 507).

`Trill_spanner_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.145 `Tuplet_engraver`

Catch tuplet events and generate appropriate bracket.

Music types accepted: `tuplet-span-event` (page 54),

Properties (read)

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

This engraver creates the following layout object(s): `TupletBracket` (page 509), and `TupletNumber` (page 510).

`Tuplet_engraver` is part of the following context(s) in `\layout`: `CueVoice` (page 65), `DrumVoice` (page 82), `GregorianTranscriptionVoice` (page 111), `KievanVoice` (page 132), `MensuralVoice` (page 155), `PetrucciVoice` (page 181), `TabVoice` (page 240), `VaticanaVoice` (page 261), and `Voice` (page 271).

2.2.146 `Tweak_engraver`

Read the `tweaks` property from the originating event, and set properties.

`Tweak_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.2.147 Vaticana_ligature_engraver

Handle ligatures by glueing special ligature heads together.

Music types accepted: `ligature-event` (page 48), and `pes-or-flexa-event` (page 51),

This engraver creates the following layout object(s): `DotColumn` (page 394), and `VaticanaLigature` (page 513).

`Vaticana_ligature_engraver` is part of the following context(s) in `\layout`: `VaticanaVoice` (page 261).

2.2.148 Vertical_align_engraver

Catch groups (staves, lyrics lines, etc.) and stack them vertically.

Properties (read)

`alignAboveContext` (string)
: Where to insert newly created context in vertical alignment.

`alignBelowContext` (string)
: Where to insert newly created context in vertical alignment.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

This engraver creates the following layout object(s): `VerticalAlignment` (page 514).

`Vertical_align_engraver` is part of the following context(s) in `\layout`: `ChoirStaff` (page 61), `GrandStaff` (page 99), `PianoStaff` (page 192), `Score` (page 197), and `StaffGroup` (page 229).

2.2.149 Volta_engraver

Make volta brackets.

Music types accepted: `volta-span-event` (page 55),

Properties (read)

`repeatCommands` (list)
: This property is a list of commands of the form `(list 'volta` *x*`)`, where *x* is a string or `#f`. `'end-repeat` is also accepted as a command.

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`voltaSpannerDuration` (moment)
: This specifies the maximum duration to use for the brackets printed for `\alternative`. This can be used to shrink the length of brackets in the situation where one alternative is very large.

This engraver creates the following layout object(s): `VoltaBracket` (page 517), and `VoltaBracketSpanner` (page 518).

`Volta_engraver` is part of the following context(s) in `\layout`: `Score` (page 197).

2.3 Tunable context properties

`accidentalGrouping` (symbol)
: If set to `'voice`, accidentals on the same note in different octaves may be horizontally staggered if in different voices.

`additionalBassStrings` (list)
: The additional tablature bass-strings, which will not get a seprate line in TabStaff. It is a list of the pitches of each string (starting with the lowest numbered one).

`additionalPitchPrefix` (string)
: Text with which to prefix additional pitches within a chord name.

`aDueText` (markup)
: Text to print at a unisono passage.

`alignAboveContext` (string)
: Where to insert newly created context in vertical alignment.

`alignBelowContext` (string)
: Where to insert newly created context in vertical alignment.

`alterationGlyphs` (list)
: Alist mapping alterations to accidental glyphs. Alterations are given as exact numbers, e.g., -1/2 for flat. This applies to all grobs that can print accidentals.

`alternativeNumber` (integer)
: When set, the index of the current `\alternative` element, starting from one. Not set outside of alternatives. Note the distinction from volta number: an alternative may pertain to multiple volte.

`alternativeNumberingStyle` (symbol)
: The scheme and style for numbering bars in repeat alternatives. If not set (the default), bar numbers continue through alternatives. Can be set to `numbers` to reset the bar number at each alternative, or set to `numbers-with-letters` to reset and also include letter suffixes.

`alternativeRestores` (symbol list)
: Timing variables that are restored to their value at the start of the first alternative in subsequent alternatives.

`associatedVoice` (string)
: Name of the context (see `associatedVoiceType` for its type, usually `Voice`) that has the melody for this `Lyrics` line.

`associatedVoiceType` (symbol)
: Type of the context that has the melody for this `Lyrics` line.

`autoAccidentals` (list)
: List of different ways to typeset an accidental.

  For determining when to print an accidental, several different rules are tried. The rule that gives the highest number of accidentals is used.

  Each entry in the list is either a symbol or a procedure.

  symbol The symbol is the name of the context in which the following rules are to be applied. For example, if *context* is Section "Score" in *Internals Reference* then all staves share accidentals, and if *context* is Section "Staff" in *Internals Reference* then all voices in the same staff share accidentals, but staves do not.

  procedure The procedure represents an accidental rule to be applied to the previously specified context.

  The procedure takes the following arguments:

  `context` The current context to which the rule should be applied.

  `pitch` The pitch of the note to be evaluated.

  `barnum` The current bar number.

The procedure returns a pair of booleans. The first states whether an extra natural should be added. The second states whether an accidental should be printed. `(#t . #f)` does not make sense.

`autoBeamCheck` (procedure)
: A procedure taking three arguments, *context*, *dir* [start/stop (-1 or 1)], and *test* [shortest note in the beam]. A non-`#f` return value starts or stops the auto beam.

`autoBeaming` (boolean)
: If set to true then beams are generated automatically.

`autoCautionaries` (list)
: List similar to `autoAccidentals`, but it controls cautionary accidentals rather than normal ones. Both lists are tried, and the one giving the most accidentals wins. In case of draw, a normal accidental is typeset.

`automaticBars` (boolean)
: If set to false then bar lines will not be printed automatically; they must be explicitly created with a `\bar` command. Unlike the `\cadenzaOn` keyword, measures are still counted. Bar line generation will resume according to that count if this property is unset.

`barAlways` (boolean)
: If set to true a bar line is drawn after each note.

`barCheckSynchronize` (boolean)
: If true then reset `measurePosition` when finding a bar check.

`barExtraVelocity` (integer)
: Extra MIDI velocity added by the '`Beat_performer`' at the start of each measure.

`barNumberFormatter` (procedure)
: A procedure that takes a bar number, measure position, and alternative number and returns a markup of the bar number to print.

`barNumberVisibility` (procedure)
: A procedure that takes a bar number and a measure position and returns whether the corresponding bar number should be printed. Note that the actual print-out of bar numbers is controlled with the `break-visibility` property.

  The following procedures are predefined:

  `all-bar-numbers-visible`
  : Enable bar numbers for all bars, including the first one and broken bars (which get bar numbers in parentheses).

  `first-bar-number-invisible`
  : Enable bar numbers for all bars (including broken bars) except the first one. If the first bar is broken, it doesn't get a bar number either.

  `first-bar-number-invisible-save-broken-bars`
  : Enable bar numbers for all bars (including broken bars) except the first one. A broken first bar gets a bar number.

  `first-bar-number-invisible-and-no-parenthesized-bar-numbers`
  : Enable bar numbers for all bars except the first bar and broken bars. This is the default.

  `(every-nth-bar-number-visible` *n*`)`
  : Assuming *n* is value 2, for example, this enables bar numbers for bars 2, 4, 6, etc.

`(modulo-bar-number-visible` *n m*`)`
: If bar numbers 1, 4, 7, etc., should be enabled, *n* (the modulo) must be set to 3 and *m* (the division remainder) to 1.

`baseMoment` (moment)
: Smallest unit of time that will stand on its own as a subdivided section.

`beamExceptions` (list)
: An alist of exceptions to autobeam rules that normally end on beats.

`beamHalfMeasure` (boolean)
: Whether to allow a beam to begin halfway through the measure in triple time, which could look like 6/8.

`beatExtraVelocity` (integer)
: Extra MIDI velocity added by the '`Beat_performer`' at the start of each beat.

`beatStructure` (list)
: List of `baseMoment`s that are combined to make beats.

`centerBarNumbers` (boolean)
: Whether to center bar numbers in their measure instead of aligning them on the bar line.

`chordChanges` (boolean)
: Only show changes in chords scheme?

`chordNameExceptions` (list)
: An alist of chord exceptions. Contains `(`*`chord . markup`*`)` entries.

`chordNameFunction` (procedure)
: The function that converts lists of pitches to chord names.

`chordNameLowercaseMinor` (boolean)
: Downcase roots of minor chords?

`chordNameSeparator` (markup)
: The markup object used to separate parts of a chord name.

`chordNoteNamer` (procedure)
: A function that converts from a pitch object to a text markup. Used for single pitches.

`chordPrefixSpacer` (number)
: The space added between the root symbol and the prefix of a chord name.

`chordRootNamer` (procedure)
: A function that converts from a pitch object to a text markup. Used for chords.

`clefGlyph` (string)
: Name of the symbol within the music font.

`clefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`clefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`clefTranspositionFormatter` (procedure)
: A procedure that takes the Transposition number as a string and the style as a symbol and returns a markup.

`clefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`completionBusy` (boolean)
: Whether a completion-note head is playing.

`completionFactor` (an exact rational or procedure)
: When `Completion_heads_engraver` and `Completion_rest_engraver` need to split a note or rest with a scaled duration, such as `c2*3`, this specifies the scale factor to use for the newly-split notes and rests created by the engraver.

  If `#f`, the completion engraver uses the scale-factor of each duration being split.

  If set to a callback procedure, that procedure is called with the context of the completion engraver, and the duration to be split.

`completionUnit` (moment)
: Sub-bar unit of completion.

`connectArpeggios` (boolean)
: If set, connect arpeggios across piano staff.

`countPercentRepeats` (boolean)
: If set, produce counters for percent repeats.

`createKeyOnClefChange` (boolean)
: Print a key signature whenever the clef is changed.

`createSpacing` (boolean)
: Create `StaffSpacing` objects? Should be set for staves.

`crescendoSpanner` (symbol)
: The type of spanner to be used for crescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin crescendo is used.

`crescendoText` (markup)
: The text to print at start of non-hairpin crescendo, i.e., '`cresc.`'.

`cueClefGlyph` (string)
: Name of the symbol within the music font.

`cueClefPosition` (number)
: Where should the center of the clef symbol go, measured in half staff spaces from the center of the staff.

`cueClefTransposition` (integer)
: Add this much extra transposition. Values of 7 and -7 are common.

`cueClefTranspositionFormatter` (procedure)
: A procedure that takes the Transposition number as a string and the style as a symbol and returns a markup.

`cueClefTranspositionStyle` (symbol)
: Determines the way the ClefModifier grob is displayed. Possible values are '`default`', '`parenthesized`' and '`bracketed`'.

`currentBarNumber` (integer)
: Contains the current barnumber. This property is incremented at every bar line.

`decrescendoSpanner` (symbol)
: The type of spanner to be used for decrescendi. Available values are '`hairpin`' and '`text`'. If unset, a hairpin decrescendo is used.

`decrescendoText` (markup)
: The text to print at start of non-hairpin decrescendo, i.e., '`dim.`'.

`defaultBarType` (string)
: Set the default type of bar line. See `whichBar` for information on available bar types.

  This variable is read by Section "Timing_translator" in *Internals Reference* at Section "Score" in *Internals Reference* level.

`defaultStrings` (list)
: A list of strings to use in calculating frets for tablatures and fretboards if no strings are provided in the notes for the current moment.

`doubleRepeatSegnoType` (string)
: Set the default bar line for the combinations double repeat with segno. Default is '`:|.S.|:`'.

`doubleRepeatType` (string)
: Set the default bar line for double repeats.

`doubleSlurs` (boolean)
: If set, two slurs are created for every slurred note, one above and one below the chord.

`drumPitchTable` (hash table)
: A table mapping percussion instruments (symbols) to pitches.

`drumStyleTable` (hash table)
: A hash table which maps drums to layout settings. Predefined values: '`drums-style`', '`agostini-drums-style`', '`timbales-style`', '`congas-style`', '`bongos-style`', and '`percussion-style`'.

  The layout style is a hash table, containing the drum-pitches (e.g., the symbol '`hihat`') as keys, and a list `(`*`notehead-style script vertical-position`*`)` as values.

`endAtSkip` (boolean)
: End `DurationLine` grob on `skip-event`

`endRepeatSegnoType` (string)
: Set the default bar line for the combinations ending of repeat with segno. Default is '`:|.S`'.

`endRepeatType` (string)
: Set the default bar line for the ending of repeats.

`explicitClefVisibility` (vector)
: '`break-visibility`' function for clef changes.

`explicitCueClefVisibility` (vector)
: '`break-visibility`' function for cue clef changes.

`explicitKeySignatureVisibility` (vector)
: '`break-visibility`' function for explicit key changes. '`\override`' of the `break-visibility` property will set the visibility for normal (i.e., at the start of the line) key signatures.

`extendersOverRests` (boolean)
: Whether to continue extenders as they cross a rest.

`extraNatural` (boolean)
: Whether to typeset an extra natural sign before accidentals that reduce the effect of a previous alteration.

`figuredBassAlterationDirection` (direction)
: Where to put alterations relative to the main figure.

`figuredBassCenterContinuations` (boolean)
: Whether to vertically center pairs of extender lines. This does not work with three or more lines.

`figuredBassFormatter` (procedure)
: A routine generating a markup for a bass figure.

`figuredBassPlusDirection` (direction)
: Where to put plus signs relative to the main figure.

`fineBarType` (string)
: The bar line for `\fine`. See `whichBar` for information on available bar types.

`fineSegnoType` (string)
: Set the default bar line for a requested segno with fine. Default is '`|.S`'.

`fineStartRepeatSegnoType` (string)
: Set the default bar line for the combinations beginning of repeat with segno and fine. Default is '`|.S.|:`'.

`fineText` (markup)
: The text to print at `\fine`.

`fingeringOrientations` (list)
: A list of symbols, containing '`left`', '`right`', '`up`' and/or '`down`'. This list determines where fingerings are put relative to the chord being fingered.

`firstClef` (boolean)
: If true, create a new clef when starting a staff.

`followVoice` (boolean)
: If set, note heads are tracked across staff switches by a thin line.

`fontSize` (number)
: The relative size of all grobs in a context.

`forbidBreak` (boolean)
: If set to `#t`, prevent a line break at this point.

`forceClef` (boolean)
: Show clef symbol, even if it has not changed. Only active for the first clef after the property is set, not for the full staff.

`fretLabels` (list)
: A list of strings or Scheme-formatted markups containing, in the correct order, the labels to be used for lettered frets in tablature.

`glissandoMap` (list)
: A map in the form of '((source1 . target1) (source2 . target2) (sourcen . targetn)) showing the glissandi to be drawn for note columns. The value '() will default to '((0 . 0) (1 . 1) (n . n)), where n is the minimal number of note-heads in the two note columns between which the glissandi occur.

`gridInterval` (moment)
: Interval for which to generate `GridPoint`s.

`handleNegativeFrets` (symbol)
: How the automatic fret calculator should handle calculated negative frets. Values include `'ignore`, to leave them out of the diagram completely, `'include`, to include

them as calculated, and `'recalculate`, to ignore the specified string and find a string where they will fit with a positive fret number.

`harmonicAccidentals` (boolean)
: If set, harmonic notes in chords get accidentals.

`harmonicDots` (boolean)
: If set, harmonic notes in dotted chords get dots.

`highStringOne` (boolean)
: Whether the first string is the string with highest pitch on the instrument. This used by the automatic string selector for tablature notation.

`ignoreBarChecks` (boolean)
: Ignore bar checks.

`ignoreBarNumberChecks` (boolean)
: Ignore bar number checks.

`ignoreFiguredBassRest` (boolean)
: Don't swallow rest events.

`ignoreMelismata` (boolean)
: Ignore melismata for this Section "Lyrics" in *Internals Reference* line.

`implicitBassFigures` (list)
: A list of bass figures that are not printed as numbers, but only as extender lines.

`includeGraceNotes` (boolean)
: Do not ignore grace notes for Section "Lyrics" in *Internals Reference*.

`initialTimeSignatureVisibility` (vector)
: break visibility for the initial time signature.

`instrumentCueName` (markup)
: The name to print if another instrument is to be taken.

`instrumentEqualizer` (procedure)
: A function taking a string (instrument name), and returning a `(min . max)` pair of numbers for the loudness range of the instrument.

`instrumentName` (markup)
: The name to print left of a staff. The `instrumentName` property labels the staff in the first system, and the `shortInstrumentName` property labels following lines.

`instrumentTransposition` (pitch)
: Define the transposition of the instrument. Its value is the pitch that sounds when the instrument plays written middle C. This is used to transpose the MIDI output, and `\quote`s.

`internalBarNumber` (integer)
: Contains the current barnumber. This property is used for internal timekeeping, among others by the `Accidental_engraver`.

`keepAliveInterfaces` (list)
: A list of symbols, signifying grob interfaces that are worth keeping a staff with `remove-empty` set around for.

`keyAlterationOrder` (list)
: A list of pairs that defines in what order alterations should be printed. The format of an entry is `(step . alter)`, where *step* is a number from 0 to 6 and *alter* from -1 (double flat) to 1 (double sharp), with exact rationals for alterations in between, e.g., 1/2 for sharp.

`keyAlterations` (list)
: The current key signature. This is an alist containing `(step . alter)` or `((octave . step) . alter)`, where *step* is a number in the range 0 to 6 and *alter* a fraction, denoting alteration. For alterations, use symbols, e.g. `keyAlterations = #`((6 . ,FLAT))`.

`lyricMelismaAlignment` (number)
: Alignment to use for a melisma syllable.

`magnifyStaffValue` (positive number)
: The most recent value set with `\magnifyStaff`.

`majorSevenSymbol` (markup)
: How should the major 7th be formatted in a chord name?

`markFormatter` (procedure)
: A procedure taking as arguments the context and the sequence number of the rehearsal mark. It should return the formatted mark as a markup object.

`maximumFretStretch` (number)
: Don't allocate frets further than this from specified frets.

`measureLength` (moment)
: Length of one measure in the current time signature.

`measurePosition` (moment)
: How much of the current measure have we had. This can be set manually to create incomplete measures.

`measureStartNow` (boolean)
: True at the beginning of a measure.

`melismaBusyProperties` (list)
: A list of properties (symbols) to determine whether a melisma is playing. Setting this property will influence how lyrics are aligned to notes. For example, if set to `'(melismaBusy beamMelismaBusy)`, only manual melismata and manual beams are considered. Possible values include `melismaBusy`, `slurMelismaBusy`, `tieMelismaBusy`, and `beamMelismaBusy`.

`metronomeMarkFormatter` (procedure)
: How to produce a metronome markup. Called with two arguments: a `TempoChangeEvent` and context.

`middleCClefPosition` (number)
: The position of the middle C, as determined only by the clef. This can be calculated by looking at `clefPosition` and `clefGlyph`.

`middleCCuePosition` (number)
: The position of the middle C, as determined only by the clef of the cue notes. This can be calculated by looking at `cueClefPosition` and `cueClefGlyph`.

`middleCOffset` (number)
: The offset of middle C from the position given by `middleCClefPosition` This is used for ottava brackets.

`middleCPosition` (number)
: The place of the middle C, measured in half staff-spaces. Usually determined by looking at `middleCClefPosition` and `middleCOffset`.

`midiBalance` (number)
: Stereo balance for the MIDI channel associated with the current context. Ranges from -1 to 1, where the values -1 (`#LEFT`), 0 (`#CENTER`) and 1 (`#RIGHT`) correspond to leftmost emphasis, center balance, and rightmost emphasis, respectively.

`midiChannelMapping` (symbol)
: How to map MIDI channels: per `staff` (default), `instrument` or `voice`.

`midiChorusLevel` (number)
: Chorus effect level for the MIDI channel associated with the current context. Ranges from 0 to 1 (0=off, 1=full effect).

`midiExpression` (number)
: Expression control for the MIDI channel associated with the current context. Ranges from 0 to 1 (0=off, 1=full effect).

`midiInstrument` (string)
: Name of the MIDI instrument to use.

`midiMaximumVolume` (number)
: Analogous to `midiMinimumVolume`.

`midiMergeUnisons` (boolean)
: If true, output only one MIDI note-on event when notes with the same pitch, in the same MIDI-file track, overlap.

`midiMinimumVolume` (number)
: Set the minimum loudness for MIDI. Ranges from 0 to 1.

`midiPanPosition` (number)
: Pan position for the MIDI channel associated with the current context. Ranges from -1 to 1, where the values -1 (`#LEFT`), 0 (`#CENTER`) and 1 (`#RIGHT`) correspond to hard left, center, and hard right, respectively.

`midiReverbLevel` (number)
: Reverb effect level for the MIDI channel associated with the current context. Ranges from 0 to 1 (0=off, 1=full effect).

`minimumFret` (number)
: The tablature auto string-selecting mechanism selects the highest string with a fret at least `minimumFret`.

`minimumPageTurnLength` (moment)
: Minimum length of a rest for a page turn to be allowed.

`minimumRepeatLengthForPageTurn` (moment)
: Minimum length of a repeated section for a page turn to be allowed within that section.

`minorChordModifier` (markup)
: Markup displayed following the root for a minor chord

`noChordSymbol` (markup)
: Markup to be displayed for rests in a ChordNames context.

`noteNameFunction` (procedure)
: Function used to convert pitches into strings and markups.

`noteNameSeparator` (string)
: String used to separate simultaneous NoteName objects.

`noteToFretFunction` (procedure)
: Convert list of notes and list of defined strings to full list of strings and fret numbers. Parameters: The context, a list of note events, a list of tabstring events, and the fretboard grob if a fretboard is desired.

`nullAccidentals` (boolean)
: The `Accidental_engraver` generates no accidentals for notes in contexts were this is set. In addition to supressing the printed accidental, this option removes any effect the note would have had on accidentals in other voices.

`ottavaStartNow` (boolean)
: Is an ottava starting in this time step?

`ottavation` (markup)
: If set, the text for an ottava spanner. Changing this creates a new text spanner.

`ottavationMarkups` (list)
: An alist defining the markups used for ottava brackets. It contains entries of the form `(`*`number of octaves . markup`*`)`.

`output` (music output)
: The output produced by a score-level translator during music interpretation.

`partCombineForced` (symbol)
: Override for the partCombine decision. Can be `apart`, `chords`, `unisono`, `solo1`, or `solo2`.

`partCombineTextsOnNote` (boolean)
: Print part-combine texts only on the next note rather than immediately on rests or skips.

`pedalSostenutoStrings` (list)
: See `pedalSustainStrings`.

`pedalSostenutoStyle` (symbol)
: See `pedalSustainStyle`.

`pedalSustainStrings` (list)
: A list of strings to print for sustain-pedal. Format is `(`*`up updown down`*`)`, where each of the three is the string to print when this is done with the pedal.

`pedalSustainStyle` (symbol)
: A symbol that indicates how to print sustain pedals: `text`, `bracket` or `mixed` (both).

`pedalUnaCordaStrings` (list)
: See `pedalSustainStrings`.

`pedalUnaCordaStyle` (symbol)
: See `pedalSustainStyle`.

`predefinedDiagramTable` (hash table)
: The hash table of predefined fret diagrams to use in FretBoards.

`printAccidentalNames` (boolean or symbol)
: Print accidentals in the `NoteNames` context.

`printKeyCancellation` (boolean)
: Print restoration alterations before a key signature change.

`printNotesLanguage` (string)
: Use a specific language in the `NoteNames` context.

`printOctaveNames` (boolean or symbol)
: Print octave marks in the `NoteNames` context.

`printPartCombineTexts` (boolean)
: Set 'Solo' and 'A due' texts in the part combiner?

`proportionalNotationDuration` (moment)
: Global override for shortest-playing duration. This is used for switching on proportional notation.

`rehearsalMark` (integer)
: The last rehearsal mark printed.

`repeatCommands` (list)
: This property is a list of commands of the form `(list 'volta` *x*`)`, where *x* is a string or `#f`. `'end-repeat` is also accepted as a command.

`repeatCountVisibility` (procedure)
: A procedure taking as arguments an integer and context, returning whether the corresponding percent repeat number should be printed when `countPercentRepeats` is set.

`restCompletionBusy` (boolean)
: Signal whether a completion-rest is active.

`restNumberThreshold` (number)
: If a multimeasure rest has more measures than this, a number is printed.

`restrainOpenStrings` (boolean)
: Exclude open strings from the automatic fret calculator.

`searchForVoice` (boolean)
: Signal whether a search should be made of all contexts in the context hierarchy for a voice to provide rhythms for the lyrics.

`sectionBarType` (string)
: The bar line for `\section`. See `whichBar` for information on available bar types.

`segnoType` (string)
: Set the default bar line for a requested segno. Default is '`S`'.

`shapeNoteStyles` (vector)
: Vector of symbols, listing style for each note head relative to the tonic (qv.) of the scale.

`shortInstrumentName` (markup)
: See `instrumentName`.

`shortVocalName` (markup)
: Name of a vocal line, short version.

`skipBars` (boolean)
: If set to true, then skip the empty bars that are produced by multimeasure notes and rests. These bars will not appear on the printed output. If not set (the default), multimeasure notes and rests expand into their full length, printing the appropriate number of empty bars so that synchronization with other voices is preserved.

```
{
  r1 r1*3 R1*3
  \set Score.skipBars= ##t
  r1*3 R1*3
}
```

`skipTypesetting` (boolean)
: If true, no typesetting is done, speeding up the interpretation phase. Useful for debugging large scores.

`slashChordSeparator` (markup)
: The markup object used to separate a chord name from its root note in case of inversions or slash chords.

`soloIIText` (markup)
: The text for the start of a solo for voice 'two' when part-combining.

`soloText` (markup)
: The text for the start of a solo when part-combining.

`squashedPosition` (integer)
: Vertical position of squashing for Section "Pitch_squash_engraver" in *Internals Reference*.

`staffLineLayoutFunction` (procedure)
: Layout of staff lines, `traditional`, or `semitone`.

`stanza` (markup)
: Stanza 'number' to print before the start of a verse. Use in `Lyrics` context.

`startAtNoteColumn` (boolean)
: Start `DurationLine` grob at entire `NoteColumn`.

`startAtSkip` (boolean)
: Start `DurationLine` grob at `skip-event`.

`startRepeatSegnoType` (string)
: Set the default bar line for the combinations beginning of repeat with segno. Default is '`S.|:`'.

`startRepeatType` (string)
: Set the default bar line for the beginning of repeats.

`stemLeftBeamCount` (integer)
: Specify the number of beams to draw on the left side of the next note. Overrides automatic beaming. The value is only used once, and then it is erased.

`stemRightBeamCount` (integer)
: See `stemLeftBeamCount`.

`strictBeatBeaming` (boolean)
: Should partial beams reflect the beat structure even if it causes flags to hang out?

`stringNumberOrientations` (list)
: See `fingeringOrientations`.

`stringOneTopmost` (boolean)
: Whether the first string is printed on the top line of the tablature.

`stringTunings` (list)
: The tablature strings tuning. It is a list of the pitches of each string (starting with the lowest numbered one).

`strokeFingerOrientations` (list)
: See `fingeringOrientations`.

`subdivideBeams` (boolean)
: If set, multiple beams will be subdivided at `baseMoment` positions by only drawing one beam over the beat.

`suggestAccidentals` (boolean or symbol)
: If set to `#t`, accidentals are typeset as suggestions above the note. Setting it to `'cautionary` only applies that to cautionary accidentals.

`supportNonIntegerFret` (boolean)
: If set in `Score` the `TabStaff` will print micro-tones as '$2\frac{1}{2}$'

`suspendMelodyDecisions` (boolean)
: When using the `Melody_engraver`, stop changing orientation of stems based on the melody when this is set to true.

`suspendRestMerging` (boolean)
: When using the Merge_rest_engraver do not merge rests when this is set to true.

`systemStartDelimiter` (symbol)
: Which grob to make for the start of the system/staff? Set to `SystemStartBrace`, `SystemStartBracket` or `SystemStartBar`.

`systemStartDelimiterHierarchy` (pair)
: A nested list, indicating the nesting of a start delimiters.

`tablatureFormat` (procedure)
: A function formatting a tablature note head. Called with three arguments: context, string number and, fret number. It returns the text as a markup.

`tabStaffLineLayoutFunction` (procedure)
: A function determining the staff position of a tablature note head. Called with two arguments: the context and the string.

`tempoHideNote` (boolean)
: Hide the note = count in tempo marks.

`tempoWholesPerMinute` (moment)
: The tempo in whole notes per minute.

`tieWaitForNote` (boolean)
: If true, tied notes do not have to follow each other directly. This can be used for writing out arpeggios.

`timeSignatureFraction` (fraction, as pair)
: A pair of numbers, signifying the time signature. For example, `'(4 . 4)` is a 4/4 time signature.

`timeSignatureSettings` (list)
: A nested alist of settings for time signatures. Contains elements for various time signatures. The element for each time signature contains entries for `baseMoment`, `beatStructure`, and `beamExceptions`.

`timing` (boolean)
: Keep administration of measure length, position, bar number, etc.? Switch off for cadenzas.

`tonic` (pitch)
: The tonic of the current scale.

`topLevelAlignment` (boolean)
: If true, the *Vertical_align_engraver* will create a *VerticalAlignment*; otherwise, it will create a *StaffGrouper*

`tupletFullLength` (boolean)
: If set, the tuplet is printed up to the start of the next note.

`tupletFullLengthNote` (boolean)
: If set, end at the next note, otherwise end on the matter (time signatures, etc.) before the note.

`tupletSpannerDuration` (moment)
: Normally, a tuplet bracket is as wide as the `\times` expression that gave rise to it. By setting this property, you can make brackets last shorter.

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

`underlyingRepeatType` (string)
: Set the bar line to use at points of repetition or departure where no bar line would normally appear, for example at the end of a system broken in mid measure where the next system begins with a segno.

`useBassFigureExtenders` (boolean)
: Whether to use extender lines for repeated bass figures.

`vocalName` (markup)
: Name of a vocal line.

`voltaSpannerDuration` (moment)
: This specifies the maximum duration to use for the brackets printed for `\alternative`. This can be used to shrink the length of brackets in the situation where one alternative is very large.

`whichBar` (string)
: This property is read to determine what type of bar line to create.

  Example:

```
\set Staff.whichBar = ".|:"
```

  This will create a start-repeat bar in this staff only. Valid values are described in `scm/bar-line.scm`.

2.4 Internal context properties

`associatedVoiceContext` (context)
: The context object of the `Voice` that has the melody for this `Lyrics`.

`barCheckLastFail` (moment)
: Where in the measure did the last barcheck fail?

`beamMelismaBusy` (boolean)
: Signal if a beam is present.

`busyGrobs` (list)
: A queue of `(`*`end-moment . grob`*`)` cons cells. This is for internal (C++) use only. This property contains the grobs which are still busy (e.g. note heads, spanners, etc.).

`currentCommandColumn` (graphical (layout) object)
: Grob that is X-parent to all current breakable (clef, key signature, etc.) items.

`currentMarkEvent` (stream event)
: The event selected by `Mark_tracking_translator` for engraving by `Mark_engraver`.

`currentMusicalColumn` (graphical (layout) object)
: Grob that is X-parent to all non-breakable items (note heads, lyrics, etc.).

`dynamicAbsoluteVolumeFunction` (procedure)
: A procedure that takes one argument, the text value of a dynamic event, and returns the absolute volume of that dynamic event.

`finalizations` (list)
: A list of expressions to evaluate before proceeding to next time step. This is an internal variable.

`graceSettings` (list)
: Overrides for grace notes. This property should be manipulated through the `add-grace-property` function.

`hasAxisGroup` (boolean)
: True if the current context is contained in an axis group.

`hasStaffSpacing` (boolean)
: True if the current `CommandColumn` contains items that will affect spacing.

`lastChord` (markup)
: Last chord, used for detecting chord changes.

`lastKeyAlterations` (list)
: Last key signature before a key signature change.

`localAlterations` (list)
: The key signature at this point in the measure. The format is the same as for `keyAlterations`, but can also contain `((`*`octave . name`*`) . (`*`alter barnumber . measureposition`*`))` pairs.

`melismaBusy` (boolean)
: Signifies whether a melisma is active. This can be used to signal melismas on top of those automatically detected.

`midiSkipOffset` (moment)
: This is the accrued MIDI offset to account for time skipped via `skipTypesetting`.

`partialBusy` (boolean)
: Signal that \partial acts at the current timestep.

`quotedCueEventTypes` (list)
: A list of symbols, representing the event types that should be duplicated for `\cueDuring` commands.

`quotedEventTypes` (list)
: A list of symbols, representing the event types that should be duplicated for `\quoteDuring` commands. This is also a fallback for `\cueDuring` if `quotedCueEventTypes` is not set

`rootSystem` (graphical (layout) object)
: The System object.

`scriptDefinitions` (list)
: The description of scripts. This is used by the `Script_engraver` for typesetting note-superscripts and subscripts. See `scm/script.scm` for more information.

`slurMelismaBusy` (boolean)
: Signal if a slur is present.

`stavesFound` (list of grobs)
: A list of all staff-symbols found.

`stringFretFingerList` (list)
: A list containg three entries. In `TabVoice` and `FretBoards` they determine the string, fret and finger to use

`tieMelismaBusy` (boolean)
: Signal whether a tie is present.

3 Backend

3.1 All layout objects

3.1.1 Accidental

`Accidental` objects are created by: `Accidental_engraver` (page 282).

Standard settings:

`after-line-breaking` (boolean):
`ly:accidental-interface::remove-tied`

Dummy property, used to trigger callback for `after-line-breaking`.

`alteration` (number):
`accidental-interface::calc-alteration`

Alteration numbers for accidental.

`avoid-slur` (symbol):
`'inside`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`extra-spacing-width` (pair of numbers):
`'(-0.2 . 0.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`horizontal-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:accidental-interface::horizontal-skylines> >`

Two skylines, one to the left and one to the right of this grob.

`stencil` (stencil):
`ly:accidental-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`ly:grob::x-parent-positioning`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:accidental-interface::height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-interface` (page 521), `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), and `inline-accidental-interface` (page 555).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.2 AccidentalCautionary

`AccidentalCautionary` objects are created by: `Accidental_engraver` (page 282).

Standard settings:

`after-line-breaking` (boolean):
`ly:accidental-interface::remove-tied`
Dummy property, used to trigger callback for `after-line-breaking`.

`alteration` (number):
`accidental-interface::calc-alteration`
Alteration numbers for accidental.

`avoid-slur` (symbol):
`'inside`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`parenthesized` (boolean):
`#t`
Parenthesize this grob.

`stencil` (stencil):
`ly:accidental-interface::print`
The symbol to print.

`X-offset` (number):
`ly:grob::x-parent-positioning`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:accidental-interface::height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-interface` (page 521), `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), and `inline-accidental-interface` (page 555).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.3 AccidentalPlacement

AccidentalPlacement objects are created by: Accidental_engraver (page 282), and Ambitus_engraver (page 283).

Standard settings:

direction (direction):
-1

If side-axis is 0 (or X), then this property determines whether the object is placed LEFT, CENTER or RIGHT with respect to the other object. Otherwise, it determines whether the object is placed UP, CENTER or DOWN. Numerical values may also be used: UP=1, DOWN=-1, LEFT=-1, RIGHT=1, CENTER=0.

right-padding (dimension, in staff space):
0.15

Space to insert on the right side of an object (e.g., between note and its accidentals).

script-priority (number):
-100

A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

X-extent (pair of numbers):
ly:axis-group-interface::width

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): accidental-placement-interface (page 521), and grob-interface (page 548).

This object is of class item-interface,Item (page ⟨undefined⟩).

3.1.4 AccidentalSuggestion

AccidentalSuggestion objects are created by: Accidental_engraver (page 282).

Standard settings:

after-line-breaking (boolean):
ly:accidental-interface::remove-tied

Dummy property, used to trigger callback for after-line-breaking.

alteration (number):
accidental-interface::calc-alteration

Alteration numbers for accidental.

direction (direction):
1

If side-axis is 0 (or X), then this property determines whether the object is placed LEFT, CENTER or RIGHT with respect to the other object. Otherwise, it determines whether the object is placed UP, CENTER or DOWN. Numerical values may also be used: UP=1, DOWN=-1, LEFT=-1, RIGHT=1, CENTER=0.

font-size (number):
-2

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`outside-staff-priority` (number):
`0`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`script-priority` (number):
`0`

A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`0.25`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:accidental-interface::print`

The symbol to print.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:accidental-interface::height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):

`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-interface` (page 521), `accidental-suggestion-interface` (page 522), `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `script-interface` (page 578), `self-alignment-interface` (page 579), and `side-position-interface` (page 581).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.5 Ambitus

`Ambitus` objects are created by: `Ambitus_engraver` (page 283).

Standard settings:

`axes` (list):

`'(0 1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`break-align-symbol` (symbol):

`'ambitus`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):

`#(#f #f #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`non-musical` (boolean):

`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):

```
'((cue-end-clef extra-space . 0.5)
  (clef extra-space . 1.15)
  (cue-clef extra-space . 0.5)
  (key-signature extra-space . 1.15)
  (staff-bar extra-space . 1.15)
  (time-signature extra-space . 1.15)
  (right-edge extra-space . 0.5)
  (first-note extra-space . 1.15))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `ambitus-interface` (page 523), `axis-group-interface` (page 524), `break-aligned-interface` (page 533), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.6 AmbitusAccidental

`AmbitusAccidental` objects are created by: `Ambitus_engraver` (page 283).

Standard settings:

`stencil` (stencil):
`ly:accidental-interface::print`
The symbol to print.

`X-offset` (number):
`ly:grob::x-parent-positioning`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:accidental-interface::height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-interface` (page 521), `accidental-switch-interface` (page 522), `break-aligned-interface` (page 533), `font-interface` (page 543), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.7 AmbitusLine

`AmbitusLine` objects are created by: `Ambitus_engraver` (page 283).

Standard settings:

`gap` (dimension, in staff space):
`ambitus-line::calc-gap`
Size of a gap in a variable symbol.

`length-fraction` (number):
`0.7`
Multiplier for lengths. Used for determining ledger lines and stem lengths.

`maximum-gap` (number):
`0.45`
Maximum value allowed for `gap` property.

`stencil` (stencil):
`ambitus::print`
The symbol to print.

`thickness` (number):
`2`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`X-offset` (number):
`ly:self-alignment-interface::centered-on-x-parent`
The horizontal amount that this object is moved relative to its X-parent.

This object supports the following interface(s): `ambitus-interface` (page 523), `font-interface` (page 543), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.8 AmbitusNoteHead

`AmbitusNoteHead` objects are created by: `Ambitus_engraver` (page 283).

Standard settings:

`duration-log` (integer):
`2`
The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`glyph-name` (string):
`note-head::calc-glyph-name`
The glyph name within the font.
In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`ignore-ambitus` (boolean):
`#t`
If set, don't consider this notehead for ambitus calculation.

`stencil` (stencil):
`ly:note-head::print`
The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `ambitus-interface` (page 523), `font-interface` (page 543), `grob-interface` (page 548), `ledgered-interface` (page 559), `note-head-interface` (page 569), `rhythmic-head-interface` (page 577), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.9 Arpeggio

`Arpeggio` objects are created by: `Arpeggio_engraver` (page 284), and `Span_arpeggio_engraver` (page 322).

Standard settings:

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`line-thickness` (number):
`1`
For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`padding` (dimension, in staff space):
`0.5`
Add this much extra space between objects that are next to each other.

`positions` (pair of numbers):
`ly:arpeggio::calc-positions`
Pair of staff coordinates `(`*`start . end`*`)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`protrusion` (number):
`0.4`
In an arpeggio bracket, the length of the horizontal edges.

`script-priority` (number):
`0`
A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`side-axis` (number):
`0`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-position` (number):
`0.0`
Vertical position, measured in half staff spaces, counted from the middle line.

`stencil` (stencil):
`ly:arpeggio::print`
The symbol to print.

`thickness` (number):
`1`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`X-extent` (pair of numbers):
`ly:arpeggio::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> #<primitive-procedure ly:arpeggio::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `arpeggio-interface` (page 523), `font-interface` (page 543), `grob-interface` (page 548), `side-position-interface` (page 581), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.10 BalloonText

`BalloonText` objects are created by: `Balloon_engraver` (page 285).

Standard settings:

`annotation-balloon` (boolean):
`#t`
Print the balloon around an annotation.

`annotation-line` (boolean):
`#t`
Print the line from an annotation to the grob that it annotates.

`break-visibility` (vector):
`#<procedure #f (grob)>`
A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`stencil` (stencil):
`ly:balloon-interface::print`
The symbol to print.

`text` (markup):
`#<procedure #f (grob)>`
Text markup. See Section "Formatting text" in *Notation Reference*.

`X-offset` (number):
`#<procedure #f (grob)>`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> #<procedure #f (grob start end)> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<procedure #f (grob)>`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `balloon-interface` (page 526), `font-interface` (page 543), `grob-interface` (page 548), `sticky-grob-interface` (page 593), and `text-interface` (page 596).

This object can be of either of the following classes: `item-interface,Item` (page ⟨undefined⟩), and `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.11 BarLine

`BarLine` objects are created by: `Bar_engraver` (page 285).

Standard settings:

`allow-span-bar` (boolean):
`#t`
If false, no inter-staff bar line will be created below this bar line.

`bar-extent` (pair of numbers):
`ly:bar-line::calc-bar-extent`
The Y-extent of the actual bar line. This may differ from `Y-extent` because it does not include the dots in a repeat bar line.

`break-align-anchor` (number):
`ly:bar-line::calc-anchor`
Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-symbol` (symbol):
`'staff-bar`
This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`bar-line::calc-break-visibility`
A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):
`pure-from-neighbor-interface::account-for-span-bar`
In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`gap` (dimension, in staff space):
`0.4`

Size of a gap in a variable symbol.

`glyph` (string):
`"|"`

A string determining what 'style' of glyph is typeset. Valid choices depend on the function that is reading this property.

In combination with (span) bar lines, it is a string resembling the bar line appearance in ASCII form.

`glyph-name` (string):
`bar-line::calc-glyph-name`

The glyph name within the font.

In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`hair-thickness` (number):
`1.9`

Thickness of the thin line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`kern` (dimension, in staff space):
`3.0`

The space between individual elements in any compound bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`layer` (integer):
`0`

An integer which determines the order of printing objects. 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.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`rounded` (boolean)
Decide whether lines should be drawn rounded or not.

`segno-kern` (number):
`3.0`

The space between the two thin lines of the segno bar line symbol, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`space-alist` (list):

```
'((ambitus extra-space . 1.0)
  (time-signature extra-space . 0.75)
  (custos minimum-space . 2.0)
  (clef extra-space . 1.0)
  (key-signature extra-space . 1.0)
```

```
(key-cancellation extra-space . 1.0)
(first-note fixed-space . 1.3)
(next-note semi-fixed-space . 0.9)
(right-edge extra-space . 0.0))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *break-align-symbol* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *spacing-style* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
`ly:bar-line::print`
The symbol to print.

`thick-thickness` (number):
`6.0`
Thickness of the thick line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `bar-line-interface` (page 527), `break-aligned-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), and `pure-from-neighbor-interface` (page 576).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.12 BarNumber

`BarNumber` objects are created by: `Bar_number_engraver` (page 286).

Standard settings:

`after-line-breaking` (boolean):
`ly:side-position-interface::move-to-extremal-staff`
Dummy property, used to trigger callback for `after-line-breaking`.

`break-align-symbols` (list):
`'(left-edge staff-bar)`
A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#f #f #t)`
A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-family` (symbol):
`'roman`

The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`font-size` (number):
`-2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`horizon-padding` (number):
`0.05`

The amount to pad the axis along which a `Skyline` is built for the `side-position-interface`.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`outside-staff-priority` (number):
`100`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`1.0`

Add this much extra space between objects that are next to each other.

`self-alignment-X` (number):
`#<procedure #f (grob)>`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`X-offset` (number):
`self-alignment-interface::self-aligned-on-breakable`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `bar-number-interface` (page 528), `break-alignable-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.13 BassFigure

`BassFigure` objects are created by: `Figured_bass_engraver` (page 299).

Standard settings:

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `bass-figure-interface` (page 528), `font-interface` (page 543), `grob-interface` (page 548), `rhythmic-grob-interface` (page 577), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.14 BassFigureAlignment

`BassFigureAlignment` objects are created by: `Figured_bass_engraver` (page 299).

Standard settings:

`axes` (list):
`'(1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`padding` (dimension, in staff space):
`0.2`

Add this much extra space between objects that are next to each other.

`stacking-dir` (direction):
`-1`

Stack objects in which direction?

`vertical-skylines` (pair of skylines):
`ly:axis-group-interface::calc-skylines`

Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `align-interface` (page 522), `axis-group-interface` (page 524), `bass-figure-alignment-interface` (page 528), and `grob-interface` (page 548).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.15 BassFigureAlignmentPositioning

`BassFigureAlignmentPositioning` objects are created by: `Figured_bass_position_engraver` (page 299).

Standard settings:

`add-stem-support` (boolean):
`#t`
If set, the `Stem` object is included in this script's support.

`axes` (list):
`'(1)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`padding` (dimension, in staff space):
`0.5`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`1.0`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

  Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
: `#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

  The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.16 BassFigureBracket

`BassFigureBracket` objects are created by: `Figured_bass_engraver` (page 299).

Standard settings:

`edge-height` (pair):
: `'(0.2 . 0.2)`

  A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`stencil` (stencil):
: `ly:enclosing-bracket::print`

  The symbol to print.

`X-extent` (pair of numbers):
: `ly:enclosing-bracket::width`

  Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `enclosing-bracket-interface` (page 540), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.17 BassFigureContinuation

`BassFigureContinuation` objects are created by: `Figured_bass_engraver` (page 299).

Standard settings:

`stencil` (stencil):
: `ly:figured-bass-continuation::print`

  The symbol to print.

`Y-offset` (number):
: `ly:figured-bass-continuation::center-on-figures`

  The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `figured-bass-continuation-interface` (page 541), and `grob-interface` (page 548).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.18 BassFigureLine

`BassFigureLine` objects are created by: `Figured_bass_engraver` (page 299).

Standard settings:

`axes` (list):
`'(1)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`vertical-skylines` (pair of skylines):
`ly:axis-group-interface::calc-skylines`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), and `outside-staff-axis-group-interface` (page 571).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.19 Beam

`Beam` objects are created by: `Auto_beam_engraver` (page 284), `Beam_engraver` (page 287), `Chord_tremolo_engraver` (page 291), `Grace_auto_beam_engraver` (page 302), and `Grace_beam_engraver` (page 302).

Standard settings:

`auto-knee-gap` (dimension, in staff space):
`5.5`
If a gap is found between note heads where a horizontal beam fits and it is larger than this number, make a kneed beam.

`beam-thickness` (dimension, in staff space):
`0.48`
Beam thickness, measured in `staff-space` units.

`beamed-stem-shorten` (list):
`'(1.0 0.5 0.25)`
How much to shorten beamed stems, when their direction is forced. It is a list, since the value is different depending on the number of flags and beams.

`beaming` (pair):
`ly:beam::calc-beaming`
Pair of number lists. Each number list specifies which beams to make. `0` is the central beam, `1` is the next beam toward the note, etc. This information is used to determine how to connect the beaming patterns from stem to stem inside a beam.

`clip-edges` (boolean):
`#t`
Allow outward pointing beamlets at the edges of beams?

`collision-interfaces` (list):
```
'(beam-interface
   clef-interface
   clef-modifier-interface
   flag-interface
   inline-accidental-interface
   key-signature-interface
   note-head-interface
   stem-interface
   time-signature-interface)
```
A list of interfaces for which automatic beam-collision resolution is run.

`damping` (number):
`1`
Amount of beam slope damping.

`details` (list):
```
'((secondary-beam-demerit . 10)
  (stem-length-demerit-factor . 5)
  (region-size . 2)
  (beam-eps . 0.001)
  (stem-length-limit-penalty . 5000)
  (damping-direction-penalty . 800)
  (hint-direction-penalty . 20)
  (musical-direction-factor . 400)
  (ideal-slope-factor . 10)
  (collision-penalty . 500)
  (collision-padding . 0.35)
  (round-to-zero-slope . 0.02))
```
Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`ly:beam::calc-direction`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-family` (symbol):
`'roman`
The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`gap` (dimension, in staff space):
`0.8`
Size of a gap in a variable symbol.

`neutral-direction` (direction):
`-1`

Which direction to take in the center of the staff.

`normalized-endpoints` (pair):
`ly:spanner::calc-normalized-endpoints`

Represents left and right placement over the total spanner, where the width of the spanner is normalized between 0 and 1.

`positions` (pair of numbers):
`beam::place-broken-parts-individually`

Pair of staff coordinates `(start . end)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`stencil` (stencil):
`ly:beam::print`

The symbol to print.

`transparent` (boolean):
`#<procedure #f (grob)>`

This makes the grob invisible.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-positions` (pair of numbers):
`ly:beam::calc-x-positions`

Pair of X staff coordinates of a spanner in the form `(left . right)`, where both *left* and *right* are in `staff-space` units of the current staff.

This object supports the following interface(s): `beam-interface` (page 528), `grob-interface` (page 548), `staff-symbol-referencer-interface` (page 590), and `unbreakable-spanner-interface` (page 603).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.20 BendAfter

`BendAfter` objects are created by: `Bend_engraver` (page 289).

Standard settings:

`minimum-length` (dimension, in staff space):
`0.5`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`stencil` (stencil):
`bend::print`

The symbol to print.

`thickness` (number):
`2.0`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

This object supports the following interface(s): bend-after-interface (page 531), and grob-interface (page 548).

This object is of class spanner-interface,Spanner (page ⟨undefined⟩).

3.1.21 BendSpanner

BendSpanner objects are created by: Bend_spanner_engraver (page 289).

Standard settings:

avoid-slur (symbol):
'ignore

Method of handling slur collisions. Choices are inside, outside, around, and ignore. inside adjusts the slur if needed to keep the grob inside the slur. outside moves the grob vertically to the outside of the slur. around moves the grob vertically to the outside of the slur only if there is a collision. ignore does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), outside and around behave like ignore.

baseline-skip (dimension, in staff space):
3

Distance between base lines of multiple lines of text.

before-line-breaking (boolean):
bend::target-cautionary

Dummy property, used to trigger a callback function.

details (list):

```
'((arrow-stencil
    .
    #<procedure bend::arrow-head-stencil (thickness x-y-coords height width c
  (curvature-factor . 0.35)
  (bend-arrowhead-height . 1.25)
  (bend-arrowhead-width . 0.8)
  (bend-amount-strings
    (quarter . "1/4")
    (half . "1/2")
    (three-quarter . "3/4")
    (full . #f))
  (curve-x-padding-line-end . 0.5)
  (curve-y-padding-line-end . 1)
  (dashed-line-settings 0.4 0.4 0)
  (head-text-break-visibility . #(#f #t #t))
  (horizontal-left-padding . 0.1)
  (successive-level . 1)
  (target-visibility . #f)
  (vertical-padding . 0.2)
  (y-distance-from-tabstaff-to-arrow-tip . 2.75))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`font-encoding` (symbol):
`'latin1`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number):
`-2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.15`

Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`spanner-id` (index or symbol):
`""`

An identifier to distinguish concurrent spanners.

`stencil` (stencil):
`bend-spanner::print`

The symbol to print.

`style` (symbol):
`'()`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`text` (markup)
Text markup. See Section "Formatting text" in *Notation Reference*.

`thickness` (number):
`1`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`word-space` (dimension, in staff space):
`0.6`

Space to insert between words in texts.

`Y-offset` (number):
`0`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `bend-interface` (page 531), `font-interface` (page 543), `grob-interface` (page 548), `line-spanner-interface` (page 561), `outside-staff-interface` (page 572), `text-interface` (page 596), and `text-script-interface` (page 597).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.22 BreakAlignGroup

`BreakAlignGroup` objects are created by: `Break_align_engraver` (page 290).

Standard settings:

`axes` (list):
`'(0)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-average-anchor`

Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-anchor-alignment` (number):
`ly:break-aligned-interface::calc-joint-anchor-alignment`

Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's extent.

`break-visibility` (vector):
`ly:break-aligned-interface::calc-break-visibility`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `break-aligned-interface` (page 533), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.23 BreakAlignment

`BreakAlignment` objects are created by: `Break_align_engraver` (page 290).

Standard settings:

`axes` (list):

```
'(0)
```

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`break-align-orders` (vector):

```
#((left-edge
    cue-end-clef
    ambitus
    breathing-sign
    clef
    cue-clef
    staff-bar
    key-cancellation
    key-signature
    time-signature
    custos)
  (left-edge
    cue-end-clef
    ambitus
    breathing-sign
    clef
    cue-clef
    staff-bar
    key-cancellation
    key-signature
    time-signature
    custos)
  (left-edge
    ambitus
    breathing-sign
    clef
    key-cancellation
    key-signature
    time-signature
    staff-bar
    cue-clef
    custos))
```

This is a vector of 3 lists: `#(`*`end-of-line unbroken start-of-line`*`)`. Each list contains *break-align symbols* that specify an order of breakable items (see Section "break-alignment-interface" in *Internals Reference*).

For example, this places time signatures before clefs:

```
\override Score.BreakAlignment.break-align-orders =
  #(make-vector 3 '(left-edge
```

```
                                        cue-end-clef
                                        ambitus
                                        breathing-sign
                                        time-signature
                                        clef
                                        cue-clef
                                        staff-bar
                                        key-cancellation
                                        key-signature
                                        custos))
```

`non-musical` (boolean):

```
#t
```

True if the grob belongs to a `NonMusicalPaperColumn`.

`stacking-dir` (direction):

```
1
```

Stack objects in which direction?

`X-extent` (pair of numbers):

```
ly:axis-group-interface::width
```

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `break-alignment-interface` (page 534), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.24 BreathingSign

`BreathingSign` objects are created by: `Breathing_sign_engraver` (page 290).

Standard settings:

`break-align-symbol` (symbol):

```
'breathing-sign
```

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):

```
#(#t #t #f)
```

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`direction` (direction):

```
1
```

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`non-musical` (boolean):

```
#t
```

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):

```
'((ambitus extra-space . 2.0)
```

```
(custos minimum-space . 1.0)
(key-signature minimum-space . 1.5)
(time-signature minimum-space . 1.5)
(staff-bar minimum-space . 1.5)
(clef minimum-space . 2.0)
(cue-clef minimum-space . 2.0)
(cue-end-clef minimum-space . 2.0)
(first-note fixed-space . 1.0)
(right-edge extra-space . 0.1))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *break-align-symbol* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *spacing-style* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
: `ly:text-interface::print`

The symbol to print.

`text` (markup):
: `'(#<procedure musicglyph-markup (layout props glyph-name)>
"scripts.rcomma")`

Text markup. See Section "Formatting text" in *Notation Reference*.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
: `#<unpure-pure-container #<primitive-procedure ly:breathing-sign::offset-callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-aligned-interface` (page 533), `breathing-sign-interface` (page 535), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.25 CenteredBarNumber

`CenteredBarNumber` objects are created by: `Bar_number_engraver` (page 286).

Standard settings:

`extra-spacing-width` (pair of numbers):
: `'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-family` (symbol):
: `'roman`

The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`font-size` (number):
: `0`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`self-alignment-X` (number):
`0`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
`centered-text-interface::print`
The symbol to print.

This object supports the following interface(s): `bar-number-interface` (page 528), `centered-bar-number-interface` (page 535), `centered-text-interface` (page 536), `font-interface` (page 543), `grob-interface` (page 548), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.26 CenteredBarNumberLineSpanner

`CenteredBarNumberLineSpanner` objects are created by: `Centered_bar_number_align_engraver` (page 290).

Standard settings:

`after-line-breaking` (boolean):
`ly:side-position-interface::move-to-extremal-staff`
Dummy property, used to trigger callback for `after-line-breaking`.

`axes` (list):
`'(1)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`outside-staff-priority` (number):
`1200`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`4`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-element-stencils> #<primitive-procedure ly:grob::pure-vertical-skylines-from-element-stencils> >`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `bar-number-interface` (page 528), `centered-bar-number-line-spanner-interface` (page 536), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.27 ChordName

`ChordName` objects are created by: `Chord_name_engraver` (page 290).

Standard settings:

`after-line-breaking` (boolean):
`ly:chord-name::after-line-breaking`
Dummy property, used to trigger callback for `after-line-breaking`.

`extra-spacing-height` (pair of numbers):
`'(0.2 . -0.2)`
In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):
`'(-0.5 . 0.5)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-family` (symbol):
`'sans`
The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`font-size` (number):
`1.5`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`word-space` (dimension, in staff space):
`0.0`
Space to insert between words in texts.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `chord-name-interface` (page 536), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `rhythmic-grob-interface` (page 577), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.28 Clef

`Clef` objects are created by: `Clef_engraver` (page 291).

Standard settings:

`avoid-slur` (symbol):
`'inside`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-extent-aligned-anchor`
Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-anchor-alignment` (number):
`1`
Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's extent.

`break-align-symbol` (symbol):
`'clef`
This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
: `#(#f #f #t)`

  A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):
: `pure-from-neighbor-interface::extra-spacing-height-at-beginning-of-line`

  In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`glyph-name` (string):
: `ly:clef::calc-glyph-name`

  The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`non-musical` (boolean):
: `#t`

  True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):
: ```
  '((cue-clef extra-space . 2.0)
    (staff-bar extra-space . 0.7)
    (ambitus extra-space . 1.15)
    (key-cancellation minimum-space . 3.5)
    (key-signature minimum-space . 3.5)
    (time-signature minimum-space . 4.2)
    (first-note minimum-fixed-space . 5.0)
    (next-note extra-space . 1.0)
    (right-edge extra-space . 0.5))
  ```

  An alist that specifies distances from this grob to other breakable items, using the format:

  ```
  '((break-align-symbol . (spacing-style . space))
    (break-align-symbol . (spacing-style . space))
    ...)
  ```

  Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

  `first-note`
  : used when the grob is just left of the first note on a line

  `next-note`
  : used when the grob is just left of any other note; if not set, the value of `first-note` gets used

  `right-edge`
  : used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)
  ```

Choices for *spacing-style* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
`ly:clef::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-aligned-interface` (page 533), `clef-interface` (page 536), `font-interface` (page 543), `grob-interface` (page 548), `pure-from-neighbor-interface` (page 576), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.29 ClefModifier

`ClefModifier` objects are created by: `Clef_engraver` (page 291), and `Cue_clef_engraver` (page 294).

Standard settings:

`break-visibility` (vector):
`#<procedure #f (grob)>`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`clef-alignments` (list):
`'((G -0.2 . 0.1) (F -0.3 . -0.2) (C 0 . 0))`

An alist of parent-alignments that should be used for clef modifiers with various clefs

`color` (color):
`#<procedure #f (grob)>`

The color of this grob.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number):
`-4`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`parent-alignment-X` (number):
`ly:clef-modifier::calc-parent-alignment`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`staff-padding` (dimension, in staff space):
`0.7`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`transparent` (boolean):
`#<procedure #f (grob)>`
This makes the grob invisible.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`
Two skylines, one above and one below this grob.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `clef-modifier-interface` (page 537), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.30 ClusterSpanner

`ClusterSpanner` objects are created by: `Cluster_spanner_engraver` (page 292).

Standard settings:

`minimum-length` (dimension, in staff space):
`0.0`
Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`padding` (dimension, in staff space):
`0.25`
Add this much extra space between objects that are next to each other.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`
Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:cluster::print`
The symbol to print.

`style` (symbol):
`'ramp`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

This object supports the following interface(s): `cluster-interface` (page 537), and `grob-interface` (page 548).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.31 ClusterSpannerBeacon

`ClusterSpannerBeacon` objects are created by: `Cluster_spanner_engraver` (page 292).

Standard settings:

`Y-extent` (pair of numbers):
`ly:cluster-beacon::height`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `cluster-beacon-interface` (page 537), `grob-interface` (page 548), and `rhythmic-grob-interface` (page 577).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.32 CombineTextScript

`CombineTextScript` objects are created by: `Part_combine_engraver` (page 315).

Standard settings:

`avoid-slur` (symbol):
`'outside`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`baseline-skip` (dimension, in staff space):
`2`
Distance between base lines of multiple lines of text.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-series` (symbol):
`'bold`
Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`outside-staff-priority` (number):
`450`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.5`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number)
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`script-priority` (number):
`200`

A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`self-alignment-X` (number)
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`0.5`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), `text-interface` (page 596), and `text-script-interface` (page 597).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.33 ControlPoint

`ControlPoint` objects are created by: `Show_control_points_engraver` (page 321).

Standard settings:

`color` (color):
`"IndianRed"`
The color of this grob.

`horizontal-skylines` (pair of skylines)
Two skylines, one to the left and one to the right of this grob.

`layer` (integer):
`3`
An integer which determines the order of printing objects. 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.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`text` (markup):
```
'(#<procedure draw-circle-markup (layout props radius thickness filled)>
  0.3
  0.01
  #t)
```
Text markup. See Section “Formatting text” in *Notation Reference*.

`vertical-skylines` (pair of skylines)
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number):
`#<procedure #f (grob)>`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<procedure #f (grob)>`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `control-point-interface` (page 538), `grob-interface` (page 548), `sticky-grob-interface` (page 593), and `text-interface` (page 596).

This object can be of either of the following classes: `item-interface,Item` (page ⟨undefined⟩), and `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.34 ControlPolygon

`ControlPolygon` objects are created by: `Show_control_points_engraver` (page 321).

Standard settings:

`color` (color):
: `"BurlyWood"`

  The color of this grob.

`extroversion` (number):
: `0.5`

  For polygons, how the thickness of the line is spread on each side of the exact polygon whith ideal zero thickness. If this is 0, the middle of line is on the polygon. If 1, the line sticks out of the polygon. If -1, the outer side of the line is exactly on the polygon. Other numeric values are interpolated.

`filled` (boolean)
: Whether an object is filled with ink.

`horizontal-skylines` (pair of skylines)
: Two skylines, one to the left and one to the right of this grob.

`layer` (integer):
: `2`

  An integer which determines the order of printing objects. 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.

`stencil` (stencil):
: `ly:text-interface::print`

  The symbol to print.

`text` (markup):
: `control-polygon::calc-text`

  Text markup. See Section “Formatting text” in *Notation Reference*.

`thickness` (number):
: `1.2`

  For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual “pen” that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines)
: Two skylines, one above and one below this grob.

`X-extent` (pair of numbers)
: Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
: Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `control-polygon-interface` (page 538), `grob-interface` (page 548), `sticky-grob-interface` (page 593), and `text-interface` (page 596).

This object can be of either of the following classes: `item-interface,Item` (page ⟨undefined⟩), and `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.35 CueClef

`CueClef` objects are created by: `Cue_clef_engraver` (page 294).

Standard settings:

`avoid-slur` (symbol):
`'inside`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-extent-aligned-anchor`

Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-symbol` (symbol):
`'cue-clef`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#f #f #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):
`pure-from-neighbor-interface::extra-spacing-height-at-beginning-of-line`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`font-size` (number):
`-4`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`full-size-change` (boolean):
`#t`

Don't make a change clef smaller.

`glyph-name` (string):
`ly:clef::calc-glyph-name`

The glyph name within the font.

In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):

```
'((staff-bar minimum-space . 2.7)
  (key-cancellation minimum-space . 3.5)
  (key-signature minimum-space . 3.5)
  (time-signature minimum-space . 4.2)
  (custos minimum-space . 0.0)
  (first-note minimum-fixed-space . 3.0)
  (next-note extra-space . 1.0)
  (right-edge extra-space . 0.5))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
`ly:clef::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-aligned-interface` (page 533), `clef-interface` (page 536), `font-interface` (page 543), `grob-interface` (page 548), `pure-from-neighbor-interface` (page 576), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.36 CueEndClef

`CueEndClef` objects are created by: `Cue_clef_engraver` (page 294).

Standard settings:

`avoid-slur` (symbol):
`'inside`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around`

moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-extent-aligned-anchor`

Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-symbol` (symbol):
`'cue-end-clef`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#t #t #f)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):
`pure-from-neighbor-interface::extra-spacing-height-at-beginning-of-line`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`font-size` (number):
`-4`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`full-size-change` (boolean):
`#t`

Don't make a change clef smaller.

`glyph-name` (string):
`ly:clef::calc-glyph-name`

The glyph name within the font.

In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):

```
'((clef extra-space . 0.7)
  (cue-clef extra-space . 0.7)
  (staff-bar extra-space . 0.7)
```

```
(key-cancellation minimum-space . 3.5)
(key-signature minimum-space . 3.5)
(time-signature minimum-space . 4.2)
(first-note minimum-fixed-space . 5.0)
(next-note extra-space . 1.0)
(right-edge extra-space . 0.5))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *break-align-symbol* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *spacing-style* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
`ly:clef::print`

The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-aligned-interface` (page 533), `clef-interface` (page 536), `font-interface` (page 543), `grob-interface` (page 548), `pure-from-neighbor-interface` (page 576), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.37 Custos

`Custos` objects are created by: `Custos_engraver` (page 295).

Standard settings:

`break-align-symbol` (symbol):
`'custos`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#t #f #f)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`neutral-direction` (direction):
`-1`

Which direction to take in the center of the staff.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):
```
'((first-note minimum-fixed-space . 0.0)
  (right-edge extra-space . 0.1))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *break-align-symbol* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *spacing-style* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
: `ly:custos::print`

The symbol to print.

`style` (symbol):
: `'vaticana`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-aligned-interface` (page 533), `custos-interface` (page 538), `font-interface` (page 543), `grob-interface` (page 548), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.38 DotColumn

`DotColumn` objects are created by: `Dot_column_engraver` (page 295), and `Vaticana_ligature_engraver` (page 331).

Standard settings:

`axes` (list):
`'(0)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`chord-dots-limit` (integer):
`3`
Limits the column of dots on each chord to the height of the chord plus `chord-dots-limit` staff-positions.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `dot-column-interface` (page 538), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.39 Dots

`Dots` objects are created by: `Dots_engraver` (page 296).

Standard settings:

`avoid-slur` (symbol):
`'inside`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`dot-count` (integer):
`dots::calc-dot-count`

The number of dots.

`extra-spacing-height` (pair of numbers):
`'(-0.5 . 0.5)`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):
`'(0.0 . 0.2)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`staff-position` (number):
`dots::calc-staff-position`

Vertical position, measured in half staff spaces, counted from the middle line.

`stencil` (stencil):
`ly:dots::print`

The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `dots-interface` (page 539), `font-interface` (page 543), `grob-interface` (page 548), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.40 DoublePercentRepeat

`DoublePercentRepeat` objects are created by: `Double_percent_repeat_engraver` (page 296).

Standard settings:

`break-align-symbol` (symbol):
`'staff-bar`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#t #t #f)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`dot-negative-kern` (number):
`0.75`

The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

`font-encoding` (symbol):
`'fetaMusic`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`slash-negative-kern` (number):
`1.6`

The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

`slope` (number):
`1.0`

The slope of this object.

`stencil` (stencil):
`ly:percent-repeat-item-interface::double-percent`

The symbol to print.

`thickness` (number):
`0.48`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `break-aligned-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `percent-repeat-interface` (page 574), and `percent-repeat-item-interface` (page 574).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.41 DoublePercentRepeatCounter

`DoublePercentRepeatCounter` objects are created by: `Double_percent_repeat_engraver` (page 296).

Standard settings:

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-encoding` (symbol):
`'fetaText`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-size` (number):
`-2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.2`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`0.25`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):

`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `percent-repeat-interface` (page 574), `percent-repeat-item-interface` (page 574), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.42 DoubleRepeatSlash

`DoubleRepeatSlash` objects are created by: `Slash_repeat_engraver` (page 321).

Standard settings:

`dot-negative-kern` (number):

`0.75`

The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

`font-encoding` (symbol):

`'fetaMusic`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`slash-negative-kern` (number):

`1.6`

The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

`slope` (number):

`1.0`

The slope of this object.

`stencil` (stencil):

`ly:percent-repeat-item-interface::beat-slash`

The symbol to print.

`thickness` (number):

`0.48`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-extent` (pair of numbers):

`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `percent-repeat-interface` (page 574), `percent-repeat-item-interface` (page 574), and `rhythmic-grob-interface` (page 577).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.43 DurationLine

`DurationLine` objects are created by: `Duration_line_engraver` (page 297).

Standard settings:

`after-line-breaking` (boolean):

```
ly:spanner::kill-zero-spanned-time
```

Dummy property, used to trigger callback for `after-line-breaking`.

`arrow-length` (number):

```
2
```

Arrow length.

`arrow-width` (number):

```
1.5
```

Arrow width.

`bound-details` (list):

```
'((right (end-on-accidental . #t)
         (end-on-arpeggio . #t)
         (padding . 0.4)
         (end-style . #f))
  (right-broken (padding . 0.4) (end-style . #f))
  (left-broken (padding . 0.4))
  (left (padding . -0.3) (start-at-dot . #f)))
```

An alist of properties for determining attachments of spanners to edges.

`breakable` (boolean):

```
#t
```

Allow breaks here.

`details` (list):

```
'((hook-height . 0.34)
  (hook-thickness . #f)
  (hook-direction . 1))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`minimum-length` (dimension, in staff space):

```
2
```

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`minimum-length-after-break` (dimension, in staff space):

```
6
```

If set, try to make a broken spanner starting a line this long. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance to the notehead.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`duration-line::print`

The symbol to print.

`style` (symbol):
`'beam`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number):
`4`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`to-barline` (boolean)
If true, the spanner will stop at the bar line just before it would otherwise stop.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`Y-offset` (number):
`0`

The vertical amount that this object is moved relative to its Y-parent.

`zigzag-length` (dimension, in staff space):
`1`

The length of the lines of a zigzag, relative to `zigzag-width`. A value of `1` gives 60-degree zigzags.

`zigzag-width` (dimension, in staff space):
`1`

The width of one zigzag squiggle. This number is adjusted slightly so that the spanner line can be constructed from a whole number of squiggles.

This object supports the following interface(s): `duration-line-interface` (page 539), `font-interface` (page 543), `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), and `unbreakable-spanner-interface` (page 603).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.44 DynamicLineSpanner

DynamicLineSpanner objects are created by: Dynamic_align_engraver (page 297).

Standard settings:

axes (list):
'(1)

List of axis numbers. In the case of alignment grobs, this should contain only one number.

direction (direction):
-1

If side-axis is 0 (or X), then this property determines whether the object is placed LEFT, CENTER or RIGHT with respect to the other object. Otherwise, it determines whether the object is placed UP, CENTER or DOWN. Numerical values may also be used: UP=1, DOWN=-1, LEFT=-1, RIGHT=1, CENTER=0.

minimum-space (dimension, in staff space):
1.2

Minimum distance that the victim should move (after padding).

outside-staff-priority (number):
250

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller outside-staff-priority is closer to the staff.

padding (dimension, in staff space):
0.6

Add this much extra space between objects that are next to each other.

side-axis (number):
1

If the value is X (or equivalently 0), the object is placed horizontally next to the other object. If the value is Y or 1, it is placed vertically.

slur-padding (number):
0.3

Extra distance between slur and script.

staff-padding (dimension, in staff space):
0.1

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

vertical-skylines (pair of skylines):
#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-element-stencils> #<primitive-procedure ly:grob::pure-vertical-skylines-from-element-stencils> >

Two skylines, one above and one below this grob.

X-extent (pair of numbers):
ly:axis-group-interface::width

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `dynamic-interface` (page 540), `dynamic-line-spanner-interface` (page 540), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.45 DynamicText

`DynamicText` objects are created by: `Dynamic_engraver` (page 297).

Standard settings:

`direction` (direction):
`ly:script-interface::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-encoding` (symbol):
`'fetaText`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-series` (symbol):
`'bold`

Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`right-padding` (dimension, in staff space):
`0.5`
Space to insert on the right side of an object (e.g., between note and its accidentals).

`self-alignment-X` (number):
`0`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`
Two skylines, one above and one below this grob.

`X-align-on-main-noteheads` (boolean):
`#t`
If true, this grob will ignore suspended noteheads when aligning itself on NoteColumn.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<procedure #f (grob)> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `dynamic-interface` (page 540), `dynamic-text-interface` (page 540), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `script-interface` (page 578), `self-alignment-interface` (page 579), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.46 DynamicTextSpanner

`DynamicTextSpanner` objects are created by: `Dynamic_engraver` (page 297).

Standard settings:

`before-line-breaking` (boolean):
`dynamic-text-spanner::before-line-breaking`

Dummy property, used to trigger a callback function.

`bound-details` (list):

```
'((right (attach-dir . -1)
         (Y . 0)
         (padding . 0.75))
  (right-broken (attach-dir . 1) (padding . 0.0))
  (left (attach-dir . -1)
        (Y . 0)
        (stencil-offset -0.75 . -0.5)
        (padding . 0.75))
  (left-broken (attach-dir . 1)))
```

An alist of properties for determining attachments of spanners to edges.

`dash-fraction` (number):

`0.2`

Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`dash-period` (number):

`3.0`

The length of one dash together with whitespace. If negative, no line is drawn at all.

`font-shape` (symbol):

`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number):

`1`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`left-bound-info` (list):

`ly:line-spanner::calc-left-bound-info-and-text`

An alist of properties for determining attachments of spanners to edges.

`minimum-length` (dimension, in staff space):

`2.0`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`minimum-Y-extent` (pair of numbers):

`'(-1 . 1)`

Minimum size of an object in Y dimension, measured in `staff-space` units.

`right-bound-info` (list):

`ly:line-spanner::calc-right-bound-info`

An alist of properties for determining attachments of spanners to edges.

`skyline-horizontal-padding` (number):

`0.2`

For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:line-spanner::print`

The symbol to print.

`style` (symbol):
`'dashed-line`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

This object supports the following interface(s): `dynamic-interface` (page 540), `dynamic-text-spanner-interface` (page 540), `font-interface` (page 543), `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.47 Episema

`Episema` objects are created by: `Episema_engraver` (page 298).

Standard settings:

`bound-details` (list):
```
'((left (Y . 0) (padding . 0) (attach-dir . -1))
  (right (Y . 0) (padding . 0) (attach-dir . 1)))
```

An alist of properties for determining attachments of spanners to edges.

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`left-bound-info` (list):
`ly:line-spanner::calc-left-bound-info`

An alist of properties for determining attachments of spanners to edges.

`right-bound-info` (list):
`ly:line-spanner::calc-right-bound-info`

An alist of properties for determining attachments of spanners to edges.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`stencil` (stencil):
`ly:line-spanner::print`

The symbol to print.

`style` (symbol):
`'line`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `episema-interface` (page 541), `font-interface` (page 543), `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.48 FingerGlideSpanner

`FingerGlideSpanner` objects are created by: `Finger_glide_engraver` (page 300).

Standard settings:

`bound-details` (list):
```
'((right (attach-dir . -1)
         (right-stub-length . 1)
         (padding . 0.2))
  (left (attach-dir . 1)
        (left-stub-length . 1)
        (padding . 0.2)))
```

An alist of properties for determining attachments of spanners to edges.

`dash-fraction` (number):
`0.4`

Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`dash-period` (number):
`1`

The length of one dash together with whitespace. If negative, no line is drawn at all.

`details` (list):
`'((bow-direction . #f))`

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`left-bound-info` (list):
`ly:line-spanner::calc-left-bound-info`
An alist of properties for determining attachments of spanners to edges.

`minimum-length` (dimension, in staff space):
`2.5`
Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`minimum-length-after-break` (dimension, in staff space):
`2.5`
If set, try to make a broken spanner starting a line this long. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance to the notehead.

`normalized-endpoints` (pair):
`ly:spanner::calc-normalized-endpoints`
Represents left and right placement over the total spanner, where the width of the spanner is normalized between 0 and 1.

`right-bound-info` (list):
`ly:line-spanner::calc-right-bound-info`
An alist of properties for determining attachments of spanners to edges.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`
Dummy variable for triggering spacing routines.

`stencil` (stencil):
`finger-glide::print`
The symbol to print.

`style` (symbol):
`'line`
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number):
`1.4`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

`zigzag-length` (dimension, in staff space):
`1`
The length of the lines of a zigzag, relative to `zigzag-width`. A value of `1` gives 60-degree zigzags.

`zigzag-width` (dimension, in staff space):

`1`

The width of one zigzag squiggle. This number is adjusted slightly so that the spanner line can be constructed from a whole number of squiggles.

This object supports the following interface(s): `finger-glide-interface` (page 542), `grob-interface` (page 548), and `line-spanner-interface` (page 561).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.49 Fingering

`Fingering` objects are created by: `Fingering_engraver` (page 300), and `New_fingering_engraver` (page 312).

Standard settings:

`add-stem-support` (boolean):

`only-if-beamed`

If set, the `Stem` object is included in this script's support.

`avoid-slur` (symbol):

`'around`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`direction` (direction):

`ly:script-interface::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-encoding` (symbol):

`'fetaText`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-size` (number):

`-5`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):

`0.5`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):

`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`parent-alignment-Y` (number):
`0`

Like `parent-alignment-X` but for the Y axis.

`script-priority` (number):
`100`

A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number):
`0`

Like `self-alignment-X` but for the Y axis.

`slur-padding` (number):
`0.2`

Extra distance between slur and script.

`staff-padding` (dimension, in staff space):
`0.5`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`text` (markup):
`fingering::calc-text`

Text markup. See Section "Formatting text" in *Notation Reference*.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `finger-interface` (page 543), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), `text-interface` (page 596), and `text-script-interface` (page 597).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.50 FingeringColumn

FingeringColumn objects are created by: Fingering_column_engraver (page 300).

Standard settings:

padding (dimension, in staff space):
0.2

Add this much extra space between objects that are next to each other.

snap-radius (number):
0.3

The maximum distance between two objects that will cause them to snap to alignment along an axis.

This object supports the following interface(s): fingering-column-interface (page 543), and grob-interface (page 548).

This object is of class item-interface,Item (page ⟨undefined⟩).

3.1.51 Flag

Flag objects are created by: Stem_engraver (page 324).

Standard settings:

color (color):
#<procedure #f (grob)>

The color of this grob.

glyph-name (string):
ly:flag::glyph-name

The glyph name within the font.

In the context of (span) bar lines, *glyph-name* represents a processed form of glyph, where decisions about line breaking etc. are already taken.

stencil (stencil):
ly:flag::print

The symbol to print.

transparent (boolean):
#<procedure #f (grob)>

This makes the grob invisible.

vertical-skylines (pair of skylines):
#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >

Two skylines, one above and one below this grob.

X-extent (pair of numbers):
ly:flag::width

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

X-offset (number):
ly:flag::calc-x-offset

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:flag::calc-y-offset> #<primitive-procedure ly:flag::pure-calc-y-offset> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `flag-interface` (page 543), `font-interface` (page 543), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.52 Footnote

`Footnote` objects are created by: `Footnote_engraver` (page 300).

Standard settings:

`annotation-balloon` (boolean)
Print the balloon around an annotation.

`annotation-line` (boolean):
`#t`
Print the line from an annotation to the grob that it annotates.

`automatically-numbered` (boolean):
`#<procedure #f (grob)>`
If set, footnotes are automatically numbered.

`break-visibility` (vector):
`#<procedure #f (grob)>`
A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`footnote` (boolean):
`#t`
Should this be a footnote or in-note?

`footnote-text` (markup):
`#<procedure #f (grob)>`
A footnote for the grob.

`stencil` (stencil):
`ly:balloon-interface::print`
The symbol to print.

`text` (markup):
`#<procedure #f (grob)>`
Text markup. See Section "Formatting text" in *Notation Reference*.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number):
`#<procedure #f (grob)>`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<procedure #f (grob)>`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `balloon-interface` (page 526), `font-interface` (page 543), `footnote-interface` (page 545), `grob-interface` (page 548), `sticky-grob-interface` (page 593), and `text-interface` (page 596).

This object can be of either of the following classes: `item-interface,Item` (page ⟨undefined⟩), and `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.53 FretBoard

`FretBoard` objects are created by: `Fretboard_engraver` (page 301).

Standard settings:

`after-line-breaking` (boolean):
`ly:chord-name::after-line-breaking`

Dummy property, used to trigger callback for `after-line-breaking`.

`extra-spacing-height` (pair of numbers):
`'(0.2 . -0.2)`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):
`'(-0.5 . 0.5)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`fret-diagram-details` (list):
`'((finger-code . below-string))`

An alist of detailed grob properties for fret diagrams. Each alist entry consists of a (*property* . *value*) pair. The properties which can be included in `fret-diagram-details` include the following:

- `barre-type` – Type of barre indication used. Choices include `curved`, `straight`, and `none`. Default `curved`.
- `capo-thickness` – Thickness of capo indicator, in multiples of fret-space. Default value 0.5.
- `dot-color` – Color of dots. Options include `black` and `white`. Default `black`.

- `dot-label-font-mag` – Magnification for font used to label fret dots. Default value 1.
- `dot-position` – Location of dot in fret space. Default 0.6 for dots without labels, 0.95-`dot-radius` for dots with labels.
- `dot-radius` – Radius of dots, in terms of fret spaces. Default value 0.425 for labeled dots, 0.25 for unlabeled dots.
- `finger-code` – Code for the type of fingering indication used. Options include `none`, `in-dot`, and `below-string`. Default `none` for markup fret diagrams, `below-string` for `FretBoards` fret diagrams.
- `fret-count` – The number of frets. Default 4.
- `fret-distance` – Multiplier to adjust the distance between frets. Default 1.0.
- `fret-label-custom-format` – The format string to be used label the lowest fret number, when `number-type` equals to `custom`. Default `"~a"`.
- `fret-label-font-mag` – The magnification of the font used to label the lowest fret number. Default 0.5.
- `fret-label-vertical-offset` – The offset of the fret label from the center of the fret in direction parallel to strings. Default 0.
- `fret-label-horizontal-offset` – The offset of the fret label from the center of the fret in direction orthogonal to strings. Default 0.
- `handedness` – Print the fret-diagram left- or right-handed. `-1`, `LEFT` for left ; `1`, `RIGHT` for right. Default `RIGHT`.
- `paren-padding` – The padding for the parenthesis. Default 0.05.
- `label-dir` – Side to which the fret label is attached. `-1`, `LEFT`, or `DOWN` for left or down; `1`, `RIGHT`, or `UP` for right or up. Default `RIGHT`.
- `mute-string` – Character string to be used to indicate muted string. Default `"x"`.
- `number-type` – Type of numbers to use in fret label. Choices include `roman-lower`, `roman-upper`, `arabic` and `custom`. In the later case, the format string is supplied by the `fret-label-custom-format` property. Default `roman-lower`.
- `open-string` – Character string to be used to indicate open string. Default `"o"`.
- `orientation` – Orientation of fret-diagram. Options include `normal`, `landscape`, and `opposing-landscape`. Default `normal`.
- `string-count` – The number of strings. Default 6.
- `string-distance` – Multiplier to adjust the distance between strings. Default 1.0.
- `string-label-font-mag` – The magnification of the font used to label fingerings at the string, rather than in the dot. Default value 0.6 for `normal` orientation, 0.5 for `landscape` and `opposing-landscape`.
- `string-thickness-factor` – Factor for changing thickness of each string in the fret diagram. Thickness of string k is given by `thickness` * (1+`string-thickness-factor`) ^ (k-1). Default 0.
- `top-fret-thickness` – The thickness of the top fret line, as a multiple of the standard thickness. Default value 3.
- `xo-font-magnification` – Magnification used for mute and open string indicators. Default value 0.5.

- `xo-padding` – Padding for open and mute indicators from top fret. Default value 0.25.

`stencil` (stencil):
`fret-board::calc-stencil`
The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `chord-name-interface` (page 536), `font-interface` (page 543), `fret-diagram-interface` (page 545), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `rhythmic-grob-interface` (page 577).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.54 Glissando

`Glissando` objects are created by: `Glissando_engraver` (page 302).

Standard settings:

`after-line-breaking` (boolean):
`ly:spanner::kill-zero-spanned-time`
Dummy property, used to trigger callback for `after-line-breaking`.

`bound-details` (list):
```
'((right (attach-dir . -1)
         (end-on-accidental . #t)
         (padding . 0.5))
  (left (attach-dir . 1)
        (padding . 0.5)
        (start-at-dot . #t)))
```
An alist of properties for determining attachments of spanners to edges.

`gap` (dimension, in staff space):
`0.5`
Size of a gap in a variable symbol.

`left-bound-info` (list):
`ly:line-spanner::calc-left-bound-info`
An alist of properties for determining attachments of spanners to edges.

`normalized-endpoints` (pair):
`ly:spanner::calc-normalized-endpoints`
Represents left and right placement over the total spanner, where the width of the spanner is normalized between 0 and 1.

`right-bound-info` (list):
`ly:line-spanner::calc-right-bound-info`
An alist of properties for determining attachments of spanners to edges.

`stencil` (stencil):
`ly:line-spanner::print`
The symbol to print.

`style` (symbol):
`'line`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`zigzag-width` (dimension, in staff space):
`0.75`

The width of one zigzag squiggle. This number is adjusted slightly so that the spanner line can be constructed from a whole number of squiggles.

This object supports the following interface(s): `glissando-interface` (page 547), `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), and `unbreakable-spanner-interface` (page 603).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.55 GraceSpacing

`GraceSpacing` objects are created by: `Grace_spacing_engraver` (page 303).

Standard settings:

`common-shortest-duration` (moment):
`grace-spacing::calc-shortest-duration`

The most common shortest note length. This is used in spacing. Enlarging this sets the score tighter.

`shortest-duration-space` (number):
`1.6`

Start with this multiple of `spacing-increment` space for the shortest duration. See also Section "spacing-spanner-interface" in *Internals Reference*.

`spacing-increment` (dimension, in staff space):
`0.8`

The unit of length for note-spacing. Typically, the width of a note head. See also Section "spacing-spanner-interface" in *Internals Reference*.

This object supports the following interface(s): `grace-spacing-interface` (page 547), `grob-interface` (page 548), and `spacing-options-interface` (page 586).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.56 GridLine

GridLine objects are created by: Grid_line_span_engraver (page 303).

Standard settings:

layer (integer):
0

An integer which determines the order of printing objects. 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.

parent-alignment-X (number):
0

Specify on which point of the parent the object is aligned. The value -1 means aligned on parent's left edge, 0 on center, and 1 right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from self-alignment-X property will be used.

self-alignment-X (number):
0

Specify alignment of an object. The value -1 means left aligned, 0 centered, and 1 right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

stencil (stencil):
ly:grid-line-interface::print

The symbol to print.

X-extent (pair of numbers):
ly:grid-line-interface::width

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

X-offset (number):
ly:self-alignment-interface::aligned-on-x-parent

The horizontal amount that this object is moved relative to its X-parent.

This object supports the following interface(s): grid-line-interface (page 548), grob-interface (page 548), and self-alignment-interface (page 579).

This object is of class item-interface,Item (page ⟨undefined⟩).

3.1.57 GridPoint

GridPoint objects are created by: Grid_point_engraver (page 303).

Standard settings:

X-extent (pair of numbers):
'(0 . 0)

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

Y-extent (pair of numbers):
'(0 . 0)

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grid-point-interface` (page 548), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.58 Hairpin

`Hairpin` objects are created by: `Dynamic_engraver` (page 297).

Standard settings:

`after-line-breaking` (boolean):
`ly:spanner::kill-zero-spanned-time`

Dummy property, used to trigger callback for `after-line-breaking`.

`bound-padding` (number):
`1.0`

The amount of padding to insert around spanner bounds.

`broken-bound-padding` (number):
`ly:hairpin::broken-bound-padding`

The amount of padding to insert when a spanner is broken at a line break.

`circled-tip` (boolean)
Put a circle at start/end of hairpins (al/del niente).

`endpoint-alignments` (pair of numbers):
`'(-1 . 1)`

A pair of numbers representing the alignments of an object's endpoints. E.g., the ends of a hairpin relative to `NoteColumn` grobs.

`grow-direction` (direction):
`hairpin::calc-grow-direction`

Crescendo or decrescendo?

`height` (dimension, in staff space):
`0.6666`

Height of an object in `staff-space` units.

`minimum-length` (dimension, in staff space):
`2.0`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`self-alignment-Y` (number):
`0`

Like `self-alignment-X` but for the Y axis.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:hairpin::print`

The symbol to print.

`thickness` (number):
`1.0`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

`to-barline` (boolean):
`#t`

If true, the spanner will stop at the bar line just before it would otherwise stop.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> #<primitive-procedure ly:hairpin::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:self-alignment-interface::y-aligned-on-self> #<primitive-procedure ly:self-alignment-interface::pure-y-aligned-on-self> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `dynamic-interface` (page 540), `grob-interface` (page 548), `hairpin-interface` (page 552), `line-interface` (page 560), `outside-staff-interface` (page 572), and `self-alignment-interface` (page 579).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.59 HorizontalBracket

`HorizontalBracket` objects are created by: `Horizontal_bracket_engraver` (page 304).

Standard settings:

`bracket-flare` (pair of numbers):
`'(0.5 . 0.5)`

A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`connect-to-neighbor` (pair):
`ly:tuplet-bracket::calc-connect-to-neighbors`

Pair of booleans, indicating whether this grob looks as a continued break.

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`padding` (dimension, in staff space):
`0.2`

Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`0.2`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:horizontal-bracket::print`

The symbol to print.

`thickness` (number):
`1.0`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `grob-interface` (page 548), `horizontal-bracket-interface` (page 554), `line-interface` (page 560), `outside-staff-interface` (page 572), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.60 HorizontalBracketText

`HorizontalBracketText` objects are created by: `Horizontal_bracket_engraver` (page 304).

Standard settings:

`direction` (direction):
`ly:horizontal-bracket-text::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-size` (number):
`-1`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.5`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`stencil` (stencil):
`ly:horizontal-bracket-text::print`

The symbol to print.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `horizontal-bracket-text-interface` (page 554), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.61 InstrumentName

`InstrumentName` objects are created by: `Instrument_name_engraver` (page 304).

Standard settings:

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`padding` (dimension, in staff space):
`0.3`
Add this much extra space between objects that are next to each other.

`self-alignment-X` (number):
`0`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number):
`0`
Like `self-alignment-X` but for the Y axis.

`stencil` (stencil):
`system-start-text::print`
The symbol to print.

`X-offset` (number):
`system-start-text::calc-x-offset`
The horizontal amount that this object is moved relative to its X-parent.

`Y-offset` (number):
`system-start-text::calc-y-offset`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `self-alignment-interface` (page 579), `side-position-interface` (page 581), `system-start-text-interface` (page 595), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.62 InstrumentSwitch

`InstrumentSwitch` objects are created by: `Instrument_switch_engraver` (page 305).

Standard settings:

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`outside-staff-priority` (number):
`500`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.5`
Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number)
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`-1`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`0.5`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.63 JumpScript

`JumpScript` objects are created by: `Jump_engraver` (page 305).

Standard settings:

`after-line-breaking` (boolean):
`ly:side-position-interface::move-to-extremal-staff`

Dummy property, used to trigger callback for `after-line-breaking`.

`baseline-skip` (dimension, in staff space):
`2`

Distance between base lines of multiple lines of text.

`break-align-symbols` (list):
`'(staff-bar key-signature clef)`

A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#t #t #f)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`outside-staff-horizontal-padding` (number):
`0.2`

By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-priority` (number):
`1500`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.8`
Add this much extra space between objects that are next to each other.

`self-alignment-X` (number):
`1`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`
Two skylines, one above and one below this grob.

`X-offset` (number):
`self-alignment-interface::self-aligned-on-breakable`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-alignable-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `jump-script-interface` (page 558), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.64 KeyCancellation

`KeyCancellation` objects are created by: `Key_engraver` (page 305).

Standard settings:

`break-align-symbol` (symbol):
`'key-cancellation`
This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#t #t #f)`
A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):

`pure-from-neighbor-interface::extra-spacing-height-including-staff`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):

`'(0.0 . 1.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`flat-positions` (list):

`'(2 3 4 2 1 2 1)`

Flats in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`non-musical` (boolean):

`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`sharp-positions` (list):

`'(4 5 4 2 3 2 3)`

Sharps in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`space-alist` (list):

```
'((time-signature extra-space . 1.25)
  (staff-bar extra-space . 0.6)
  (key-signature extra-space . 0.5)
  (cue-clef extra-space . 0.5)
  (right-edge extra-space . 0.5)
  (first-note fixed-space . 2.5)
  (custos extra-space . 1.0))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
: `ly:key-signature-interface::print`

  The symbol to print.

`vertical-skylines` (pair of skylines):
: `#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

  Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `break-aligned-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `key-cancellation-interface` (page 558), `key-signature-interface` (page 558), `pure-from-neighbor-interface` (page 576), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.65 KeySignature

`KeySignature` objects are created by: `Key_engraver` (page 305).

Standard settings:

`avoid-slur` (symbol):
`'inside`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-extent-aligned-anchor`

Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-anchor-alignment` (number):
`1`

Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's extent.

`break-align-symbol` (symbol):
`'key-signature`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#f #f #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):
`pure-from-neighbor-interface::extra-spacing-height-including-staff`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr'

to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):
`'(0.0 . 1.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`flat-positions` (list):
`'(2 3 4 2 1 2 1)`

Flats in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`sharp-positions` (list):
`'(4 5 4 2 3 2 3)`

Sharps in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`space-alist` (list):
```
'((ambitus extra-space . 1.15)
  (time-signature extra-space . 1.15)
  (staff-bar extra-space . 1.1)
  (cue-clef extra-space . 0.5)
  (right-edge extra-space . 0.5)
  (first-note fixed-space . 2.5))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
`ly:key-signature-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `break-aligned-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `key-signature-interface` (page 558), `pure-from-neighbor-interface` (page 576), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.66 KievanLigature

`KievanLigature` objects are created by: `Kievan_ligature_engraver` (page 307).

Standard settings:

`padding` (dimension, in staff space):
`0.5`

Add this much extra space between objects that are next to each other.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:kievan-ligature::print`

The symbol to print.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), and `kievan-ligature-interface` (page 558).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.67 LaissezVibrerTie

`LaissezVibrerTie` objects are created by: `Laissez_vibrer_engraver` (page 307).

Standard settings:

`control-points` (list of number pairs):
`ly:semi-tie::calc-control-points`

List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`details` (list):
`'((ratio . 0.333) (height-limit . 1.0))`

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`ly:tie::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-height` (pair of numbers):
`'(-0.5 . 0.5)`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`head-direction` (direction):
`-1`

Are the note heads left or right in a semitie?

`stencil` (stencil):
`ly:tie::print`

The symbol to print.

`thickness` (number):
`1.0`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `bezier-curve-interface` (page 532), `grob-interface` (page 548), `semi-tie-interface` (page 580), and `tie-interface` (page 598).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.68 LaissezVibrerTieColumn

`LaissezVibrerTieColumn` objects are created by: `Laissez_vibrer_engraver` (page 307).

Standard settings:

`head-direction` (direction):
`ly:semi-tie-column::calc-head-direction`

Are the note heads left or right in a semitie?

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), and `semi-tie-column-interface` (page 579).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.69 LedgerLineSpanner

`LedgerLineSpanner` objects are created by: `Ledger_line_engraver` (page 307).

Standard settings:

`layer` (integer):
`0`
An integer which determines the order of printing objects. 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.

`length-fraction` (number):
`0.25`
Multiplier for lengths. Used for determining ledger lines and stem lengths.

`minimum-length-fraction` (number):
`0.25`
Minimum length of ledger line as fraction of note head size.

`springs-and-rods` (boolean):
`ly:ledger-line-spanner::set-spacing-rods`
Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:ledger-line-spanner::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), and `ledger-line-spanner-interface` (page 559).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.70 LeftEdge

`LeftEdge` objects are created by: `Break_align_engraver` (page 290).

Standard settings:

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-extent-aligned-anchor`

Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-symbol` (symbol):
`'left-edge`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#f #f #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):

```
'((ambitus extra-space . 1.15)
  (breathing-sign minimum-space . 0.0)
  (cue-end-clef extra-space . 0.8)
  (clef extra-space . 0.8)
  (cue-clef extra-space . 0.8)
  (staff-bar extra-space . 0.0)
  (key-cancellation extra-space . 0.0)
  (key-signature extra-space . 0.8)
  (time-signature extra-space . 1.0)
  (custos extra-space . 0.0)
  (first-note fixed-space . 2.0)
  (right-edge extra-space . 0.0))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`X-extent` (pair of numbers):
`'(0 . 0)`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`'(0 . 0)`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `break-aligned-interface` (page 533), and `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.71 LigatureBracket

`LigatureBracket` objects are created by: `Ligature_bracket_engraver` (page 307).

Standard settings:

`bracket-visibility` (boolean or symbol):
`#t`

This controls the visibility of the tuplet bracket. Setting it to false prevents printing of the bracket. Setting the property to `if-no-beam` makes it print only if there is no beam associated with this tuplet bracket.

`connect-to-neighbor` (pair):
`ly:tuplet-bracket::calc-connect-to-neighbors`
Pair of booleans, indicating whether this grob looks as a continued break.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair):
`'(0.7 . 0.7)`
A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`padding` (dimension, in staff space):
`2.0`
Add this much extra space between objects that are next to each other.

`positions` (pair of numbers):
`ly:tuplet-bracket::calc-positions`
Pair of staff coordinates `(`*`start . end`*`)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`shorten-pair` (pair of numbers):
`'(-0.2 . -0.2)`
The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`staff-padding` (dimension, in staff space):
`0.25`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:tuplet-bracket::print`
The symbol to print.

`thickness` (number):
`1.6`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to `Staff.StaffSymbol.thickness`).

`tuplet-slur` (boolean)
Draw a slur instead of a bracket for tuplets.

`X-positions` (pair of numbers):
`ly:tuplet-bracket::calc-x-positions`
Pair of X staff coordinates of a spanner in the form `(`*`left . right`*`)`, where both *left* and *right* are in `staff-space` units of the current staff.

This object supports the following interface(s): `grob-interface` (page 548), `line-interface` (page 560), and `tuplet-bracket-interface` (page 601).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.72 LyricExtender

`LyricExtender` objects are created by: `Extender_engraver` (page 299).

Standard settings:

`minimum-length` (dimension, in staff space):
`1.5`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`stencil` (stencil):
`ly:lyric-extender::print`

The symbol to print.

`thickness` (number):
`0.8`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-extent` (pair of numbers):
`'(0 . 0)`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), `lyric-extender-interface` (page 562), and `lyric-interface` (page 563).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.73 LyricHyphen

`LyricHyphen` objects are created by: `Hyphen_engraver` (page 304).

Standard settings:

`after-line-breaking` (boolean):
`ly:spanner::kill-zero-spanned-time`

Dummy property, used to trigger callback for `after-line-breaking`.

`dash-period` (number):
`10.0`

The length of one dash together with whitespace. If negative, no line is drawn at all.

`height` (dimension, in staff space):
`0.42`

Height of an object in `staff-space` units.

`length` (dimension, in staff space):
`0.66`

User override for the stem length of unbeamed stems (each unit represents half a `staff-space`).

`minimum-distance` (dimension, in staff space):
`0.1`
Minimum distance between rest and notes or beam.

`minimum-length` (dimension, in staff space):
`0.3`
Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`padding` (dimension, in staff space):
`0.07`
Add this much extra space between objects that are next to each other.

`springs-and-rods` (boolean):
`ly:lyric-hyphen::set-spacing-rods`
Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:lyric-hyphen::print`
The symbol to print.

`thickness` (number):
`1.3`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`'(0 . 0)`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `lyric-hyphen-interface` (page 562), and `lyric-interface` (page 563).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.74 LyricSpace

`LyricSpace` objects are created by: `Hyphen_engraver` (page 304).

Standard settings:

`minimum-distance` (dimension, in staff space):
`0.45`
Minimum distance between rest and notes or beam.

`padding` (dimension, in staff space):
`0.0`
Add this much extra space between objects that are next to each other.

`springs-and-rods` (boolean):
`ly:lyric-hyphen::set-spacing-rods`
Dummy variable for triggering spacing routines.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), `lyric-hyphen-interface` (page 562), and `lyric-space-interface` (page 563).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.75 LyricText

`LyricText` objects are created by: `Lyric_engraver` (page 307).

Standard settings:

`extra-spacing-height` (pair of numbers):
`'(0.2 . -0.2)`
In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):
`'(0.0 . 0.0)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-series` (symbol):
`'medium`
Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-size` (number):
`1.0`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`parent-alignment-X` (number):
`'()`
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`skyline-horizontal-padding` (number):
`0.1`

For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`stencil` (stencil):
`lyric-text::print`

The symbol to print.

`text` (markup):
`#<procedure #f (grob)>`

Text markup. See Section "Formatting text" in *Notation Reference*.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`word-space` (dimension, in staff space):
`0.6`

Space to insert between words in texts.

`X-align-on-main-noteheads` (boolean):
`#t`

If true, this grob will ignore suspended noteheads when aligning itself on NoteColumn.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `lyric-syllable-interface` (page 563), `rhythmic-grob-interface` (page 577), `self-alignment-interface` (page 579), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.76 MeasureCounter

`MeasureCounter` objects are created by: `Measure_counter_engraver` (page 309).

Standard settings:

`count-from` (integer):
`1`

The first measure in a measure count receives this number. The following measures are numbered in increments from this initial value.

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-encoding` (symbol):
`'fetaText`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-size` (number):
`-2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`number-range-separator` (markup):
`"-"`

For a measure counter extending over several measures (like with compressed multi-measure rests), this is the separator between the two printed numbers.

`outside-staff-horizontal-padding` (number):
`0.5`

By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-priority` (number):
`750`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`spacing-pair` (pair):
`'(break-alignment . break-alignment)`

A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.

For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`staff-padding` (dimension, in staff space):
`0.5`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`centered-text-interface::print`

The symbol to print.

`text` (markup):
`measure-counter::text`

Text markup. See Section "Formatting text" in *Notation Reference*.

`word-space` (dimension, in staff space):
`0.2`

Space to insert between words in texts.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `centered-text-interface` (page 536), `font-interface` (page 543), `grob-interface` (page 548), `measure-counter-interface` (page 563), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.77 MeasureGrouping

`MeasureGrouping` objects are created by: `Measure_grouping_engraver` (page 309).

Standard settings:

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`height` (dimension, in staff space):
`2.0`

Height of an object in `staff-space` units.

`padding` (dimension, in staff space):
`2`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`3`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:measure-grouping::print`
The symbol to print.

`thickness` (number):
`1`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `grob-interface` (page 548), `measure-grouping-interface` (page 564), `outside-staff-interface` (page 572), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.78 MeasureSpanner

`MeasureSpanner` objects are created by: `Measure_spanner_engraver` (page 309).

Standard settings:

`connect-to-neighbor` (pair):
`ly:measure-spanner::calc-connect-to-neighbors`
Pair of booleans, indicating whether this grob looks as a continued break.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair):
`'(0.7 . 0.7)`
A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`outside-staff-priority` (number):
`750`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`self-alignment-X` (number):
`0`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`spacing-pair` (pair):
`'(staff-bar . staff-bar)`
A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.
For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`staff-padding` (dimension, in staff space):
`0.5`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:measure-spanner::print`
The symbol to print.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `line-interface` (page 560), `measure-spanner-interface` (page 564), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.79 MelodyItem

`MelodyItem` objects are created by: `Melody_engraver` (page 310).

Standard settings:

`neutral-direction` (direction):
`-1`
Which direction to take in the center of the staff.

This object supports the following interface(s): `grob-interface` (page 548), and `melody-spanner-interface` (page 565).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.80 MensuralLigature

`MensuralLigature` objects are created by: `Mensural_ligature_engraver` (page 310).

Standard settings:

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:mensural-ligature::print`

The symbol to print.

`thickness` (number):
`1.3`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), and `mensural-ligature-interface` (page 566).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.81 MetronomeMark

`MetronomeMark` objects are created by: `Metronome_mark_engraver` (page 310).

Standard settings:

`after-line-breaking` (boolean):
`ly:side-position-interface::move-to-extremal-staff`

Dummy property, used to trigger callback for `after-line-breaking`.

`break-align-symbols` (list):
`'(time-signature)`

A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#f #t #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`flag-style` (symbol):
`'default`

The style of the flag to be used with `MetronomeMark`. Available are `'modern-straight-flag`, `'old-straight-flag`, `flat-flag`, `mensural` and `'default`

`non-break-align-symbols` (list):
`'(paper-column-interface)`

A list of symbols that determine which NON-break-aligned interfaces to align this to.

`outside-staff-horizontal-padding` (number):
`0.2`

By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-priority` (number):
`1300`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.8`

Add this much extra space between objects that are next to each other.

`self-alignment-X` (number):
`-1`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`self-alignment-interface::self-aligned-on-breakable`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `break-alignable-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `metronome-mark-interface` (page 566), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.82 MultiMeasureRest

`MultiMeasureRest` objects are created by: `Multi_measure_rest_engraver` (page 311).

Standard settings:

`bound-padding` (number):
`0.5`
The amount of padding to insert around spanner bounds.

`expand-limit` (integer):
`10`
Maximum number of measures expanded in church rests.

`hair-thickness` (number):
`2.0`
Thickness of the thin line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *Staff*`.StaffSymbol.thickness`).

`max-symbol-separation` (number):
`8.0`
The maximum distance between symbols making up a church rest.

`round-up-exceptions` (list):
`'()`
A list of pairs where car is the numerator and cdr the denominator of a moment. Each pair in this list means that the multi-measure rests of the corresponding length will be rounded up to the longer rest. See *round-up-to-longer-rest*.

`spacing-pair` (pair):
`'(break-alignment . break-alignment)`
A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.

For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`springs-and-rods` (boolean):
`ly:multi-measure-rest::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:multi-measure-rest::print`

The symbol to print.

`thick-thickness` (number):
`6.6`

Thickness of the thick line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *Staff*`.StaffSymbol.thickness`).

`usable-duration-logs` (list):
`'(-3 -2 -1 0)`

List of `duration-log`s that can be used in typesetting the grob.

`voiced-position` (number):
`4`

The staff-position of a voiced `Rest`, negative if the rest has `direction DOWN`.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:multi-measure-rest::height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `multi-measure-interface` (page 566), `multi-measure-rest-interface` (page 567), `outside-staff-interface` (page 572), `rest-interface` (page 576), and `staff-symbol-referencer-interface` (page 590).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.83 MultiMeasureRestNumber

`MultiMeasureRestNumber` objects are created by: `Multi_measure_rest_engraver` (page 311).

Standard settings:

`bound-padding` (number):
`1.0`

The amount of padding to insert around spanner bounds.

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-encoding` (symbol):
`'fetaText`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`padding` (dimension, in staff space):
`0.4`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`springs-and-rods` (boolean):
`ly:multi-measure-rest::set-text-rods`

Dummy variable for triggering spacing routines.

`staff-padding` (dimension, in staff space):
`0.4`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):

`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):

`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `multi-measure-interface` (page 566), `multi-measure-rest-number-interface` (page 568), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.84 MultiMeasureRestScript

`MultiMeasureRestScript` objects are created by: `Multi_measure_rest_engraver` (page 311).

Standard settings:

`direction` (direction):

`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`outside-staff-padding` (number):

`0`

The padding to place between grobs when spacing according to `outside-staff-priority`. Two grobs with different `outside-staff-padding` values have the larger value of padding between them.

`outside-staff-priority` (number):

`40`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`parent-alignment-X` (number):

`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):

`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`staff-padding` (dimension, in staff space):
`0.25`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:script-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `multi-measure-interface` (page 566), `outside-staff-interface` (page 572), `script-interface` (page 578), `self-alignment-interface` (page 579), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.85 MultiMeasureRestText

`MultiMeasureRestText` objects are created by: `Multi_measure_rest_engraver` (page 311).

Standard settings:

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`outside-staff-priority` (number):
`450`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.2`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`skyline-horizontal-padding` (number):
`0.2`

For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`staff-padding` (dimension, in staff space):
`0.25`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `multi-measure-interface` (page 566), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.86 NonMusicalPaperColumn

`NonMusicalPaperColumn` objects are created by: `Paper_column_engraver` (page 315).

Standard settings:

`allow-loose-spacing` (boolean):
`#t`
If set, column can be detached from main spacing.

`axes` (list):
`'(0)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`before-line-breaking` (boolean):
`ly:paper-column::before-line-breaking`
Dummy property, used to trigger a callback function.

`font-size` (number):
`-7.5`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`full-measure-extra-space` (number):
`1.0`
Extra space that is allocated at the beginning of a measure with only one note. This property is read from the NonMusicalPaperColumn that begins the measure.

`horizontal-skylines` (pair of skylines):
`ly:separation-item::calc-skylines`
Two skylines, one to the left and one to the right of this grob.

`keep-inside-line` (boolean):
`#t`
If set, this column cannot have objects sticking into the margin.

`layer` (integer):
`1000`
An integer which determines the order of printing objects. 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.

`line-break-permission` (symbol):
`'allow`
Instructs the line breaker on whether to put a line break at this column. Can be `force` or `allow`.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`page-break-permission` (symbol):
`'allow`

Instructs the page breaker on whether to put a page break at this column. Can be `force` or `allow`.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `font-interface` (page 543), `grob-interface` (page 548), `separation-item-interface` (page 581), and `spaceable-grob-interface` (page 585).

This object is of class `paper-column-interface,Paper_column` (page ⟨undefined⟩).

3.1.87 NoteCollision

`NoteCollision` objects are created by: `Collision_engraver` (page 292).

Standard settings:

`axes` (list):
`'(0 1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`note-collision-threshold` (dimension, in staff space):
`1`

Simultaneous notes that are this close or closer in units of `staff-space` will be identified as vertically colliding. Used by `Stem` grobs for notes in the same voice, and `NoteCollision` grobs for notes in different voices. Default value 1.

`prefer-dotted-right` (boolean):
`#t`

For note collisions, prefer to shift dotted up-note to the right, rather than shifting just the dot.

`vertical-skylines` (pair of skylines):
`ly:axis-group-interface::calc-skylines`

Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), and `note-collision-interface` (page 568).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.88 NoteColumn

`NoteColumn` objects are created by: `Rhythmic_column_engraver` (page 320).

Standard settings:

`axes` (list):
`'(0 1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`bend-me` (boolean):
`'()`

Decide whether this grob is bent.

`horizontal-skylines` (pair of skylines):
`ly:separation-item::calc-skylines`

Two skylines, one to the left and one to the right of this grob.

`main-extent` (pair of numbers):
`ly:note-column::calc-main-extent`

The horizontal extent of a `NoteColumn` grob without taking suspended `NoteHead` grobs into account (i.e., `NoteHead`s forced into the unnatural direction of the `Stem` because of a chromatic clash).

`skyline-vertical-padding` (number):
`0.15`

The amount by which the left and right skylines of a column are padded vertically, beyond the `Y-extent`s and `extra-spacing-height`s of the constituent grobs in the column. Increase this to prevent interleaving of grobs from adjacent columns.

`vertical-skylines` (pair of skylines):
`ly:axis-group-interface::calc-skylines`

Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `bend-interface` (page 531), `grob-interface` (page 548), `note-column-interface` (page 568), and `separation-item-interface` (page 581).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.89 NoteHead

NoteHead objects are created by: Completion_heads_engraver (page 292), Drum_notes_engraver (page 296), and Note_heads_engraver (page 313).

Standard settings:

bend-me (boolean):
'()

Decide whether this grob is bent.

duration-log (integer):
note-head::calc-duration-log

The 2-log of the note head duration, i.e., 0 = whole note, 1 = half note, etc.

extra-spacing-height (pair of numbers):
ly:note-head::include-ledger-line-height

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to (-inf.0 . +inf.0).

glyph-name (string):
note-head::calc-glyph-name

The glyph name within the font.

In the context of (span) bar lines, *glyph-name* represents a processed form of glyph, where decisions about line breaking etc. are already taken.

parenthesis-friends (list):
'(accidental-grob dot)

A list of Grob types, as symbols. When parentheses enclose a Grob that has 'parenthesis-friends, the parentheses widen to include any child Grobs with type among 'parenthesis-friends.

stem-attachment (pair of numbers):
ly:note-head::calc-stem-attachment

An (*x* . *y*) pair where the stem attaches to the notehead.

stencil (stencil):
ly:note-head::print

The symbol to print.

X-offset (number):
ly:note-head::stem-x-shift

The horizontal amount that this object is moved relative to its X-parent.

Y-extent (pair of numbers):
#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

Y-offset (number):
#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `bend-interface` (page 531), `font-interface` (page 543), `gregorian-ligature-interface` (page 547), `grob-interface` (page 548), `ledgered-interface` (page 559), `ligature-head-interface` (page 560), `mensural-ligature-interface` (page 566), `note-head-interface` (page 569), `rhythmic-grob-interface` (page 577), `rhythmic-head-interface` (page 577), `staff-symbol-referencer-interface` (page 590), and `vaticana-ligature-interface` (page 603).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.90 NoteName

`NoteName` objects are created by: `Note_name_engraver` (page 313).

Standard settings:

`stencil` (stencil):
: `ly:text-interface::print`

  The symbol to print.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

  Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `note-name-interface` (page 570), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.91 NoteSpacing

`NoteSpacing` objects are created by: `Note_spacing_engraver` (page 313).

Standard settings:

`knee-spacing-correction` (number):
: `1.0`

  Factor for the optical correction amount for kneed beams. Set between `0` for no correction and `1` for full correction.

`same-direction-correction` (number):
: `0.25`

  Optical correction amount for stems that are placed in tight configurations. This amount is used for stems with the same direction to compensate for note head to stem distance.

`space-to-barline` (boolean):
: `#t`

  If set, the distance between a note and the following non-musical column will be measured to the bar line instead of to the beginning of the non-musical column. If there is a clef change followed by a bar line, for example, this means that we will try to space the non-musical column as though the clef is not there.

`stem-spacing-correction` (number):
: `0.5`

Optical correction amount for stems that are placed in tight configurations. For opposite directions, this amount is the correction for two normal sized stems that overlap completely.

This object supports the following interface(s): `grob-interface` (page 548), `note-spacing-interface` (page 570), and `spacing-interface` (page 586).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.92 OttavaBracket

`OttavaBracket` objects are created by: `Ottava_spanner_engraver` (page 314).

Standard settings:

`dash-fraction` (number):
`0.3`

Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`edge-height` (pair):
`'(0 . 0.8)`

A pair of numbers specifying the heights of the vertical edges: `(`*`left-height`* `.` *`right-height`*`)`.

`font-series` (symbol):
`'bold`

Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`minimum-length` (dimension, in staff space):
`0.3`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`outside-staff-priority` (number):
`400`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.5`

Add this much extra space between objects that are next to each other.

`shorten-pair` (pair of numbers):
`'(-0.8 . -0.6)`

The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`staff-padding` (dimension, in staff space):
`2.0`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:ottava-bracket::print`
The symbol to print.

`style` (symbol):
`'dashed-line`
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `horizontal-bracket-interface` (page 554), `line-interface` (page 560), `ottava-bracket-interface` (page 571), `outside-staff-interface` (page 572), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.93 PaperColumn

`PaperColumn` objects are created by: `Paper_column_engraver` (page 315).

Standard settings:

`allow-loose-spacing` (boolean):
`#t`
If set, column can be detached from main spacing.

`axes` (list):
`'(0)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`before-line-breaking` (boolean):
`ly:paper-column::before-line-breaking`
Dummy property, used to trigger a callback function.

`font-size` (number):
`-7.5`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`horizontal-skylines` (pair of skylines):
`ly:separation-item::calc-skylines`
Two skylines, one to the left and one to the right of this grob.

`keep-inside-line` (boolean):
`#t`
If set, this column cannot have objects sticking into the margin.

`layer` (integer):
`1000`
An integer which determines the order of printing objects. 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.

`skyline-vertical-padding` (number):
`0.08`
The amount by which the left and right skylines of a column are padded vertically, beyond the `Y-extent`s and `extra-spacing-height`s of the constituent grobs in the column. Increase this to prevent interleaving of grobs from adjacent columns.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `font-interface` (page 543), `grob-interface` (page 548), `separation-item-interface` (page 581), and `spaceable-grob-interface` (page 585).

This object is of class `paper-column-interface,Paper_column` (page ⟨undefined⟩).

3.1.94 Parentheses

`Parentheses` objects are created by: `Parenthesis_engraver` (page 315).

Standard settings:

`break-visibility` (vector):
`#<procedure #f (grob)>`
A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`font-size` (number):
`-6`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.2`
Add this much extra space between objects that are next to each other.

`stencil` (stencil):
`parentheses-interface::print`
The symbol to print.

`stencils` (list):
`parentheses-interface::calc-parenthesis-stencils`
Multiple stencils, used as intermediate value.

`X-extent` (pair of numbers):
`'(0 . 0)`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`parentheses-interface::y-extent`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `parentheses-interface` (page 574), and `sticky-grob-interface` (page 593).

This object can be of either of the following classes: `item-interface,Item` (page ⟨undefined⟩), and `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.95 PercentRepeat

`PercentRepeat` objects are created by: `Percent_repeat_engraver` (page 316).

Standard settings:

`dot-negative-kern` (number):
`0.75`

The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

`font-encoding` (symbol):
`'fetaMusic`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`slope` (number):
`1.0`

The slope of this object.

`spacing-pair` (pair):
`'(break-alignment . staff-bar)`

A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.

For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`springs-and-rods` (boolean):
`ly:multi-measure-rest::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:multi-measure-rest::percent`

The symbol to print.

`thickness` (number):
`0.48`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

This object supports the following interface(s): font-interface (page 543), grob-interface (page 548), multi-measure-rest-interface (page 567), and percent-repeat-interface (page 574).

This object is of class spanner-interface,Spanner (page ⟨undefined⟩).

3.1.96 PercentRepeatCounter

PercentRepeatCounter objects are created by: Percent_repeat_engraver (page 316).

Standard settings:

direction (direction):
1

If side-axis is 0 (or X), then this property determines whether the object is placed LEFT, CENTER or RIGHT with respect to the other object. Otherwise, it determines whether the object is placed UP, CENTER or DOWN. Numerical values may also be used: UP=1, DOWN=-1, LEFT=-1, RIGHT=1, CENTER=0.

font-encoding (symbol):
'fetaText

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are fetaMusic (Emmentaler), fetaBraces, fetaText (Emmentaler).

font-size (number):
-2

The font size, compared to the 'normal' size. 0 is style-sheet's normal size, -1 is smaller, +1 is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property fontSize is set, its value is added to this before the glyph is printed. Fractional values are allowed.

padding (dimension, in staff space):
0.2

Add this much extra space between objects that are next to each other.

parent-alignment-X (number):
0

Specify on which point of the parent the object is aligned. The value -1 means aligned on parent's left edge, 0 on center, and 1 right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from self-alignment-X property will be used.

self-alignment-X (number):
0

Specify alignment of an object. The value -1 means left aligned, 0 centered, and 1 right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

staff-padding (dimension, in staff space):
0.25

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `percent-repeat-interface` (page 574), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.97 PhrasingSlur

`PhrasingSlur` objects are created by: `Phrasing_slur_engraver` (page 316).

Standard settings:

`control-points` (list of number pairs):
`ly:slur::calc-control-points`

List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`details` (list):

```
'((region-size . 4)
  (head-encompass-penalty . 1000.0)
  (stem-encompass-penalty . 30.0)
  (edge-attraction-factor . 4)
  (same-slope-penalty . 20)
  (steeper-slope-factor . 50)
  (non-horizontal-penalty . 15)
  (max-slope . 1.1)
  (max-slope-factor . 10)
  (free-head-distance . 0.3)
  (free-slur-distance . 0.8)
  (gap-to-staffline-inside . 0.2)
  (gap-to-staffline-outside . 0.1)
  (extra-object-collision-penalty . 50)
```

```
(accidental-collision . 3)
(extra-encompass-free-distance . 0.3)
(extra-encompass-collision-distance . 0.8)
(head-slur-distance-max-ratio . 3)
(head-slur-distance-factor . 10)
(absolute-closeness-measure . 0.3)
(edge-slope-exponent . 1.7)
(close-to-edge-length . 2.5)
(encompass-object-range-overshoot . 0.5)
(slur-tie-extrema-min-distance . 0.2)
(slur-tie-extrema-min-distance-penalty . 2))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`ly:slur::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`height-limit` (dimension, in staff space):
`2.0`

Maximum slur height: The longer the slur, the closer it is to this height.

`minimum-length` (dimension, in staff space):
`1.5`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`ratio` (number):
`0.333`

Parameter for slur shape. The higher this number, the quicker the slur attains its `height-limit`.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:slur::print`

The symbol to print.

`thickness` (number):
`1.1`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
: `#<unpure-pure-container #<primitive-procedure ly:slur::vertical-skylines> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

  Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:slur::height> #<primitive-procedure ly:slur::pure-height> >`

  Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `bezier-curve-interface` (page 532), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `slur-interface` (page 583).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.98 PianoPedalBracket

`PianoPedalBracket` objects are created by: `Piano_pedal_engraver` (page 317).

Standard settings:

`bound-padding` (number):
: `1.0`

  The amount of padding to insert around spanner bounds.

`bracket-flare` (pair of numbers):
: `'(0.5 . 0.5)`

  A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`direction` (direction):
: `-1`

  If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair):
: `'(1.0 . 1.0)`

  A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`shorten-pair` (pair of numbers):
: `'(0.0 . 0.0)`

  The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`stencil` (stencil):
: `ly:piano-pedal-bracket::print`

  The symbol to print.

`style` (symbol):
: `'line`

  This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number):
`1.0`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.`StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

This object supports the following interface(s): `grob-interface` (page 548), `line-interface` (page 560), `piano-pedal-bracket-interface` (page 575), and `piano-pedal-interface` (page 575).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.99 RehearsalMark

`RehearsalMark` objects are created by: `Mark_engraver` (page 308).

Standard settings:

`after-line-breaking` (boolean):
`ly:side-position-interface::move-to-extremal-staff`

Dummy property, used to trigger callback for `after-line-breaking`.

`baseline-skip` (dimension, in staff space):
`2`

Distance between base lines of multiple lines of text.

`break-align-symbols` (list):
`'(staff-bar key-signature clef)`

A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#f #t #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right

side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-size` (number):
`2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`outside-staff-horizontal-padding` (number):
`0.2`

By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-priority` (number):
`1500`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.8`

Add this much extra space between objects that are next to each other.

`self-alignment-X` (number):
`break-alignable-interface::self-alignment-opposite-of-anchor`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`self-alignment-interface::self-aligned-on-breakable`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `break-alignable-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `mark-interface` (page 563), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.100 RepeatSlash

`RepeatSlash` objects are created by: `Slash_repeat_engraver` (page 321).

Standard settings:

`slash-negative-kern` (number):
`0.85`
The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

`slope` (number):
`1.7`
The slope of this object.

`stencil` (stencil):
`ly:percent-repeat-item-interface::beat-slash`
The symbol to print.

`thickness` (number):
`0.48`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), `percent-repeat-interface` (page 574), `percent-repeat-item-interface` (page 574), and `rhythmic-grob-interface` (page 577).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.101 RepeatTie

`RepeatTie` objects are created by: `Repeat_tie_engraver` (page 319).

Standard settings:

`control-points` (list of number pairs):
`ly:semi-tie::calc-control-points`

List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`details` (list):
`'((ratio . 0.333) (height-limit . 1.0))`
Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`ly:tie::calc-direction`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-height` (pair of numbers):
`'(-0.5 . 0.5)`
In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`head-direction` (direction):
`1`
Are the note heads left or right in a semitie?

`stencil` (stencil):
`ly:tie::print`
The symbol to print.

`thickness` (number):
`1.0`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`
Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `bezier-curve-interface` (page 532), `grob-interface` (page 548), `semi-tie-interface` (page 580), and `tie-interface` (page 598).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.102 RepeatTieColumn

`RepeatTieColumn` objects are created by: `Repeat_tie_engraver` (page 319).

Standard settings:

`head-direction` (direction):
`ly:semi-tie-column::calc-head-direction`
Are the note heads left or right in a semitie?

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), and `semi-tie-column-interface` (page 579).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.103 Rest

`Rest` objects are created by: `Completion_rest_engraver` (page 293), and `Rest_engraver` (page 320).

Standard settings:

`duration-log` (integer):
`stem::calc-duration-log`
The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`minimum-distance` (dimension, in staff space):
`0.25`
Minimum distance between rest and notes or beam.

`parenthesis-friends` (list):
`'(dot)`
A list of Grob types, as symbols. When parentheses enclose a Grob that has 'parenthesis-friends, the parentheses widen to include any child Grobs with type among 'parenthesis-friends.

`stencil` (stencil):
`ly:rest::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

`voiced-position` (number):
`4`
The staff-position of a voiced `Rest`, negative if the rest has `direction DOWN`.

`X-extent` (pair of numbers):
`ly:rest::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:rest::height> #<primitive-procedure ly:rest::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:rest::y-offset-callback> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `rest-interface` (page 576), `rhythmic-grob-interface` (page 577), `rhythmic-head-interface` (page 577), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.104 RestCollision

`RestCollision` objects are created by: `Rest_collision_engraver` (page 320).

Standard settings:

`minimum-distance` (dimension, in staff space):
`0.75`
Minimum distance between rest and notes or beam.

This object supports the following interface(s): `grob-interface` (page 548), and `rest-collision-interface` (page 576).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.105 Script

`Script` objects are created by: `Drum_notes_engraver` (page 296), `New_fingering_engraver` (page 312), and `Script_engraver` (page 320).

Standard settings:

`add-stem-support` (boolean):
`#t`
If set, the `Stem` object is included in this script's support.

`direction` (direction):
`ly:script-interface::calc-direction`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`font-encoding` (symbol):
`'fetaMusic`
The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`horizon-padding` (number):
`0.1`
The amount to pad the axis along which a `Skyline` is built for the `side-position-interface`.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`slur-padding` (number):
`0.2`

Extra distance between slur and script.

`staff-padding` (dimension, in staff space):
`0.25`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:script-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`script-interface::calc-x-offset`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `script-interface` (page 578), `self-alignment-interface` (page 579), and `side-position-interface` (page 581).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.106 ScriptColumn

`ScriptColumn` objects are created by: `Script_column_engraver` (page 320).

Standard settings:

`before-line-breaking` (boolean):
`ly:script-column::before-line-breaking`
Dummy property, used to trigger a callback function.

This object supports the following interface(s): `grob-interface` (page 548), and `script-column-interface` (page 577).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.107 ScriptRow

`ScriptRow` objects are created by: `Script_row_engraver` (page 321).

Standard settings:

`before-line-breaking` (boolean):
`ly:script-column::row-before-line-breaking`
Dummy property, used to trigger a callback function.

This object supports the following interface(s): `grob-interface` (page 548), and `script-column-interface` (page 577).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.108 Slur

`Slur` objects are created by: `Slur_engraver` (page 322).

Standard settings:

`avoid-slur` (symbol):
`'inside`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`control-points` (list of number pairs):
`ly:slur::calc-control-points`
List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`details` (list):

```
'((region-size . 4)
  (head-encompass-penalty . 1000.0)
  (stem-encompass-penalty . 30.0)
  (edge-attraction-factor . 4)
  (same-slope-penalty . 20)
  (steeper-slope-factor . 50)
  (non-horizontal-penalty . 15)
  (max-slope . 1.1)
  (max-slope-factor . 10)
  (free-head-distance . 0.3)
  (free-slur-distance . 0.8)
```

```
(gap-to-staffline-inside . 0.2)
(gap-to-staffline-outside . 0.1)
(extra-object-collision-penalty . 50)
(accidental-collision . 3)
(extra-encompass-free-distance . 0.3)
(extra-encompass-collision-distance . 0.8)
(head-slur-distance-max-ratio . 3)
(head-slur-distance-factor . 10)
(absolute-closeness-measure . 0.3)
(edge-slope-exponent . 1.7)
(close-to-edge-length . 2.5)
(encompass-object-range-overshoot . 0.5)
(slur-tie-extrema-min-distance . 0.2)
(slur-tie-extrema-min-distance-penalty . 2))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`ly:slur::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`height-limit` (dimension, in staff space):
`2.0`

Maximum slur height: The longer the slur, the closer it is to this height.

`line-thickness` (number):
`0.8`

For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`minimum-length` (dimension, in staff space):
`1.5`

Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`ratio` (number):
`0.25`

Parameter for slur shape. The higher this number, the quicker the slur attains its `height-limit`.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`

Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:slur::print`

The symbol to print.

`thickness` (number):
`1.2`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:slur::vertical-skylines> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:slur::height> #<primitive-procedure ly:slur::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `bezier-curve-interface` (page 532), `grob-interface` (page 548), `outside-staff-interface` (page 572), and `slur-interface` (page 583).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.109 SostenutoPedal

`SostenutoPedal` objects are created by: `Piano_pedal_engraver` (page 317).

Standard settings:

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`padding` (dimension, in staff space):
`0.0`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number)
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction.

Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `piano-pedal-script-interface` (page 576), `self-alignment-interface` (page 579), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.110 SostenutoPedalLineSpanner

`SostenutoPedalLineSpanner` objects are created by: `Piano_pedal_align_engraver` (page 316).

Standard settings:

`axes` (list):
`'(1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`minimum-space` (dimension, in staff space):
`1.0`

Minimum distance that the victim should move (after padding).

`outside-staff-priority` (number):
`1000`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`1.2`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`1.0`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-element-stencils> #<primitive-procedure ly:grob::pure-vertical-skylines-from-element-stencils> >`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), `outside-staff-interface` (page 572), `piano-pedal-interface` (page 575), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.111 SpacingSpanner

`SpacingSpanner` objects are created by: `Spacing_engraver` (page 322).

Standard settings:

`average-spacing-wishes` (boolean):
`#t`
If set, the spacing wishes are averaged over staves.

`base-shortest-duration` (moment):
`#<Mom 3/16>`

Spacing is based on the shortest notes in a piece. Normally, pieces are spaced as if notes at least as short as this are present.

`common-shortest-duration` (moment):
`ly:spacing-spanner::calc-common-shortest-duration`

The most common shortest note length. This is used in spacing. Enlarging this sets the score tighter.

`shortest-duration-space` (number):
`2.0`

Start with this multiple of `spacing-increment` space for the shortest duration. See also Section "spacing-spanner-interface" in *Internals Reference*.

`spacing-increment` (dimension, in staff space):
`1.2`

The unit of length for note-spacing. Typically, the width of a note head. See also Section "spacing-spanner-interface" in *Internals Reference*.

`springs-and-rods` (boolean):
`ly:spacing-spanner::set-springs`

Dummy variable for triggering spacing routines.

This object supports the following interface(s): `grob-interface` (page 548), `spacing-options-interface` (page 586), and `spacing-spanner-interface` (page 586).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.112 SpanBar

`SpanBar` objects are created by: `Span_bar_engraver` (page 323).

Standard settings:

`allow-span-bar` (boolean):
`#t`

If false, no inter-staff bar line will be created below this bar line.

`bar-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

The Y-extent of the actual bar line. This may differ from `Y-extent` because it does not include the dots in a repeat bar line.

`before-line-breaking` (boolean):
`ly:span-bar::before-line-breaking`

Dummy property, used to trigger a callback function.

`break-align-symbol` (symbol):
`'staff-bar`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`glyph-name` (string):
`ly:span-bar::calc-glyph-name`

The glyph name within the font.

In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`layer` (integer):
`0`
An integer which determines the order of printing objects. 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.

`non-musical` (boolean):
`#t`
True if the grob belongs to a `NonMusicalPaperColumn`.

`stencil` (stencil):
`ly:span-bar::print`
The symbol to print.

`X-extent` (pair of numbers):
`ly:span-bar::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`'(+inf.0 . -inf.0)`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `bar-line-interface` (page 527), `font-interface` (page 543), `grob-interface` (page 548), and `span-bar-interface` (page 587).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.113 SpanBarStub

`SpanBarStub` objects are created by: `Span_bar_stub_engraver` (page 323).

Standard settings:

`extra-spacing-height` (pair of numbers):
`pure-from-neighbor-interface::extra-spacing-height`
In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`X-extent` (pair of numbers):
`#<procedure #f (grob)>`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #f #<procedure pure-from-neighbor-interface::pure-height (grob beg end)> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), and `pure-from-neighbor-interface` (page 576).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.114 StaffGrouper

`StaffGrouper` objects are not created by any engraver.

Standard settings:

`staff-staff-spacing` (list):

```
'((basic-distance . 9)
  (minimum-distance . 7)
  (padding . 1)
  (stretchability . 5))
```

When applied to a staff-group's `StaffGrouper` grob, this spacing alist controls the distance between consecutive staves within the staff-group. When applied to a staff's `VerticalAxisGroup` grob, it controls the distance between the staff and the nearest staff below it in the same system, replacing any settings inherited from the `StaffGrouper` grob of the containing staff-group, if there is one. This property remains in effect even when non-staff lines appear between staves. The alist can contain the following 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.
- `minimum-distance` – the smallest allowable vertical distance, measured in staff-spaces, between the reference points of the two items, when compressing is in effect.
- `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).

`staffgroup-staff-spacing` (list):

```
'((basic-distance . 10.5)
  (minimum-distance . 8)
  (padding . 1)
  (stretchability . 9))
```

The spacing alist controlling 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 exist between the two staves. If the `staff-staff-spacing` property of the staff's `VerticalAxisGroup` grob is set, that is used instead. See `staff-staff-spacing` for a description of the alist structure.

This object supports the following interface(s): `grob-interface` (page 548), and `staff-grouper-interface` (page 588).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.115 StaffSpacing

`StaffSpacing` objects are created by: `Separating_line_group_engraver` (page 321).

Standard settings:

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`stem-spacing-correction` (number):
`0.4`

Optical correction amount for stems that are placed in tight configurations. For opposite directions, this amount is the correction for two normal sized stems that overlap completely.

This object supports the following interface(s): `grob-interface` (page 548), `spacing-interface` (page 586), and `staff-spacing-interface` (page 589).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.116 StaffSymbol

`StaffSymbol` objects are created by: `Staff_symbol_engraver` (page 324), and `Tab_staff_symbol_engraver` (page 326).

Standard settings:

`break-align-symbols` (list):
`'(staff-bar break-alignment)`

A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`layer` (integer):
`0`

An integer which determines the order of printing objects. 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.

`ledger-line-thickness` (pair of numbers):
`'(1.0 . 0.1)`

The thickness of ledger lines. It is the sum of 2 numbers: The first is the factor for line thickness, and the second for staff space. Both contributions are added.

`line-count` (integer):
`5`

The number of staff lines.

`stencil` (stencil):
`ly:staff-symbol::print`

The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol::height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), and `staff-symbol-interface` (page 589).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.117 StanzaNumber

`StanzaNumber` objects are created by: `Stanza_number_engraver` (page 324).

Standard settings:

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-series` (symbol):
`'bold`

Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`padding` (dimension, in staff space):
`1.0`

Add this much extra space between objects that are next to each other.

`side-axis` (number):
`0`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`stencil` (stencil):
`ly:text-interface::print`

The symbol to print.

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `side-position-interface` (page 581), `stanza-number-interface` (page 590), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.118 Stem

`Stem` objects are created by: `Span_stem_engraver` (page 323), and `Stem_engraver` (page 324).

Standard settings:

`beamlet-default-length` (pair):
`'(1.1 . 1.1)`

A pair of numbers. The first number specifies the default length of a beamlet that sticks out of the left hand side of this stem; the second number specifies the default length of the beamlet to the right. The actual length of a beamlet is determined by taking either the default length or the length specified by `beamlet-max-length-proportion`, whichever is smaller.

`beamlet-max-length-proportion` (pair):
`'(0.75 . 0.75)`

The maximum length of a beamlet, as a proportion of the distance between two adjacent stems.

`default-direction` (direction):
`ly:stem::calc-default-direction`

Direction determined by note head positions.

`details` (list):
```
'((lengths 3.5 3.5 3.5 4.25 5.0 6.0 7.0 8.0 9.0)
  (beamed-lengths 3.26 3.5 3.6)
  (beamed-minimum-free-lengths 1.83 1.5 1.25)
  (beamed-extreme-minimum-free-lengths 2.0 1.25)
  (stem-shorten 1.0 0.5 0.25))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`ly:stem::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`double-stem-separation` (number):
`0.5`

The distance between the two stems of a half note in tablature when using `\tabFullNotation`, not counting the width of the stems themselves, expressed as a multiple of the default height of a staff-space in the traditional five-line staff.

`duration-log` (integer):
`stem::calc-duration-log`

The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`length` (dimension, in staff space):
`#<unpure-pure-container #<primitive-procedure ly:stem::calc-length> #<primitive-procedure ly:stem::pure-calc-length> >`

User override for the stem length of unbeamed stems (each unit represents half a `staff-space`).

`neutral-direction` (direction):
`-1`

Which direction to take in the center of the staff.

`note-collision-threshold` (dimension, in staff space):
`1`

Simultaneous notes that are this close or closer in units of `staff-space` will be identified as vertically colliding. Used by `Stem` grobs for notes in the same voice, and `NoteCollision` grobs for notes in different voices. Default value 1.

`stem-begin-position` (number):
`#<unpure-pure-container #<primitive-procedure ly:stem::calc-stem-begin-position> #<primitive-procedure ly:stem::pure-calc-stem-begin-position> >`

User override for the begin position of a stem.

`stencil` (stencil):
`ly:stem::print`

The symbol to print.

`thickness` (number):
`1.3`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`X-extent` (pair of numbers):
`ly:stem::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number):
`ly:stem::offset-callback`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:stem::height> #<primitive-procedure ly:stem::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `grob-interface` (page 548), and `stem-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.119 StemStub

`StemStub` objects are created by: `Stem_engraver` (page 324).

Standard settings:

`extra-spacing-height` (pair of numbers):
`stem-stub::extra-spacing-height`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent

the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`X-extent` (pair of numbers):
`stem-stub::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #f #<procedure stem-stub::pure-height (grob beg end)> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.120 StemTremolo

`StemTremolo` objects are created by: `Stem_engraver` (page 324).

Standard settings:

`beam-thickness` (dimension, in staff space):
`0.48`
Beam thickness, measured in `staff-space` units.

`beam-width` (dimension, in staff space):
`ly:stem-tremolo::calc-width`
Width of the tremolo sign.

`direction` (direction):
`ly:stem-tremolo::calc-direction`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`parent-alignment-X` (number):
`0`
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`shape` (symbol):
`ly:stem-tremolo::calc-shape`
This setting determines what shape a grob has. Valid choices depend on the `stencil` callback reading this property.

`slope` (number):
`ly:stem-tremolo::calc-slope`
The slope of this object.

`stencil` (stencil):
`ly:stem-tremolo::print`
The symbol to print.

`X-extent` (pair of numbers):
`ly:stem-tremolo::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> #<primitive-procedure ly:stem-tremolo::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:stem-tremolo::calc-y-offset> #<primitive-procedure ly:stem-tremolo::pure-calc-y-offset> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `grob-interface` (page 548), `self-alignment-interface` (page 579), and `stem-tremolo-interface` (page 593).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.121 StringNumber

`StringNumber` objects are created by: `New_fingering_engraver` (page 312).

Standard settings:

`add-stem-support` (boolean):
`only-if-beamed`
If set, the `Stem` object is included in this script's support.

`avoid-slur` (symbol):
`'around`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`font-encoding` (symbol):
`'fetaText`
The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-size` (number):
`-5`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps

are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`number-type` (symbol):
`'arabic`
Numbering style. Choices include `roman-lower`, `roman-upper` and `arabic`.

`padding` (dimension, in staff space):
`0.5`
Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):
`0`
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`script-priority` (number):
`100`
A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`self-alignment-X` (number):
`0`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number):
`0`
Like `self-alignment-X` but for the Y axis.

`staff-padding` (dimension, in staff space):
`0.5`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`print-circled-text-callback`
The symbol to print.

`text` (markup):
`string-number::calc-text`
Text markup. See Section "Formatting text" in *Notation Reference*.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `number-interface` (page 570), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface`

(page 581), `string-number-interface` (page 593), `text-interface` (page 596), and `text-script-interface` (page 597).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.122 StrokeFinger

`StrokeFinger` objects are created by: `New_fingering_engraver` (page 312).

Standard settings:

`add-stem-support` (boolean):
`only-if-beamed`

If set, the `Stem` object is included in this script's support.

`digit-names` (vector):
`#("p" "i" "m" "a" "x")`

Names for string finger digits.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number):
`-4`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.5`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number):
`0`

Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`script-priority` (number):
`100`

A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`self-alignment-X` (number):
`0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number):
`0`

Like `self-alignment-X` but for the Y axis.

`staff-padding` (dimension, in staff space):
`0.5`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`text` (markup):
`stroke-finger::calc-text`
Text markup. See Section "Formatting text" in *Notation Reference*.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), `stroke-finger-interface` (page 593), `text-interface` (page 596), and `text-script-interface` (page 597).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.123 SustainPedal

`SustainPedal` objects are created by: `Piano_pedal_engraver` (page 317).

Standard settings:

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`
In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`padding` (dimension, in staff space):
`0.0`
Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number)
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
`0`
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
`ly:sustain-pedal::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
: `#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

  Two skylines, one above and one below this grob.

`X-offset` (number):
: `ly:self-alignment-interface::aligned-on-x-parent`

  The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

  Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `piano-pedal-interface` (page 575), `piano-pedal-script-interface` (page 576), `self-alignment-interface` (page 579), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.124 SustainPedalLineSpanner

`SustainPedalLineSpanner` objects are created by: `Piano_pedal_align_engraver` (page 316).

Standard settings:

`axes` (list):
: `'(1)`

  List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
: `-1`

  If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`minimum-space` (dimension, in staff space):
: `1.0`

  Minimum distance that the victim should move (after padding).

`outside-staff-priority` (number):
: `1000`

  If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
: `1.2`

  Add this much extra space between objects that are next to each other.

`side-axis` (number):
: `1`

  If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`1.2`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-element-stencils> #<primitive-procedure ly:grob::pure-vertical-skylines-from-element-stencils> >`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), `outside-staff-interface` (page 572), `piano-pedal-interface` (page 575), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.125 System

`System` objects are not created by any engraver.

Standard settings:

`axes` (list):
`'(0 1)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`outside-staff-placement-directive` (symbol):
`'left-to-right-polite`
One of four directives telling how outside staff objects should be placed.

- `left-to-right-greedy` – Place each successive grob from left to right.
- `left-to-right-polite` – Place a grob from left to right only if it does not potentially overlap with another grob that has been placed on a pass through a grob array. If there is overlap, do another pass to determine placement.
- `right-to-left-greedy` – Same as `left-to-right-greedy`, but from right to left.

- `right-to-left-polite` – Same as `left-to-right-polite`, but from right to left.

`skyline-horizontal-padding` (number):
`1.0`

For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`vertical-skylines` (pair of skylines):
`ly:axis-group-interface::calc-skylines`

Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:system::height> #<primitive-procedure ly:system::calc-pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), and `outside-staff-axis-group-interface` (page 571).

This object is of class `system-interface,System` (page ⟨undefined⟩).

3.1.126 SystemStartBar

`SystemStartBar` objects are created by: `System_start_delimiter_engraver` (page 325).

Standard settings:

`collapse-height` (dimension, in staff space):
`5.0`

Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`padding` (dimension, in staff space):
`-0.1`

Add this much extra space between objects that are next to each other.

`stencil` (stencil):
`ly:system-start-delimiter::print`

The symbol to print.

`style` (symbol):
`'bar-line`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number):
`1.6`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`

The horizontal amount that this object is moved relative to its X-parent.

This object supports the following interface(s): `grob-interface` (page 548), `side-position-interface` (page 581), and `system-start-delimiter-interface` (page 595).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.127 SystemStartBrace

`SystemStartBrace` objects are created by: `System_start_delimiter_engraver` (page 325).

Standard settings:

`collapse-height` (dimension, in staff space):
`5.0`

Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-encoding` (symbol):
`'fetaBraces`

The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`padding` (dimension, in staff space):
`0.3`

Add this much extra space between objects that are next to each other.

`stencil` (stencil):
`ly:system-start-delimiter::print`

The symbol to print.

`style` (symbol):
`'brace`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`
The horizontal amount that this object is moved relative to its X-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `side-position-interface` (page 581), and `system-start-delimiter-interface` (page 595).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.128 SystemStartBracket

`SystemStartBracket` objects are created by: `System_start_delimiter_engraver` (page 325).

Standard settings:

`collapse-height` (dimension, in staff space):
`5.0`
Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

`direction` (direction):
`-1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`padding` (dimension, in staff space):
`0.8`
Add this much extra space between objects that are next to each other.

`stencil` (stencil):
`ly:system-start-delimiter::print`
The symbol to print.

`style` (symbol):
`'bracket`
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number):
`0.45`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`
The horizontal amount that this object is moved relative to its X-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `side-position-interface` (page 581), and `system-start-delimiter-interface` (page 595).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.129 SystemStartSquare

SystemStartSquare objects are created by: System_start_delimiter_engraver (page 325).

Standard settings:

collapse-height (dimension, in staff space):
5.0

Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

direction (direction):
-1

If side-axis is 0 (or X), then this property determines whether the object is placed LEFT, CENTER or RIGHT with respect to the other object. Otherwise, it determines whether the object is placed UP, CENTER or DOWN. Numerical values may also be used: UP=1, DOWN=-1, LEFT=-1, RIGHT=1, CENTER=0.

stencil (stencil):
ly:system-start-delimiter::print

The symbol to print.

style (symbol):
'line-bracket

This setting determines in what style a grob is typeset. Valid choices depend on the stencil callback reading this property.

thickness (number):
1.0

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

X-offset (number):
ly:side-position-interface::x-aligned-side

The horizontal amount that this object is moved relative to its X-parent.

This object supports the following interface(s): font-interface (page 543), grob-interface (page 548), side-position-interface (page 581), and system-start-delimiter-interface (page 595).

This object is of class spanner-interface,Spanner (page ⟨undefined⟩).

3.1.130 TabNoteHead

TabNoteHead objects are created by: Tab_note_heads_engraver (page 325).

Standard settings:

bend-me (boolean):
'()

Decide whether this grob is bent.

details (list):
```
'((cautionary-properties
    (angularity . 0.4)
    (half-thickness . 0.075)
```

```
    (padding . 0)
    (procedure
      .
      #<procedure parenthesize-stencil (stencil half-thickness width angular
    (width . 0.25))
  (head-offset . 3/5)
  (harmonic-properties
    (angularity . 2)
    (half-thickness . 0.075)
    (padding . 0)
    (procedure
      .
      #<procedure parenthesize-stencil (stencil half-thickness width angular
    (width . 0.25))
  (repeat-tied-properties
    (note-head-visible . #t)
    (parenthesize . #t))
  (tied-properties (parenthesize . #t)))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction):
`0`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`duration-log` (integer):
`note-head::calc-duration-log`

The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`font-series` (symbol):
`'bold`

Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-size` (number):
`-2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`parenthesis-friends` (list):
`'(dot)`

A list of Grob types, as symbols. When parentheses enclose a Grob that has 'parenthesis-friends, the parentheses widen to include any child Grobs with type among 'parenthesis-friends.

`stem-attachment` (pair of numbers):
`ly:note-head::calc-tab-stem-attachment`

An `(`*x* `.` *y*`)` pair where the stem attaches to the notehead.

`stencil` (stencil):
`tab-note-head::print`
The symbol to print.

`whiteout` (boolean-or-number):
`#t`
If a number or true, the grob is printed over a white background to white-out underlying material, if the grob is visible. A number indicates how far the white background extends beyond the bounding box of the grob as a multiple of the staff-line thickness. The `LyricHyphen` grob uses a special implementation of whiteout: A positive number indicates how far the white background extends beyond the bounding box in multiples of `line-thickness`. The shape of the background is determined by `whiteout-style`. Usually `#f` by default.

`X-offset` (number):
`ly:self-alignment-interface::x-aligned-on-self`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `bend-interface` (page 531), `font-interface` (page 543), `grob-interface` (page 548), `note-head-interface` (page 569), `rhythmic-grob-interface` (page 577), `rhythmic-head-interface` (page 577), `staff-symbol-referencer-interface` (page 590), `tab-note-head-interface` (page 596), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.131 TextScript

`TextScript` objects are created by: `Text_engraver` (page 327).

Standard settings:

`avoid-slur` (symbol):
`'around`
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`direction` (direction):
`-1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise,

it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`outside-staff-horizontal-padding` (number):
`0.2`

By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-priority` (number):
`450`

If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.3`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number)
Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`script-priority` (number):
`200`

A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`self-alignment-X` (number)
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`side-axis` (number):
`1`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`slur-padding` (number):
`0.5`

Extra distance between slur and script.

`staff-padding` (dimension, in staff space):
`0.5`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:text-interface::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`
Two skylines, one above and one below this grob.

`X-align-on-main-noteheads` (boolean):
`#t`
If true, this grob will ignore suspended noteheads when aligning itself on NoteColumn.

`X-offset` (number):
`ly:self-alignment-interface::aligned-on-x-parent`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `instrument-specific-markup-interface` (page 555), `outside-staff-interface` (page 572), `self-alignment-interface` (page 579), `side-position-interface` (page 581), `text-interface` (page 596), and `text-script-interface` (page 597).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.132 TextSpanner

`TextSpanner` objects are created by: `Text_spanner_engraver` (page 327).

Standard settings:

`bound-details` (list):
```
'((left (Y . 0) (padding . 0.25) (attach-dir . -1))
  (left-broken (attach-dir . 1))
  (right (Y . 0) (padding . 0.25)))
```
An alist of properties for determining attachments of spanners to edges.

`dash-fraction` (number):
`0.2`
Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`dash-period` (number):
`3.0`
The length of one dash together with whitespace. If negative, no line is drawn at all.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-shape` (symbol):
`'italic`
Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`left-bound-info` (list):
`ly:line-spanner::calc-left-bound-info`
An alist of properties for determining attachments of spanners to edges.

`outside-staff-priority` (number):
`350`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`right-bound-info` (list):
`ly:line-spanner::calc-right-bound-info`
An alist of properties for determining attachments of spanners to edges.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`0.8`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:line-spanner::print`
The symbol to print.

`style` (symbol):
`'dashed-line`
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), `outside-staff-interface` (page 572), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.133 Tie

Tie objects are created by: Completion_heads_engraver (page 292), and Tie_engraver (page 327).

Standard settings:

avoid-slur (symbol):
'inside

Method of handling slur collisions. Choices are inside, outside, around, and ignore. inside adjusts the slur if needed to keep the grob inside the slur. outside moves the grob vertically to the outside of the slur. around moves the grob vertically to the outside of the slur only if there is a collision. ignore does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), outside and around behave like ignore.

control-points (list of number pairs):
ly:tie::calc-control-points

List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

details (list):

```
'((ratio . 0.333)
  (center-staff-line-clearance . 0.6)
  (tip-staff-line-clearance . 0.45)
  (note-head-gap . 0.2)
  (stem-gap . 0.35)
  (height-limit . 1.0)
  (horizontal-distance-penalty-factor . 10)
  (same-dir-as-stem-penalty . 8)
  (min-length-penalty-factor . 26)
  (tie-tie-collision-distance . 0.45)
  (tie-tie-collision-penalty . 25.0)
  (intra-space-threshold . 1.25)
  (outer-tie-vertical-distance-symmetry-penalty-factor
    .
    10)
  (outer-tie-length-symmetry-penalty-factor . 10)
  (vertical-distance-penalty-factor . 7)
  (outer-tie-vertical-gap . 0.25)
  (multi-tie-region-size . 3)
  (single-tie-region-size . 4)
  (between-length-limit . 1.0))
```

Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a details property.

direction (direction):
ly:tie::calc-direction

If side-axis is 0 (or X), then this property determines whether the object is placed LEFT, CENTER or RIGHT with respect to the other object. Otherwise, it determines whether the object is placed UP, CENTER or DOWN. Numerical values may also be used: UP=1, DOWN=-1, LEFT=-1, RIGHT=1, CENTER=0.

`line-thickness` (number):
`0.8`
For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`neutral-direction` (direction):
`1`
Which direction to take in the center of the staff.

`springs-and-rods` (boolean):
`ly:spanner::set-spacing-rods`
Dummy variable for triggering spacing routines.

`stencil` (stencil):
`ly:tie::print`
The symbol to print.

`thickness` (number):
`1.2`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

This object supports the following interface(s): `bezier-curve-interface` (page 532), `grob-interface` (page 548), and `tie-interface` (page 598).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.134 TieColumn

`TieColumn` objects are created by: `Completion_heads_engraver` (page 292), and `Tie_engraver` (page 327).

Standard settings:

`before-line-breaking` (boolean):
`ly:tie-column::before-line-breaking`
Dummy property, used to trigger a callback function.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), and `tie-column-interface` (page 597).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.135 TimeSignature

`TimeSignature` objects are created by: `Time_signature_engraver` (page 328).

Standard settings:

`avoid-slur` (symbol):
`'inside`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`break-align-anchor` (number):
`ly:break-aligned-interface::calc-extent-aligned-anchor`

Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-anchor-alignment` (number):
`-1`

Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's extent.

`break-align-symbol` (symbol):
`'time-signature`

This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-visibility` (vector):
`#(#t #t #t)`

A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers):
`pure-from-neighbor-interface::extra-spacing-height-including-staff`

In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers):
`'(0.0 . 0.8)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`non-musical` (boolean):
`#t`

True if the grob belongs to a `NonMusicalPaperColumn`.

`space-alist` (list):

```
'((ambitus extra-space . 1.0)
  (cue-clef extra-space . 1.5)
  (first-note fixed-space . 2.0)
  (right-edge extra-space . 0.5)
  (staff-bar extra-space . 1.0))
```

An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`stencil` (stencil):
`ly:time-signature::print`

The symbol to print.

`style` (symbol):
`'C`

This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `break-aligned-interface` (page 533), `font-interface` (page 543), `grob-interface` (page 548), `pure-from-neighbor-interface` (page 576), and `time-signature-interface` (page 600).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.136 TrillPitchAccidental

`TrillPitchAccidental` objects are created by: `Pitched_trill_engraver` (page 318).

Standard settings:

`direction` (direction):
`-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-size` (number):
`-4`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.2`

Add this much extra space between objects that are next to each other.

`side-axis` (number):
`0`

If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`stencil` (stencil):
`ly:accidental-interface::print`

The symbol to print.

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:accidental-interface::height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `accidental-interface` (page 521), `accidental-switch-interface` (page 522), `font-interface` (page 543), `grob-interface` (page 548), `inline-accidental-interface` (page 555), `side-position-interface` (page 581), and `trill-pitch-accidental-interface` (page 601).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.137 TrillPitchGroup

`TrillPitchGroup` objects are created by: `Pitched_trill_engraver` (page 318).

Standard settings:

`axes` (list):
`'(0)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`horizon-padding` (number):
`0.1`
The amount to pad the axis along which a `Skyline` is built for the `side-position-interface`.

`minimum-space` (dimension, in staff space):
`2.5`
Minimum distance that the victim should move (after padding).

`padding` (dimension, in staff space):
`0.3`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`0`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number):
`ly:side-position-interface::x-aligned-side`
The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<procedure trill-pitch-group::pure-height (grob start end)> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), and `side-position-interface` (page 581).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.138 TrillPitchHead

`TrillPitchHead` objects are created by: `Pitched_trill_engraver` (page 318).

Standard settings:

`duration-log` (integer):
`2`
The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`font-size` (number):
`-4`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`parenthesis-friends` (list):
`'(accidental-grob)`
A list of Grob types, as symbols. When parentheses enclose a Grob that has 'parenthesis-friends, the parentheses widen to include any child Grobs with type among 'parenthesis-friends.

`stencil` (stencil):
`ly:note-head::print`
The symbol to print.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:staff-symbol-referencer::callback> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `ledgered-interface` (page 559), `pitched-trill-interface` (page 576), `rhythmic-head-interface` (page 577), and `staff-symbol-referencer-interface` (page 590).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.139 TrillPitchParentheses

`TrillPitchParentheses` objects are not created by any engraver.

Standard settings:

`font-size` (number):
`-4`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`padding` (dimension, in staff space):
`0.3`

Add this much extra space between objects that are next to each other.

`stencil` (stencil):
`parentheses-interface::print`

The symbol to print.

`stencils` (list):
`parentheses-interface::calc-parenthesis-stencils`

Multiple stencils, used as intermediate value.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `parentheses-interface` (page 574), and `pitched-trill-interface` (page 576).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.140 TrillSpanner

`TrillSpanner` objects are created by: `Trill_spanner_engraver` (page 330).

Standard settings:

`after-line-breaking` (boolean):
`ly:spanner::kill-zero-spanned-time`

Dummy property, used to trigger callback for `after-line-breaking`.

`bound-details` (list):

```
'((left (text #<procedure musicglyph-markup (layout props glyph-name)>
                "scripts.trill")
        (Y . 0)
        (stencil-offset -0.5 . -1)
        (padding . 0.5)
        (attach-dir . 0))
  (left-broken (end-on-note . #t))
  (right (Y . 0)))
```

An alist of properties for determining attachments of spanners to edges.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`left-bound-info` (list):
`ly:line-spanner::calc-left-bound-info`
An alist of properties for determining attachments of spanners to edges.

`outside-staff-priority` (number):
`50`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`0.5`
Add this much extra space between objects that are next to each other.

`right-bound-info` (list):
`ly:line-spanner::calc-right-bound-info`
An alist of properties for determining attachments of spanners to edges.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`1.0`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:line-spanner::print`
The symbol to print.

`style` (symbol):
`'trill`
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), `outside-staff-interface` (page 572), `side-position-interface` (page 581), and `trill-spanner-interface` (page 601).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.141 TupletBracket

`TupletBracket` objects are created by: `Tuplet_engraver` (page 330).

Standard settings:

`avoid-scripts` (boolean):
`#t`

If set, a tuplet bracket avoids the scripts associated with the note heads it encompasses.

`connect-to-neighbor` (pair):
`ly:tuplet-bracket::calc-connect-to-neighbors`

Pair of booleans, indicating whether this grob looks as a continued break.

`direction` (direction):
`ly:tuplet-bracket::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair):
`'(0.7 . 0.7)`

A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`full-length-to-extent` (boolean):
`#t`

Run to the extent of the column for a full-length tuplet bracket.

`padding` (dimension, in staff space):
`1.1`

Add this much extra space between objects that are next to each other.

`positions` (pair of numbers):
`ly:tuplet-bracket::calc-positions`

Pair of staff coordinates `(`*`start . end`*`)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`shorten-pair` (pair of numbers):
`'(-0.2 . -0.2)`

The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`staff-padding` (dimension, in staff space):
`0.25`

Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`stencil` (stencil):
`ly:tuplet-bracket::print`

The symbol to print.

`thickness` (number):
`1.6`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*`.StaffSymbol.thickness`).

`tuplet-slur` (boolean)
Draw a slur instead of a bracket for tuplets.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`

Two skylines, one above and one below this grob.

`X-positions` (pair of numbers):
`ly:tuplet-bracket::calc-x-positions`

Pair of X staff coordinates of a spanner in the form `(`*left* `.` *right*`)`, where both *left* and *right* are in `staff-space` units of the current staff.

This object supports the following interface(s): `grob-interface` (page 548), `line-interface` (page 560), `outside-staff-interface` (page 572), and `tuplet-bracket-interface` (page 601).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.142 TupletNumber

`TupletNumber` objects are created by: `Tuplet_engraver` (page 330).

Standard settings:

`avoid-slur` (symbol):
`'inside`

Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`direction` (direction):
`tuplet-number::calc-direction`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number):
`-2`

The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`knee-to-beam` (boolean):
`#t`

Determines whether a tuplet number will be positioned next to a kneed beam.

`stencil` (stencil):
`ly:tuplet-number::print`

The symbol to print.

`text` (markup):
`tuplet-number::calc-denominator-text`

Text markup. See Section "Formatting text" in *Notation Reference*.

`X-offset` (number):
`ly:tuplet-number::calc-x-offset`

The horizontal amount that this object is moved relative to its X-parent.

`Y-offset` (number):
`ly:tuplet-number::calc-y-offset`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `outside-staff-interface` (page 572), `text-interface` (page 596), and `tuplet-number-interface` (page 603).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.143 UnaCordaPedal

`UnaCordaPedal` objects are created by: `Piano_pedal_engraver` (page 317).

Standard settings:

`direction` (direction):
`1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`extra-spacing-width` (pair of numbers):
`'(+inf.0 . -inf.0)`

In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`font-shape` (symbol):
`'italic`

Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`padding` (dimension, in staff space):
`0.0`

Add this much extra space between objects that are next to each other.

`parent-alignment-X` (number)
: Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`self-alignment-X` (number):
: `0`

Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`stencil` (stencil):
: `ly:text-interface::print`

The symbol to print.

`vertical-skylines` (pair of skylines):
: `#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> >`

Two skylines, one above and one below this grob.

`X-offset` (number):
: `ly:self-alignment-interface::aligned-on-x-parent`

The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers):
: `#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `piano-pedal-script-interface` (page 576), `self-alignment-interface` (page 579), and `text-interface` (page 596).

This object is of class `item-interface,Item` (page ⟨undefined⟩).

3.1.144 UnaCordaPedalLineSpanner

`UnaCordaPedalLineSpanner` objects are created by: `Piano_pedal_align_engraver` (page 316).

Standard settings:

`axes` (list):
: `'(1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
: `-1`

If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`minimum-space` (dimension, in staff space):
: `1.0`

Minimum distance that the victim should move (after padding).

`outside-staff-priority` (number):
`1000`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`1.2`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`staff-padding` (dimension, in staff space):
`1.2`
Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-element-stencils> #<primitive-procedure ly:grob::pure-vertical-skylines-from-element-stencils> >`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), `outside-staff-interface` (page 572), `piano-pedal-interface` (page 575), and `side-position-interface` (page 581).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.145 VaticanaLigature

`VaticanaLigature` objects are created by: `Vaticana_ligature_engraver` (page 331).

Standard settings:

`stencil` (stencil):
`ly:vaticana-ligature::print`
The symbol to print.

`thickness` (number):
`0.6`

For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), and `vaticana-ligature-interface` (page 603).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.146 VerticalAlignment

`VerticalAlignment` objects are created by: `Vertical_align_engraver` (page 331).

Standard settings:

`axes` (list):
`'(1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`stacking-dir` (direction):
`-1`

Stack objects in which direction?

`vertical-skylines` (pair of skylines):
`ly:axis-group-interface::combine-skylines`

Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`

Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`

Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `align-interface` (page 522), `axis-group-interface` (page 524), and `grob-interface` (page 548).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.147 VerticalAxisGroup

`VerticalAxisGroup` objects are created by: `Axis_group_engraver` (page 285).

Standard settings:

`axes` (list):
`'(1)`

List of axis numbers. In the case of alignment grobs, this should contain only one number.

`default-staff-staff-spacing` (list):

```
'((basic-distance . 9)
  (minimum-distance . 8)
  (padding . 1))
```

The settings to use for `staff-staff-spacing` when it is unset, for ungrouped staves and for grouped staves that do not have the relevant `StaffGrouper` property set (`staff-staff-spacing` or `staffgroup-staff-spacing`).

`nonstaff-unrelatedstaff-spacing` (list):

```
'((padding . 0.5))
```

The spacing alist controlling the distance between the current non-staff line and the nearest 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`. See `staff-staff-spacing` for a description of the alist structure.

`outside-staff-placement-directive` (symbol):

```
'left-to-right-polite
```

One of four directives telling how outside staff objects should be placed.

- `left-to-right-greedy` – Place each successive grob from left to right.
- `left-to-right-polite` – Place a grob from left to right only if it does not potentially overlap with another grob that has been placed on a pass through a grob array. If there is overlap, do another pass to determine placement.
- `right-to-left-greedy` – Same as `left-to-right-greedy`, but from right to left.
- `right-to-left-polite` – Same as `left-to-right-polite`, but from right to left.

`skyline-horizontal-padding` (number):

```
0.1
```

For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`staff-staff-spacing` (list):

```
#<unpure-pure-container #<primitive-procedure ly:axis-group-
interface::calc-staff-staff-spacing> #<primitive-procedure
ly:axis-group-interface::calc-pure-staff-staff-spacing> >
```

When applied to a staff-group's `StaffGrouper` grob, this spacing alist controls the distance between consecutive staves within the staff-group. When applied to a staff's `VerticalAxisGroup` grob, it controls the distance between the staff and the nearest staff below it in the same system, replacing any settings inherited from the `StaffGrouper` grob of the containing staff-group, if there is one. This property remains in effect even when non-staff lines appear between staves. The alist can contain the following 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.
- `minimum-distance` – the smallest allowable vertical distance, measured in staff-spaces, between the reference points of the two items, when compressing is in effect.

- `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).

`stencil` (stencil):
`ly:axis-group-interface::print`
The symbol to print.

`vertical-skylines` (pair of skylines):
`ly:hara-kiri-group-spanner::calc-skylines`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:hara-kiri-group-spanner::y-extent> #<primitive-procedure ly:hara-kiri-group-spanner::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`ly:hara-kiri-group-spanner::force-hara-kiri-callback`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), `hara-kiri-group-spanner-interface` (page 553), and `outside-staff-axis-group-interface` (page 571).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.148 VoiceFollower

`VoiceFollower` objects are created by: `Note_head_line_engraver` (page 312).

Standard settings:

`after-line-breaking` (boolean):
`ly:spanner::kill-zero-spanned-time`
Dummy property, used to trigger callback for `after-line-breaking`.

`bound-details` (list):
```
'((right (attach-dir . 0) (padding . 1.5))
  (left (attach-dir . 0) (padding . 1.5)))
```
An alist of properties for determining attachments of spanners to edges.

`gap` (dimension, in staff space):
`0.5`
Size of a gap in a variable symbol.

`left-bound-info` (list):
`ly:line-spanner::calc-left-bound-info`
An alist of properties for determining attachments of spanners to edges.

`normalized-endpoints` (pair):
`ly:spanner::calc-normalized-endpoints`
Represents left and right placement over the total spanner, where the width of the spanner is normalized between 0 and 1.

`right-bound-info` (list):
`ly:line-spanner::calc-right-bound-info`
An alist of properties for determining attachments of spanners to edges.

`stencil` (stencil):
`ly:line-spanner::print`
The symbol to print.

`style` (symbol):
`'line`
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`X-extent` (pair of numbers)
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers)
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `grob-interface` (page 548), `line-interface` (page 560), and `line-spanner-interface` (page 561).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.149 VoltaBracket

`VoltaBracket` objects are created by: `Volta_engraver` (page 331).

Standard settings:

`baseline-skip` (dimension, in staff space):
`1.7`
Distance between base lines of multiple lines of text.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair):
`'(2.0 . 2.0)`
A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`font-encoding` (symbol):
`'fetaText`
The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-size` (number):
`-4`
The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`shorten-pair` (pair of numbers):
`ly:volta-bracket::calc-shorten-pair`
The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`stencil` (stencil):
`ly:volta-bracket-interface::print`
The symbol to print.

`thickness` (number):
`1.6`
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >`
Two skylines, one above and one below this grob.

`word-space` (dimension, in staff space):
`0.6`
Space to insert between words in texts.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:grob::stencil-height> #<procedure volta-bracket-interface::pure-height (grob start end)> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

This object supports the following interface(s): `font-interface` (page 543), `grob-interface` (page 548), `horizontal-bracket-interface` (page 554), `line-interface` (page 560), `side-position-interface` (page 581), `text-interface` (page 596), `volta-bracket-interface` (page 604), and `volta-interface` (page 604).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.150 VoltaBracketSpanner

`VoltaBracketSpanner` objects are created by: `Volta_engraver` (page 331).

Standard settings:

`after-line-breaking` (boolean):
`ly:side-position-interface::move-to-extremal-staff`
Dummy property, used to trigger callback for `after-line-breaking`.

`axes` (list):
`'(1)`
List of axis numbers. In the case of alignment grobs, this should contain only one number.

`direction` (direction):
`1`
If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`outside-staff-priority` (number):
`600`
If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`padding` (dimension, in staff space):
`1`
Add this much extra space between objects that are next to each other.

`side-axis` (number):
`1`
If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`vertical-skylines` (pair of skylines):
`#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-element-stencils> #<primitive-procedure ly:grob::pure-vertical-skylines-from-element-stencils> >`
Two skylines, one above and one below this grob.

`X-extent` (pair of numbers):
`ly:axis-group-interface::width`
Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`Y-extent` (pair of numbers):
`#<unpure-pure-container #<primitive-procedure ly:axis-group-interface::height> #<primitive-procedure ly:axis-group-interface::pure-height> >`
Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number):
`#<unpure-pure-container #<primitive-procedure ly:side-position-interface::y-aligned-side> #<primitive-procedure ly:side-position-interface::pure-y-aligned-side> >`
The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `axis-group-interface` (page 524), `grob-interface` (page 548), `outside-staff-interface` (page 572), `side-position-interface` (page 581), and `volta-interface` (page 604).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.1.151 VowelTransition

VowelTransition objects are created by: Hyphen_engraver (page 304).

Standard settings:

after-line-breaking (boolean):
ly:spanner::kill-zero-spanned-time
Dummy property, used to trigger callback for after-line-breaking.

arrow-length (number):
0.5
Arrow length.

arrow-width (number):
0.5
Arrow width.

bound-details (list):
```
'((left (Y . 0) (padding . 0.14) (attach-dir . 1))
  (right-broken (padding . 0))
  (left-broken (padding . 0))
  (right (Y . 0)
         (padding . 0.14)
         (attach-dir . -1)
         (arrow . #t)))
```
An alist of properties for determining attachments of spanners to edges.

left-bound-info (list):
ly:line-spanner::calc-left-bound-info
An alist of properties for determining attachments of spanners to edges.

minimum-length (dimension, in staff space):
1.0
Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the springs-and-rods property. If added to a Tie, this sets the minimum distance between noteheads.

right-bound-info (list):
ly:line-spanner::calc-right-bound-info
An alist of properties for determining attachments of spanners to edges.

springs-and-rods (boolean):
ly:vowel-transition::set-spacing-rods
Dummy variable for triggering spacing routines.

stencil (stencil):
ly:line-spanner::print
The symbol to print.

style (symbol):
'line
This setting determines in what style a grob is typeset. Valid choices depend on the stencil callback reading this property.

vertical-skylines (pair of skylines):
#<unpure-pure-container #<primitive-procedure ly:grob::vertical-skylines-from-stencil> #<primitive-procedure ly:grob::pure-simple-vertical-skylines-from-extents> >

Two skylines, one above and one below this grob.

`Y-offset` (number):
`0.5`

The vertical amount that this object is moved relative to its Y-parent.

This object supports the following interface(s): `grob-interface` (page 548), `line-interface` (page 560), `line-spanner-interface` (page 561), and `lyric-interface` (page 563).

This object is of class `spanner-interface,Spanner` (page ⟨undefined⟩).

3.2 Graphical Object Interfaces

3.2.1 `accidental-interface`

A single accidental.

User settable properties:

`alteration` (number)
Alteration numbers for accidental.

`alteration-glyph-name-alist` (list)
An alist of key-string pairs.

`avoid-slur` (symbol)
Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`hide-tied-accidental-after-break` (boolean)
If set, an accidental that appears on a tied note after a line break will not be displayed.

`restore-first` (boolean)
Print a natural before the accidental.

Internal properties:

`forced` (boolean)
Manually forced accidental.

`tie` (graphical (layout) object)
A pointer to a `Tie` object.

This grob interface is used in the following graphical object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalSuggestion` (page 350), `AmbitusAccidental` (page 354), and `TrillPitchAccidental` (page 504).

3.2.2 `accidental-placement-interface`

Resolve accidental collisions.

User settable properties:

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`right-padding` (dimension, in staff space)
: Space to insert on the right side of an object (e.g., between note and its accidentals).

`script-priority` (number)
: A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

Internal properties:

`accidental-grobs` (list)
: An alist with `(`*`notename . groblist`*`)` entries.

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): `AccidentalPlacement` (page 350).

3.2.3 accidental-suggestion-interface

An accidental, printed as a suggestion (typically: vertically over a note).

This grob interface is used in the following graphical object(s): `AccidentalSuggestion` (page 350).

3.2.4 accidental-switch-interface

Any object that prints one or several accidentals based on alterations.

User settable properties:

`alteration-glyph-name-alist` (list)
: An alist of key-string pairs.

This grob interface is used in the following graphical object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalSuggestion` (page 350), `AmbitusAccidental` (page 354), `BalloonText` (page 357), `BassFigure` (page 363), `ChordName` (page 377), `CombineTextScript` (page 383), `HorizontalBracketText` (page 419), `InstrumentName` (page 420), `InstrumentSwitch` (page 421), `KeyCancellation` (page 424), `KeySignature` (page 427), `MeasureSpanner` (page 442), `NoteName` (page 456), `RehearsalMark` (page 465), `TextScript` (page 496), and `TrillPitchAccidental` (page 504).

3.2.5 align-interface

Order grobs from top to bottom, left to right, right to left or bottom to top. For vertical alignments of staves, the `line-break-system-details` of the left Section "NonMusicalPaperColumn" in *Internals Reference* may be set to tune vertical spacing.

User settable properties:

`align-dir` (direction)
: Which side to align? `-1`: left side, `0`: around center of width, `1`: right side.

`axes` (list)
: List of axis numbers. In the case of alignment grobs, this should contain only one number.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`stacking-dir` (direction)
: Stack objects in which direction?

Internal properties:

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

`minimum-translations-alist` (list)
: An list of translations for a given start and end point.

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): `BassFigureAlignment` (page 363), and `VerticalAlignment` (page 514).

3.2.6 ambitus-interface

The line between note heads for a pitch range.

User settable properties:

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`maximum-gap` (number)
: Maximum value allowed for `gap` property.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`note-heads` (array of grobs)
: An array of note head grobs.

This grob interface is used in the following graphical object(s): `Ambitus` (page 352), `AmbitusLine` (page 354), and `AmbitusNoteHead` (page 355).

3.2.7 arpeggio-interface

Functions and settings for drawing an arpeggio symbol.

User settable properties:

`arpeggio-direction` (direction)
: If set, put an arrow on the arpeggio squiggly line.

`dash-definition` (pair)
: List of `dash-elements` defining the dash structure. Each `dash-element` has a starting t value, an ending t-value, a `dash-fraction`, and a `dash-period`.

`line-thickness` (number)
: For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`positions` (pair of numbers)
: Pair of staff coordinates (*`start . end`*), where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`protrusion` (number)
: In an arpeggio bracket, the length of the horizontal edges.

`script-priority` (number)
: A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`stems` (array of grobs)
: An array of stem objects.

This grob interface is used in the following graphical object(s): `Arpeggio` (page 355).

3.2.8 axis-group-interface

An object that groups other layout objects.

User settable properties:

`axes` (list) List of axis numbers. In the case of alignment grobs, this should contain only one number.

`default-staff-staff-spacing` (list)
: The settings to use for `staff-staff-spacing` when it is unset, for ungrouped staves and for grouped staves that do not have the relevant `StaffGrouper` property set (`staff-staff-spacing` or `staffgroup-staff-spacing`).

`nonstaff-nonstaff-spacing` (list)
: The spacing alist controlling 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`. See `staff-staff-spacing` for a description of the alist structure.

`nonstaff-relatedstaff-spacing` (list)
: The spacing alist controlling 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. See `staff-staff-spacing` for a description of the alist structure.

`nonstaff-unrelatedstaff-spacing` (list)
: The spacing alist controlling the distance between the current non-staff line and the nearest 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`. See `staff-staff-spacing` for a description of the alist structure.

`staff-affinity` (direction)
: 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. 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.

`staff-staff-spacing` (list)
: When applied to a staff-group's `StaffGrouper` grob, this spacing alist controls the distance between consecutive staves within the staff-group. When applied to a staff's `VerticalAxisGroup` grob, it controls the distance between the staff and the nearest staff below it in the same system, replacing any settings inherited from the `StaffGrouper` grob of the containing staff-group, if there is one. This property remains in effect even when non-staff lines appear between staves. The alist can contain the following 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.
  - `minimum-distance` – the smallest allowable vertical distance, measured in staff-spaces, between the reference points of the two items, when compressing is in effect.
  - `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).

Internal properties:

`adjacent-pure-heights` (pair)
: A pair of vectors. Used by a `VerticalAxisGroup` to cache the `Y-extent`s of different column ranges.

`bound-alignment-interfaces` (list)
: Interfaces to be used for positioning elements that align with a column.

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

`pure-relevant-grobs` (array of grobs)
: All the grobs (items and spanners) that are relevant for finding the `pure-Y-extent`

`pure-relevant-items` (array of grobs)
: A subset of elements that are relevant for finding the `pure-Y-extent`.

`pure-relevant-spanners` (array of grobs)
: A subset of elements that are relevant for finding the `pure-Y-extent`.

`pure-Y-common` (graphical (layout) object)
: A cache of the `common_refpoint_of_array` of the `elements` grob set.

`staff-grouper` (graphical (layout) object)
: The staff grouper we belong to.

`system-Y-offset` (number)
: The Y-offset (relative to the bottom of the top-margin of the page) of the system to which this staff belongs.

`X-common` (graphical (layout) object)
: Common reference point for axis group.

`Y-common` (graphical (layout) object)
: See `X-common`.

This grob interface is used in the following graphical object(s): `Ambitus` (page 352), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364), `BassFigureLine` (page 366), `BreakAlignGroup` (page 371), `BreakAlignment` (page 372), `CenteredBarNumberLineSpanner` (page 376), `DotColumn` (page 394), `DynamicLineSpanner` (page 401), `NonMusicalPaperColumn` (page 452), `NoteCollision` (page 453), `NoteColumn` (page 454), `PaperColumn` (page 458), `SostenutoPedalLineSpanner` (page 475), `SustainPedalLineSpanner` (page 489), `System` (page 490), `TrillPitchGroup` (page 505), `UnaCordaPedalLineSpanner` (page 512), `VerticalAlignment` (page 514), `VerticalAxisGroup` (page 514), and `VoltaBracketSpanner` (page 518).

3.2.9 `balloon-interface`

A collection of routines to put text balloons around an object.

User settable properties:

`annotation-balloon` (boolean)
: Print the balloon around an annotation.

`annotation-line` (boolean)
: Print the line from an annotation to the grob that it annotates.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`text` (markup)
: Text markup. See Section “Formatting text” in *Notation Reference*.

Internal properties:

`spanner-placement` (direction)
: The place of an annotation on a spanner. `LEFT` is for the first spanner, and `RIGHT` is for the last. `CENTER` will place it on the broken spanner that falls closest to the center of the length of the entire spanner, although this behavior is unpredictable in situations with lots of rhythmic diversity. For predictable results, use `LEFT` and `RIGHT`.

This grob interface is used in the following graphical object(s): `BalloonText` (page 357), and `Footnote` (page 411).

3.2.10 `bar-line-interface`

Print a special bar symbol. It replaces the regular bar symbol with a special symbol. The argument *bartype* is a string which specifies the kind of bar line to print.

The list of allowed glyphs and predefined bar lines can be found in `scm/bar-line.scm`.

`gap` is used for the gaps in dashed bar lines.

User settable properties:

`allow-span-bar` (boolean)
: If false, no inter-staff bar line will be created below this bar line.

`bar-extent` (pair of numbers)
: The Y-extent of the actual bar line. This may differ from `Y-extent` because it does not include the dots in a repeat bar line.

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`glyph` (string)
: A string determining what 'style' of glyph is typeset. Valid choices depend on the function that is reading this property.

  In combination with (span) bar lines, it is a string resembling the bar line appearance in ASCII form.

`glyph-name` (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`hair-thickness` (number)
: Thickness of the thin line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`kern` (dimension, in staff space)
: The space between individual elements in any compound bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`rounded` (boolean)
: Decide whether lines should be drawn rounded or not.

`segno-kern` (number)
: The space between the two thin lines of the segno bar line symbol, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`thick-thickness` (number)
: Thickness of the thick line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *Staff*`.StaffSymbol.thickness`).

Internal properties:

`has-span-bar` (pair)
: A pair of grobs containing the span bars to be drawn below and above the staff. If no span bar is in a position, the respective element is set to `#f`.

This grob interface is used in the following graphical object(s): `BarLine` (page 358), and `SpanBar` (page 477).

3.2.11 bar-number-interface

A bar number or bar number vertical support object.

This grob interface is used in the following graphical object(s): `BarNumber` (page 361), `CenteredBarNumber` (page 375), and `CenteredBarNumberLineSpanner` (page 376).

3.2.12 bass-figure-alignment-interface

Align a bass figure.

This grob interface is used in the following graphical object(s): `BassFigureAlignment` (page 363).

3.2.13 bass-figure-interface

A bass figure text.

User settable properties:

`implicit` (boolean)
: Is this an implicit bass figure?

This grob interface is used in the following graphical object(s): `BassFigure` (page 363).

3.2.14 beam-interface

A beam.

The `beam-thickness` property is the weight of beams, measured in staffspace. The `direction` property is not user-serviceable. Use the `direction` property of `Stem` instead. The following properties may be set in the `details` list.

`stem-length-demerit-factor`
: Demerit factor used for inappropriate stem lengths.

`secondary-beam-demerit`
: Demerit used in quanting calculations for multiple beams.

`region-size`
: Size of region for checking quant scores.

`beam-eps` Epsilon for beam quant code to check for presence in gap.

`stem-length-limit-penalty`
: Penalty for differences in stem lengths on a beam.

`damping-direction-penalty`
: Demerit penalty applied when beam direction is different from damping direction.

`hint-direction-penalty`
: Demerit penalty applied when beam direction is different from damping direction, but damping slope is <= `round-to-zero-slope`.

`musical-direction-factor`
: Demerit scaling factor for difference between beam slope and music slope.

`ideal-slope-factor`
: Demerit scaling factor for difference between beam slope and damping slope.

`round-to-zero-slope`
: Damping slope which is considered zero for purposes of calculating direction penalties.

User settable properties:

`auto-knee-gap` (dimension, in staff space)
: If a gap is found between note heads where a horizontal beam fits and it is larger than this number, make a kneed beam.

`beam-thickness` (dimension, in staff space)
: Beam thickness, measured in `staff-space` units.

`beamed-stem-shorten` (list)
: How much to shorten beamed stems, when their direction is forced. It is a list, since the value is different depending on the number of flags and beams.

`beaming` (pair)
: Pair of number lists. Each number list specifies which beams to make. `0` is the central beam, `1` is the next beam toward the note, etc. This information is used to determine how to connect the beaming patterns from stem to stem inside a beam.

`break-overshoot` (pair of numbers)
: How much does a broken spanner stick out of its bounds?

`clip-edges` (boolean)
: Allow outward pointing beamlets at the edges of beams?

`collision-interfaces` (list)
: A list of interfaces for which automatic beam-collision resolution is run.

`collision-voice-only` (boolean)
: Does automatic beam collsion apply only to the voice in which the beam was created?

`concaveness` (number)
: A beam is concave if its inner stems are closer to the beam than the two outside stems. This number is a measure of the closeness of the inner stems. It is used for damping the slope of the beam.

`damping` (number)
: Amount of beam slope damping.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`gap-count` (integer)
: Number of gapped beams for tremolo.

`grow-direction` (direction)
: Crescendo or decrescendo?

`inspect-quants` (pair of numbers)
: If debugging is set, set beam and slur position to a (quantized) position that is as close as possible to this value, and print the demerits for the inspected position in the output.

`knee` (boolean)
: Is this beam kneed?

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

`positions` (pair of numbers)
: Pair of staff coordinates `(`*`start . end`*`)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`skip-quanting` (boolean)
: Should beam quanting be skipped?

`X-positions` (pair of numbers)
: Pair of X staff coordinates of a spanner in the form `(`*`left . right`*`)`, where both *left* and *right* are in `staff-space` units of the current staff.

Internal properties:

`annotation` (string)
: Annotate a grob for debug purposes.

`beam-segments` (list)
: Internal representation of beam segments.

`covered-grobs` (array of grobs)
: Grobs that could potentially collide with a beam.

`least-squares-dy` (number)
: The ideal beam slope, without damping.

`normal-stems` (array of grobs)
: An array of visible stems.

`quantized-positions` (pair of numbers)
: The beam positions after quanting.

`shorten` (dimension, in staff space)
: The amount of space that a stem is shortened. Internally used to distribute beam shortening over stems.

`stems` (array of grobs)
: An array of stem objects.

This grob interface is used in the following graphical object(s): `Beam` (page 366).

3.2.15 `bend-after-interface`

A doit or drop.

User settable properties:

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`delta-position` (number)
: The vertical position difference.

This grob interface is used in the following graphical object(s): `BendAfter` (page 368).

3.2.16 `bend-interface`

The (curved) line representing a bent string.

Available for the `'style` property are `'hold`, `'pre-bend` and `'pre-bend-hold`.

The following properties may be set in the details list.

`arrow-stencil`
: The stencil procedure for the `BendSpanner` arrow head.

`curvature-factor`
: Determines the horizontal part of a bend arrow as percentage of the total horizontal extent, usually between 0 and 1.

`bend-arrowhead-height`
: The height of the arrow head.

`bend-arrowhead-width`
: The width of the arrow head.

`bend-amount-strings`
: An alist with entries for `'quarter`, `'half`, `'three-quarter` and `'full`, which are used to print how much a string is bent.

`curve-x-padding-line-end`
: For a broken `BendSpanner`, set the padding at the line end to subsequent objects like changed `Clef`, etc.

`curve-y-padding-line-end`
: For a broken `BendSpanner` started from a chord the curves don't match; there is a certain vertical gap specified by this value.

`dashed-line-settings`
: List of three numeric values representing on, off and phase of a dashed line.

`head-text-break-visibility`
: A vector of three booleans to set visibility of the arrow head and the text at a line break. This is important for `'style` set to `'hold`, `'pre-bend` or `'pre-bend-hold`.

`horizontal-left-padding`
: The amount of horizontal free space between a `TabNoteHead` and the starting `BendSpanner`.

`successive-level`
: An integer used as a factor determining the vertical coordinate of the starting `BendSpanner`. If `successive-level` is 1, the `BendSpanner` starts at the `TabNoteHead`. If consecutive `BendSpanner`s are set this value should be set to an appropriate value for the first one; later on, this value is maintained by the engraver.

`target-visibility`
: A boolean to decide whether the target `TabNoteHead` should be visible. For up-pointing bends this is usually true.

`y-distance-from-tabstaff-to-arrow-tip`
: This numeric value determines the distance between the `TabStaff` and the arrow head of the `BendSpanner`.

User settable properties:

`bend-me` (boolean)
: Decide whether this grob is bent.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

This grob interface is used in the following graphical object(s): `BendSpanner` (page 369), `NoteColumn` (page 454), `NoteHead` (page 455), and `TabNoteHead` (page 494).

3.2.17 `bezier-curve-interface`

A Bézier curve (tie, slur, etc.)

User settable properties:

`show-control-points` (boolean)
: For grobs printing Bézier curves, setting this property to true causes the control points and control polygon to be drawn on the page for ease of tweaking.

This grob interface is used in the following graphical object(s): `LaissezVibrerTie` (page 430), `PhrasingSlur` (page 462), `RepeatTie` (page 467), `Slur` (page 472), and `Tie` (page 500).

3.2.18 break-alignable-interface

Object that is aligned on a break alignment.

User settable properties:

`break-align-symbols` (list)
: A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`non-break-align-symbols` (list)
: A list of symbols that determine which NON-break-aligned interfaces to align this to.

This grob interface is used in the following graphical object(s): `BarNumber` (page 361), `JumpScript` (page 422), `MetronomeMark` (page 444), and `RehearsalMark` (page 465).

3.2.19 break-aligned-interface

Breakable items.

User settable properties:

`break-align-anchor` (number)
: Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-anchor-alignment` (number)
: Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's extent.

`break-align-symbol` (symbol)
: This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`space-alist` (list)
: An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *spacing-style* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

This grob interface is used in the following graphical object(s): `Ambitus` (page 352), `AmbitusAccidental` (page 354), `BarLine` (page 358), `BreakAlignGroup` (page 371), `BreathingSign` (page 373), `Clef` (page 378), `CueClef` (page 387), `CueEndClef` (page 389), `Custos` (page 392), `DoublePercentRepeat` (page 395), `KeyCancellation` (page 424), `KeySignature` (page 427), `LeftEdge` (page 432), and `TimeSignature` (page 502).

3.2.20 `break-alignment-interface`

The object that performs break alignment.

Three interfaces deal specifically with break alignment:

1. break-alignment-interface (this one),
2. Section 3.2.18 [`break-alignable-interface`], page 533, and
3. Section 3.2.19 [`break-aligned-interface`], page 533.

Each of these interfaces supports grob properties that use *break-align symbols*, which are Scheme symbols that are used to specify the alignment, ordering, and spacing of certain notational elements ('breakable' items).

Available break-align symbols:

```
ambitus
breathing-sign
clef
```

```
cue-clef
cue-end-clef
custos
key-cancellation
key-signature
left-edge
staff-bar
time-signature
```

User settable properties:

`break-align-orders` (vector)
: This is a vector of 3 lists: `#(end-of-line unbroken start-of-line)`. Each list contains *break-align symbols* that specify an order of breakable items (see Section "break-alignment-interface" in *Internals Reference*).

  For example, this places time signatures before clefs:

```
\override Score.BreakAlignment.break-align-orders =
  #(make-vector 3 '(left-edge
                    cue-end-clef
                    ambitus
                    breathing-sign
                    time-signature
                    clef
                    cue-clef
                    staff-bar
                    key-cancellation
                    key-signature
                    custos))
```

Internal properties:

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): `BreakAlignment` (page 372).

3.2.21 breathing-sign-interface

A breathing sign.

User settable properties:

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

This grob interface is used in the following graphical object(s): `BreathingSign` (page 373).

3.2.22 centered-bar-number-interface

A measure-centered bar number.

This grob interface is used in the following graphical object(s): `CenteredBarNumber` (page 375).

3.2.23 centered-bar-number-line-spanner-interface

An abstract object used to align centered bar numbers on the same vertical position.

This grob interface is used in the following graphical object(s): CenteredBarNumberLineSpanner (page 376).

3.2.24 centered-text-interface

A spanner that interprets a markup centered between two columns.

User settable properties:

self-alignment-X (number)
: Specify alignment of an object. The value -1 means left aligned, 0 centered, and 1 right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

spacing-pair (pair)
: A pair of alignment symbols which set an object's spacing relative to its left and right BreakAlignments.

  For example, a MultiMeasureRest will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

This grob interface is used in the following graphical object(s): CenteredBarNumber (page 375), and MeasureCounter (page 439).

3.2.25 chord-name-interface

A chord label (name or fretboard).

Internal properties:

begin-of-line-visible (boolean)
: Set to make ChordName or FretBoard be visible only at beginning of line or at chord changes.

This grob interface is used in the following graphical object(s): ChordName (page 377), and FretBoard (page 412).

3.2.26 clef-interface

A clef sign.

User settable properties:

full-size-change (boolean)
: Don't make a change clef smaller.

glyph (string)
: A string determining what 'style' of glyph is typeset. Valid choices depend on the function that is reading this property.

  In combination with (span) bar lines, it is a string resembling the bar line appearance in ASCII form.

glyph-name (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of glyph, where decisions about line breaking etc. are already taken.

`non-default` (boolean)
: Set for manually specified clefs and keys.

This grob interface is used in the following graphical object(s): `Clef` (page 378), `CueClef` (page 387), and `CueEndClef` (page 389).

3.2.27 `clef-modifier-interface`

The number describing transposition of the clef, placed below or above clef sign. Usually this is 8 (octave transposition) or 15 (two octaves), but LilyPond allows any integer here.

User settable properties:

`clef-alignments` (list)
: An alist of parent-alignments that should be used for clef modifiers with various clefs

This grob interface is used in the following graphical object(s): `ClefModifier` (page 381).

3.2.28 `cluster-beacon-interface`

A place holder for the cluster spanner to determine the vertical extents of a cluster spanner at this X position.

User settable properties:

`positions` (pair of numbers)
: Pair of staff coordinates `(`*`start . end`*`)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

This grob interface is used in the following graphical object(s): `ClusterSpannerBeacon` (page 383).

3.2.29 `cluster-interface`

A graphically drawn musical cluster.

`padding` adds to the vertical extent of the shape (top and bottom).

The property `style` controls the shape of cluster segments. Valid values include `leftsided-stairs`, `rightsided-stairs`, `centered-stairs`, and `ramp`.

User settable properties:

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

Internal properties:

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

This grob interface is used in the following graphical object(s): `ClusterSpanner` (page 382).

3.2.30 control-point-interface

A grob used to visualize one control point of a Bézier curve (such as a tie or a slur), for ease of tweaking.

Internal properties:

`bezier` (graphical (layout) object)
: A pointer to a Bézier curve, for use by control points and polygons.

`index` (non-negative, exact integer)
: For some grobs in a group, this is a number associated with the grob.

This grob interface is used in the following graphical object(s): `ControlPoint` (page 385).

3.2.31 control-polygon-interface

A grob used to visualize the control polygon of a Bézier curve (such as a tie or a slur), for ease of tweaking.

User settable properties:

`extroversion` (number)
: For polygons, how the thickness of the line is spread on each side of the exact polygon whith ideal zero thickness. If this is 0, the middle of line is on the polygon. If 1, the line sticks out of the polygon. If -1, the outer side of the line is exactly on the polygon. Other numeric values are interpolated.

`filled` (boolean)
: Whether an object is filled with ink.

Internal properties:

`bezier` (graphical (layout) object)
: A pointer to a Bézier curve, for use by control points and polygons.

This grob interface is used in the following graphical object(s): `ControlPolygon` (page 386).

3.2.32 custos-interface

A custos object. `style` can have four valid values: `mensural`, `vaticana`, `medicaea`, and `hufnagel`. `mensural` is the default style.

User settable properties:

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

`neutral-position` (number)
: Position (in half staff spaces) where to flip the direction of custos stem.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

This grob interface is used in the following graphical object(s): `Custos` (page 392).

3.2.33 dot-column-interface

Group dot objects so they form a column, and position dots so they do not clash with staff lines.

User settable properties:

`chord-dots-limit` (integer)
: Limits the column of dots on each chord to the height of the chord plus `chord-dots-limit` staff-positions.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

Internal properties:

`dots` (array of grobs)
: Multiple `Dots` objects.

`note-collision` (graphical (layout) object)
: The `NoteCollision` object of a dot column.

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): `DotColumn` (page 394).

3.2.34 dots-interface

The dots to go with a notehead or rest. `direction` sets the preferred direction to move in case of staff line collisions. `style` defaults to undefined, which is normal 19th/20th century traditional style. Set `style` to `vaticana` for ancient type dots.

User settable properties:

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`dot-count` (integer)
: The number of dots.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

This grob interface is used in the following graphical object(s): `Dots` (page 394).

3.2.35 duration-line-interface

A line lasting for the duration of a rhythmic event.

User settable properties:

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

This grob interface is used in the following graphical object(s): `DurationLine` (page 399).

3.2.36 dynamic-interface

Any kind of loudness sign.

This grob interface is used in the following graphical object(s): `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), and `Hairpin` (page 417).

3.2.37 dynamic-line-spanner-interface

Dynamic line spanner.

User settable properties:

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

This grob interface is used in the following graphical object(s): `DynamicLineSpanner` (page 401).

3.2.38 dynamic-text-interface

An absolute text dynamic.

User settable properties:

`right-padding` (dimension, in staff space)
: Space to insert on the right side of an object (e.g., between note and its accidentals).

This grob interface is used in the following graphical object(s): `DynamicText` (page 402).

3.2.39 dynamic-text-spanner-interface

Dynamic text spanner.

User settable properties:

`text` (markup)
: Text markup. See Section “Formatting text” in *Notation Reference*.

This grob interface is used in the following graphical object(s): `DynamicTextSpanner` (page 403).

3.2.40 enclosing-bracket-interface

Brackets alongside bass figures.

User settable properties:

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

This grob interface is used in the following graphical object(s): `BassFigureBracket` (page 365).

3.2.41 `episema-interface`

An episema line.

This grob interface is used in the following graphical object(s): `Episema` (page 405).

3.2.42 `figured-bass-continuation-interface`

Simple extender line between bounds.

User settable properties:

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`figures` (array of grobs)
: Figured bass objects for continuation line.

This grob interface is used in the following graphical object(s): `BassFigureContinuation` (page 365).

3.2.43 finger-glide-interface

The line between `Fingering` grobs indicating a glide with that finger.

The property `style` may take the following symbols.

`line` A simple connecting line.

`dashed-line`
: Print a dashed line. Customizable with settings for `dash-fraction` and `dash-period`.

`dotted-line`
: Print a dotted line.

`stub-right`
: The printed line is limited to a certain amount right before its right bound. This amount is configurable by a suitable setting for `bound-details.right.right-stub-length`.

`stub-left`
: The printed line is limited to a certain amount right after its left bound. The amount is configurable by a suitable setting for `bound-details.right.left-stub-length`.

`stub-both`
: The printed line combines the settings of `stub-left` and `stub-right`.

`zigzag` A zigzag line, configurable with suitable settings for `zigzag-width` and `zigzag-length`.

`trill` A trill style line.

`bow` A bow style line. The orientation of the bow may be tweaked with a suitable setting of `details.bow-direction`.

User settable properties:

`dash-fraction` (number)
: Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`dash-period` (number)
: The length of one dash together with whitespace. If negative, no line is drawn at all.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`zigzag-length` (dimension, in staff space)
: The length of the lines of a zigzag, relative to `zigzag-width`. A value of `1` gives 60-degree zigzags.

`zigzag-width` (dimension, in staff space)
: The width of one zigzag squiggle. This number is adjusted slightly so that the spanner line can be constructed from a whole number of squiggles.

This grob interface is used in the following graphical object(s): `FingerGlideSpanner` (page 406).

3.2.44 finger-interface

A fingering instruction.

This grob interface is used in the following graphical object(s): `Fingering` (page 408).

3.2.45 fingering-column-interface

Makes sure that fingerings placed laterally do not collide and that they are flush if necessary.

User settable properties:

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`snap-radius` (number)
: The maximum distance between two objects that will cause them to snap to alignment along an axis.

Internal properties:

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): `FingeringColumn` (page 410).

3.2.46 flag-interface

A flag that gets attached to a stem.The style property is symbol determining what style of flag glyph is typeset on a `Stem`. Valid options include `'()` for standard flags, `'mensural` and `'no-flag`, which switches off the flag.

User settable properties:

`glyph-name` (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`stroke-style` (string)
: Set to `"grace"` to turn stroke through flag on.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

This grob interface is used in the following graphical object(s): `Flag` (page 410).

3.2.47 font-interface

Any symbol that is typeset through fixed sets of glyphs, (i.e., fonts).

User settable properties:

`font-encoding` (symbol)
: The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-family` (symbol)
: The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`font-features` (list)
: Opentype features.

`font-name` (string)
: Specifies a file name (without extension) of the font to load. This setting overrides selection using `font-family`, `font-series` and `font-shape`.

`font-series` (symbol)
: Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-shape` (symbol)
: Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number)
: The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

Internal properties:

`font` (font metric)
: A cached font metric object.

This grob interface is used in the following graphical object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalSuggestion` (page 350), `AmbitusAccidental` (page 354), `AmbitusLine` (page 354), `AmbitusNoteHead` (page 355), `Arpeggio` (page 355), `BalloonText` (page 357), `BarLine` (page 358), `BarNumber` (page 361), `BassFigure` (page 363), `BendSpanner` (page 369), `BreathingSign` (page 373), `CenteredBarNumber` (page 375), `ChordName` (page 377), `Clef` (page 378), `ClefModifier` (page 381), `CombineTextScript` (page 383), `CueClef` (page 387), `CueEndClef` (page 389), `Custos` (page 392), `Dots` (page 394), `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DurationLine` (page 399), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `Episema` (page 405), `Fingering` (page 408), `Flag` (page 410), `Footnote` (page 411), `FretBoard` (page 412), `HorizontalBracketText` (page 419), `InstrumentName` (page 420), `InstrumentSwitch` (page 421), `JumpScript` (page 422), `KeyCancellation` (page 424), `KeySignature` (page 427), `KievanLigature` (page 430), `LyricHyphen` (page 436), `LyricText` (page 438), `MeasureCounter` (page 439), `MeasureSpanner` (page 442), `MensuralLigature` (page 444), `MetronomeMark` (page 444), `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `NonMusicalPaperColumn` (page 452), `NoteHead` (page 455), `NoteName` (page 456), `OttavaBracket` (page 457), `PaperColumn` (page 458), `Parentheses` (page 459), `PercentRepeat` (page 460), `PercentRepeatCounter` (page 461), `RehearsalMark` (page 465), `Rest` (page 469), `Script` (page 470), `SostenutoPedal` (page 474), `SpanBar` (page 477), `StanzaNumber` (page 481), `StringNumber` (page 485), `StrokeFinger` (page 487), `SustainPedal` (page 488), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), `SystemStartSquare` (page 494), `TabNoteHead` (page 494), `TextScript` (page 496), `TextSpanner` (page 498), `TimeSignature` (page 502), `TrillPitchAccidental` (page 504), `TrillPitchHead` (page 506), `TrillPitchParentheses` (page 507), `TrillSpanner` (page 507), `TupletNumber` (page 510), `UnaCordaPedal` (page 511), `VaticanaLigature` (page 513), and `VoltaBracket` (page 517).

3.2.48 footnote-interface

Make a footnote.

User settable properties:

`automatically-numbered` (boolean)
: If set, footnotes are automatically numbered.

`footnote` (boolean)
: Should this be a footnote or in-note?

`footnote-text` (markup)
: A footnote for the grob.

Internal properties:

`numbering-assertion-function` (any type)
: The function used to assert that footnotes are receiving correct automatic numbers.

`spanner-placement` (direction)
: The place of an annotation on a spanner. `LEFT` is for the first spanner, and `RIGHT` is for the last. `CENTER` will place it on the broken spanner that falls closest to the center of the length of the entire spanner, although this behavior is unpredictable in situations with lots of rhythmic diversity. For predictable results, use `LEFT` and `RIGHT`.

This grob interface is used in the following graphical object(s): `Footnote` (page 411).

3.2.49 fret-diagram-interface

A fret diagram

User settable properties:

`align-dir` (direction)
: Which side to align? `-1`: left side, `0`: around center of width, `1`: right side.

`dot-placement-list` (list)
: List consisting of `(`*`description string-number fret-number finger-number`*`)` entries used to define fret diagrams.

`fret-diagram-details` (list)
: An alist of detailed grob properties for fret diagrams. Each alist entry consists of a `(`*`property . value`*`)` pair. The properties which can be included in `fret-diagram-details` include the following:

  - `barre-type` – Type of barre indication used. Choices include `curved`, `straight`, and `none`. Default `curved`.
  - `capo-thickness` – Thickness of capo indicator, in multiples of fret-space. Default value 0.5.
  - `dot-color` – Color of dots. Options include `black` and `white`. Default `black`.
  - `dot-label-font-mag` – Magnification for font used to label fret dots. Default value 1.
  - `dot-position` – Location of dot in fret space. Default 0.6 for dots without labels, 0.95-`dot-radius` for dots with labels.

- `dot-radius` – Radius of dots, in terms of fret spaces. Default value 0.425 for labeled dots, 0.25 for unlabeled dots.
- `finger-code` – Code for the type of fingering indication used. Options include `none`, `in-dot`, and `below-string`. Default `none` for markup fret diagrams, `below-string` for `FretBoards` fret diagrams.
- `fret-count` – The number of frets. Default 4.
- `fret-distance` – Multiplier to adjust the distance between frets. Default 1.0.
- `fret-label-custom-format` – The format string to be used label the lowest fret number, when `number-type` equals to `custom`. Default `"~a"`.
- `fret-label-font-mag` – The magnification of the font used to label the lowest fret number. Default 0.5.
- `fret-label-vertical-offset` – The offset of the fret label from the center of the fret in direction parallel to strings. Default 0.
- `fret-label-horizontal-offset` – The offset of the fret label from the center of the fret in direction orthogonal to strings. Default 0.
- `handedness` – Print the fret-diagram left- or right-handed. `-1`, `LEFT` for left ; `1`, `RIGHT` for right. Default `RIGHT`.
- `paren-padding` – The padding for the parenthesis. Default 0.05.
- `label-dir` – Side to which the fret label is attached. `-1`, `LEFT`, or `DOWN` for left or down; `1`, `RIGHT`, or `UP` for right or up. Default `RIGHT`.
- `mute-string` – Character string to be used to indicate muted string. Default `"x"`.
- `number-type` – Type of numbers to use in fret label. Choices include `roman-lower`, `roman-upper`, `arabic` and `custom`. In the later case, the format string is supplied by the `fret-label-custom-format` property. Default `roman-lower`.
- `open-string` – Character string to be used to indicate open string. Default `"o"`.
- `orientation` – Orientation of fret-diagram. Options include `normal`, `landscape`, and `opposing-landscape`. Default `normal`.
- `string-count` – The number of strings. Default 6.
- `string-distance` – Multiplier to adjust the distance between strings. Default 1.0.
- `string-label-font-mag` – The magnification of the font used to label fingerings at the string, rather than in the dot. Default value 0.6 for `normal` orientation, 0.5 for `landscape` and `opposing-landscape`.
- `string-thickness-factor` – Factor for changing thickness of each string in the fret diagram. Thickness of string k is given by `thickness` * (1+`string-thickness-factor`) ^ (k-1). Default 0.
- `top-fret-thickness` – The thickness of the top fret line, as a multiple of the standard thickness. Default value 3.
- `xo-font-magnification` – Magnification used for mute and open string indicators. Default value 0.5.
- `xo-padding` – Padding for open and mute indicators from top fret. Default value 0.25.

`size` (number)

The ratio of the size of the object to its default size.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `FretBoard` (page 412).

3.2.50 glissando-interface

A glissando.

Internal properties:

`glissando-index` (integer)
: The index of a glissando in its note column.

This grob interface is used in the following graphical object(s): `Glissando` (page 414).

3.2.51 grace-spacing-interface

Keep track of durations in a run of grace notes.

User settable properties:

`common-shortest-duration` (moment)
: The most common shortest note length. This is used in spacing. Enlarging this sets the score tighter.

Internal properties:

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

This grob interface is used in the following graphical object(s): `GraceSpacing` (page 415).

3.2.52 gregorian-ligature-interface

A gregorian ligature.

Internal properties:

`ascendens` (boolean)
: Is this neume of ascending type?

`auctum` (boolean)
: Is this neume liquescentically augmented?

`cavum` (boolean)
: Is this neume outlined?

`context-info` (integer)
: Within a ligature, the final glyph or shape of a head may be affected by the left and/or right neighbour head. `context-info` holds for each head such information about the left and right neighbour, encoded as a bit mask.

`deminutum` (boolean)
: Is this neume deminished?

`descendens` (boolean)
: Is this neume of descendent type?

`inclinatum` (boolean)
: Is this neume an inclinatum?

`linea` (boolean)
: Attach vertical lines to this neume?

`oriscus` (boolean)
: Is this neume an oriscus?

`pes-or-flexa` (boolean)
: Shall this neume be joined with the previous head?

`prefix-set` (number)
: A bit mask that holds all Gregorian head prefixes, such as `\virga` or `\quilisma`.

`quilisma` (boolean)
: Is this neume a quilisma?

`stropha` (boolean)
: Is this neume a stropha?

`virga` (boolean)
: Is this neume a virga?

This grob interface is used in the following graphical object(s): `NoteHead` (page 455).

3.2.53 grid-line-interface

A line that is spanned between grid-points.

User settable properties:

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*`.StaffSymbol.thickness`).

Internal properties:

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

This grob interface is used in the following graphical object(s): `GridLine` (page 416).

3.2.54 grid-point-interface

A spanning point for grid lines.

This grob interface is used in the following graphical object(s): `GridPoint` (page 416).

3.2.55 grob-interface

A grob represents a piece of music notation.

All grobs have an X and Y position on the page. These X and Y positions are stored in a relative format, thus they can easily be combined by stacking them, hanging one grob to the side of another, or coupling them into grouping objects.

Each grob has a reference point (a.k.a. parent): The position of a grob is stored relative to that reference point. For example, the X reference point of a staccato dot usually is the note head that it applies to. When the note head is moved, the staccato dot moves along automatically.

A grob is often associated with a symbol, but some grobs do not print any symbols. They take care of grouping objects. For example, there is a separate grob that stacks staves vertically. The Section 3.1.87 [NoteCollision], page 453, object is also an abstract grob: It only moves around chords, but doesn't print anything.

Grobs have properties (Scheme variables) that can be read and set. Two types of them exist: immutable and mutable. Immutable variables define the default style and behavior. They are shared between many objects. They can be changed using `\override` and `\revert`. Mutable properties are variables that are specific to one grob. Typically, lists of other objects, or results from computations are stored in mutable properties. In particular, every call to `ly:grob-set-property!` (or its C++ equivalent) sets a mutable property.

The properties `after-line-breaking` and `before-line-breaking` are dummies that are not user-serviceable.

User settable properties:

`after-line-breaking` (boolean)
: Dummy property, used to trigger callback for `after-line-breaking`.

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`before-line-breaking` (boolean)
: Dummy property, used to trigger a callback function.

`color` (color)
: The color of this grob.

`extra-offset` (pair of numbers)
: A pair representing an offset. This offset is added just before outputting the symbol, so the typesetting engine is completely oblivious to it. The values are measured in `staff-space` units of the staff's `StaffSymbol`.

`footnote-music` (music)
: Music creating a footnote.

`forced-spacing` (number)
: Spacing forced between grobs, used in various ligature engravers.

`horizontal-skylines` (pair of skylines)
: Two skylines, one to the left and one to the right of this grob.

`id` (string)
: An id string for the grob.

`layer` (integer)
: An integer which determines the order of printing objects. 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.

`minimum-X-extent` (pair of numbers)
: Minimum size of an object in X dimension, measured in `staff-space` units.

`minimum-Y-extent` (pair of numbers)
: Minimum size of an object in Y dimension, measured in `staff-space` units.

`output-attributes` (list)
: An alist of attributes for the grob, to be included in output files. When the SVG typesetting backend is used, the attributes are assigned to a group (`<g>`) containing all of the stencils that comprise a given grob. For example,

```
'((id . 123) (class . foo) (data-whatever . "bar"))
```

produces

```
<g id="123" class="foo" data-whatever="bar"> ... </g>
```

In the Postscript backend, where there is no way to group items, the setting of the `output-attributes` property has no effect.

`parenthesis-friends` (list)
: A list of Grob types, as symbols. When parentheses enclose a Grob that has 'parenthesis-friends, the parentheses widen to include any child Grobs with type among 'parenthesis-friends.

`parenthesis-id` (symbol)
: When parenthesized grobs created in the same time step have this property, there is one set of parentheses for each group of grobs having the same value.

`parenthesized` (boolean)
: Parenthesize this grob.

`rotation` (list)
: Number of degrees to rotate this object, and what point to rotate around. For example, `'(45 0 0)` rotates by 45 degrees around the center of this object.

`skyline-horizontal-padding` (number)
: For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`springs-and-rods` (boolean)
: Dummy variable for triggering spacing routines.

`stencil` (stencil)
: The symbol to print.

`transparent` (boolean)
: This makes the grob invisible.

`vertical-skylines` (pair of skylines)
: Two skylines, one above and one below this grob.

`whiteout` (boolean-or-number)
: If a number or true, the grob is printed over a white background to white-out underlying material, if the grob is visible. A number indicates how far the white background extends beyond the bounding box of the grob as a multiple of the staff-line thickness. The `LyricHyphen` grob uses a special implementation of whiteout: A positive number indicates how far the white background extends beyond the bounding box in multiples of `line-thickness`. The shape of the background is determined by `whiteout-style`. Usually `#f` by default.

`whiteout-style` (symbol)
: Determines the shape of the `whiteout` background. Available are `'outline`, `'rounded-box`, and the default `'box`. There is one exception: Use `'special` for `LyricHyphen`.

`X-extent` (pair of numbers)
: Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number)
: The horizontal amount that this object is moved relative to its X-parent.

`Y-extent` (pair of numbers)
: Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number)
: The vertical amount that this object is moved relative to its Y-parent.

Internal properties:

`axis-group-parent-X` (graphical (layout) object)
: Containing X axis group.

`axis-group-parent-Y` (graphical (layout) object)
: Containing Y axis group.

`cause` (any type)
: Any kind of causation objects (i.e., music, or perhaps translator) that was the cause for this grob.

`cross-staff` (boolean)
: True for grobs whose `Y-extent` depends on inter-staff spacing. The extent is measured relative to the grobs's parent staff (more generally, its `VerticalAxisGroup`) so this boolean flags grobs that are not rigidly fixed to their parent staff. Beams that join notes from two staves are `cross-staff`. Grobs that are positioned around such beams are also `cross-staff`. Grobs that are grouping objects, however, like `VerticalAxisGroups` will not in general be marked `cross-staff` when some of the members of the group are `cross-staff`.

`interfaces` (list)
: A list of symbols indicating the interfaces supported by this object. It is initialized from the `meta` field.

`meta` (list) Provide meta information. It is an alist with the entries `name` and `interfaces`.

`pure-Y-offset-in-progress` (boolean)
: A debugging aid for catching cyclic dependencies.

`staff-symbol` (graphical (layout) object)
: The staff symbol grob that we are in.

This grob interface is used in the following graphical object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), `AccidentalPlacement` (page 350), `AccidentalSuggestion` (page 350), `Ambitus` (page 352), `AmbitusAccidental` (page 354), `AmbitusLine` (page 354), `AmbitusNoteHead` (page 355), `Arpeggio` (page 355), `BalloonText` (page 357), `BarLine` (page 358), `BarNumber` (page 361), `BassFigure` (page 363), `BassFigureAlignment` (page 363), `BassFigureAlignmentPositioning` (page 364),

BassFigureBracket (page 365), BassFigureContinuation (page 365), BassFigureLine (page 366), Beam (page 366), BendAfter (page 368), BendSpanner (page 369), BreakAlignGroup (page 371), BreakAlignment (page 372), BreathingSign (page 373), CenteredBarNumber (page 375), CenteredBarNumberLineSpanner (page 376), ChordName (page 377), Clef (page 378), ClefModifier (page 381), ClusterSpanner (page 382), ClusterSpannerBeacon (page 383), CombineTextScript (page 383), ControlPoint (page 385), ControlPolygon (page 386), CueClef (page 387), CueEndClef (page 389), Custos (page 392), DotColumn (page 394), Dots (page 394), DoublePercentRepeat (page 395), DoublePercentRepeatCounter (page 396), DoubleRepeatSlash (page 398), DurationLine (page 399), DynamicLineSpanner (page 401), DynamicText (page 402), DynamicTextSpanner (page 403), Episema (page 405), FingerGlideSpanner (page 406), Fingering (page 408), FingeringColumn (page 410), Flag (page 410), Footnote (page 411), FretBoard (page 412), Glissando (page 414), GraceSpacing (page 415), GridLine (page 416), GridPoint (page 416), Hairpin (page 417), HorizontalBracket (page 418), HorizontalBracketText (page 419), InstrumentName (page 420), InstrumentSwitch (page 421), JumpScript (page 422), KeyCancellation (page 424), KeySignature (page 427), KievanLigature (page 430), LaissezVibrerTie (page 430), LaissezVibrerTieColumn (page 431), LedgerLineSpanner (page 432), LeftEdge (page 432), LigatureBracket (page 434), LyricExtender (page 436), LyricHyphen (page 436), LyricSpace (page 437), LyricText (page 438), MeasureCounter (page 439), MeasureGrouping (page 441), MeasureSpanner (page 442), MelodyItem (page 443), MensuralLigature (page 444), MetronomeMark (page 444), MultiMeasureRest (page 446), MultiMeasureRestNumber (page 447), MultiMeasureRestScript (page 449), MultiMeasureRestText (page 450), NonMusicalPaperColumn (page 452), NoteCollision (page 453), NoteColumn (page 454), NoteHead (page 455), NoteName (page 456), NoteSpacing (page 456), OttavaBracket (page 457), PaperColumn (page 458), Parentheses (page 459), PercentRepeat (page 460), PercentRepeatCounter (page 461), PhrasingSlur (page 462), PianoPedalBracket (page 464), RehearsalMark (page 465), RepeatSlash (page 467), RepeatTie (page 467), RepeatTieColumn (page 469), Rest (page 469), RestCollision (page 470), Script (page 470), ScriptColumn (page 471), ScriptRow (page 472), Slur (page 472), SostenutoPedal (page 474), SostenutoPedalLineSpanner (page 475), SpacingSpanner (page 476), SpanBar (page 477), SpanBarStub (page 478), StaffGrouper (page 479), StaffSpacing (page 479), StaffSymbol (page 480), StanzaNumber (page 481), Stem (page 481), StemStub (page 483), StemTremolo (page 484), StringNumber (page 485), StrokeFinger (page 487), SustainPedal (page 488), SustainPedalLineSpanner (page 489), System (page 490), SystemStartBar (page 491), SystemStartBrace (page 492), SystemStartBracket (page 493), SystemStartSquare (page 494), TabNoteHead (page 494), TextScript (page 496), TextSpanner (page 498), Tie (page 500), TieColumn (page 501), TimeSignature (page 502), TrillPitchAccidental (page 504), TrillPitchGroup (page 505), TrillPitchHead (page 506), TrillPitchParentheses (page 507), TrillSpanner (page 507), TupletBracket (page 509), TupletNumber (page 510), UnaCordaPedal (page 511), UnaCordaPedalLineSpanner (page 512), VaticanaLigature (page 513), VerticalAlignment (page 514), VerticalAxisGroup (page 514), VoiceFollower (page 516), VoltaBracket (page 517), VoltaBracketSpanner (page 518), and VowelTransition (page 520).

3.2.56 hairpin-interface

A hairpin crescendo or decrescendo.

User settable properties:

`bound-padding` (number)
: The amount of padding to insert around spanner bounds.

`broken-bound-padding` (number)
: The amount of padding to insert when a spanner is broken at a line break.

`circled-tip` (boolean)
: Put a circle at start/end of hairpins (al/del niente).

`endpoint-alignments` (pair of numbers)
: A pair of numbers representing the alignments of an object's endpoints. E.g., the ends of a hairpin relative to `NoteColumn` grobs.

`grow-direction` (direction)
: Crescendo or decrescendo?

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

Internal properties:

`adjacent-spanners` (array of grobs)
: An array of directly neighboring dynamic spanners.

`concurrent-hairpins` (array of grobs)
: All concurrent hairpins.

This grob interface is used in the following graphical object(s): `Hairpin` (page 417).

3.2.57 hara-kiri-group-spanner-interface

A group spanner that keeps track of interesting items. If it doesn't contain any after line breaking, it removes itself and all its children. Greater control can be exercised via `remove-layer` which can prioritize layers so only the lowest-numbered non-empty layer is retained; make the layer independent of the group; or make it dependent on any other member of the group

User settable properties:

`remove-empty` (boolean)
: If set, remove group if it contains no interesting items.

`remove-first` (boolean)
: Remove the first staff of an orchestral score?

`remove-layer` (index or symbol)
: When set as a positive integer, the `Keep_alive_together_engraver` removes all `VerticalAxisGroup` grobs with a `remove-layer` larger than the smallest retained `remove-layer`. Set to `#f` to make a layer independent of the `Keep_alive_together_engraver`. Set to `'()`, the layer does not participate in the layering decisions. The property can also be set as a symbol for common behaviors: `#'any` to keep the layer alive with any other layer in the group; `#'above` or `#'below` to keep the layer alive with the context immediately before or after it, respectively.

Internal properties:

`important-column-ranks` (vector)
: A cache of columns that contain `items-worth-living` data.

`items-worth-living` (array of grobs)
: An array of interesting items. If empty in a particular staff, then that staff is erased.

`keep-alive-with` (array of grobs)
: An array of other `VerticalAxisGroup`s. If any of them are alive, then we will stay alive.

`make-dead-when` (array of grobs)
: An array of other `VerticalAxisGroup`s. If any of them are alive, then we will turn dead.

This grob interface is used in the following graphical object(s): `VerticalAxisGroup` (page 514).

3.2.58 `horizontal-bracket-interface`

A horizontal bracket encompassing notes.

User settable properties:

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`connect-to-neighbor` (pair)
: Pair of booleans, indicating whether this grob looks as a continued break.

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

Internal properties:

`bracket-text` (graphical (layout) object)
: The text for an analysis bracket.

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

This grob interface is used in the following graphical object(s): `HorizontalBracket` (page 418), `OttavaBracket` (page 457), and `VoltaBracket` (page 517).

3.2.59 `horizontal-bracket-text-interface`

Label for an analysis bracket.

Internal properties:

`bracket` (graphical (layout) object)
: The bracket for a number.

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

This grob interface is used in the following graphical object(s): `HorizontalBracketText` (page 419).

3.2.60 `inline-accidental-interface`

An inlined accidental (i.e. normal accidentals, cautionary accidentals).

This grob interface is used in the following graphical object(s): `Accidental` (page 348), `AccidentalCautionary` (page 349), and `TrillPitchAccidental` (page 504).

3.2.61 `instrument-specific-markup-interface`

Instrument-specific markup (like fret boards or harp pedal diagrams).

User settable properties:

`fret-diagram-details` (list)

An alist of detailed grob properties for fret diagrams. Each alist entry consists of a (*property . value*) pair. The properties which can be included in `fret-diagram-details` include the following:

- `barre-type` – Type of barre indication used. Choices include `curved`, `straight`, and `none`. Default `curved`.
- `capo-thickness` – Thickness of capo indicator, in multiples of fret-space. Default value 0.5.
- `dot-color` – Color of dots. Options include `black` and `white`. Default `black`.
- `dot-label-font-mag` – Magnification for font used to label fret dots. Default value 1.
- `dot-position` – Location of dot in fret space. Default 0.6 for dots without labels, 0.95-`dot-radius` for dots with labels.
- `dot-radius` – Radius of dots, in terms of fret spaces. Default value 0.425 for labeled dots, 0.25 for unlabeled dots.
- `finger-code` – Code for the type of fingering indication used. Options include `none`, `in-dot`, and `below-string`. Default `none` for markup fret diagrams, `below-string` for `FretBoards` fret diagrams.
- `fret-count` – The number of frets. Default 4.
- `fret-distance` – Multiplier to adjust the distance between frets. Default 1.0.
- `fret-label-custom-format` – The format string to be used label the lowest fret number, when `number-type` equals to `custom`. Default `"~a"`.
- `fret-label-font-mag` – The magnification of the font used to label the lowest fret number. Default 0.5.
- `fret-label-vertical-offset` – The offset of the fret label from the center of the fret in direction parallel to strings. Default 0.
- `fret-label-horizontal-offset` – The offset of the fret label from the center of the fret in direction orthogonal to strings. Default 0.
- `handedness` – Print the fret-diagram left- or right-handed. `-1`, `LEFT` for left ; `1`, `RIGHT` for right. Default `RIGHT`.
- `paren-padding` – The padding for the parenthesis. Default 0.05.
- `label-dir` – Side to which the fret label is attached. `-1`, `LEFT`, or `DOWN` for left or down; `1`, `RIGHT`, or `UP` for right or up. Default `RIGHT`.

- `mute-string` – Character string to be used to indicate muted string. Default `"x"`.
- `number-type` – Type of numbers to use in fret label. Choices include `roman-lower`, `roman-upper`, `arabic` and `custom`. In the later case, the format string is supplied by the `fret-label-custom-format` property. Default `roman-lower`.
- `open-string` – Character string to be used to indicate open string. Default `"o"`.
- `orientation` – Orientation of fret-diagram. Options include `normal`, `landscape`, and `opposing-landscape`. Default `normal`.
- `string-count` – The number of strings. Default 6.
- `string-distance` – Multiplier to adjust the distance between strings. Default 1.0.
- `string-label-font-mag` – The magnification of the font used to label fingerings at the string, rather than in the dot. Default value 0.6 for `normal` orientation, 0.5 for `landscape` and `opposing-landscape`.
- `string-thickness-factor` – Factor for changing thickness of each string in the fret diagram. Thickness of string k is given by `thickness` * (1+`string-thickness-factor`) ^ (k-1). Default 0.
- `top-fret-thickness` – The thickness of the top fret line, as a multiple of the standard thickness. Default value 3.
- `xo-font-magnification` – Magnification used for mute and open string indicators. Default value 0.5.
- `xo-padding` – Padding for open and mute indicators from top fret. Default value 0.25.

`graphical` (boolean)
: Display in graphical (vs. text) form.

`harp-pedal-details` (list)
: An alist of detailed grob properties for harp pedal diagrams. Each alist entry consists of a (*property* . *value*) pair. The properties which can be included in harp-pedal-details include the following:
  - `box-offset` – Vertical shift of the center of flat/sharp pedal boxes above/ below the horizontal line. Default value 0.8.
  - `box-width` – Width of each pedal box. Default value 0.4.
  - `box-height` – Height of each pedal box. Default value 1.0.
  - `space-before-divider` – Space between boxes before the first divider (so that the diagram can be made symmetric). Default value 0.8.
  - `space-after-divider` – Space between boxes after the first divider. Default value 0.8.
  - `circle-thickness` – Thickness (in unit of the line-thickness) of the ellipse around circled pedals. Default value 0.5.
  - `circle-x-padding` – Padding in X direction of the ellipse around circled pedals. Default value 0.15.
  - `circle-y-padding` – Padding in Y direction of the ellipse around circled pedals. Default value 0.2.

`size` (number)
: The ratio of the size of the object to its default size.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.`StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `TextScript` (page 496).

3.2.62 `item-interface`

Grobs can be distinguished in their role in the horizontal spacing. Many grobs define constraints on the spacing by their sizes, for example, note heads, clefs, stems, and all other symbols with a fixed shape. These grobs form a subtype called `Item`.

Some items need special treatment for line breaking. For example, a clef is normally only printed at the start of a line (i.e., after a line break). To model this, 'breakable' items (clef, key signature, bar lines, etc.) are copied twice. Then we have three versions of each breakable item: one version if there is no line break, one version that is printed before the line break (at the end of a system), and one version that is printed after the line break.

Whether these versions are visible and take up space is determined by the outcome of the `break-visibility` grob property, which is a function taking a direction (`-1`, `0` or `1`) as an argument. It returns a cons of booleans, signifying whether this grob should be transparent and have no extent.

The following variables for `break-visibility` are predefined:

grob will show:	before break	no break	after break
all-invisible	no	no	no
begin-of-line-visible	no	no	yes
end-of-line-visible	yes	no	no
all-visible	yes	yes	yes
begin-of-line-invisible	yes	yes	no
end-of-line-invisible	no	yes	yes
center-invisible	yes	no	yes

User settable properties:

`break-visibility` (vector)
: A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`extra-spacing-height` (pair of numbers)
: In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers)
: In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`non-musical` (boolean)
: True if the grob belongs to a `NonMusicalPaperColumn`.

This grob interface is added dynamically to grobs of class `Item`.

3.2.63 jump-script-interface

A jump instruction, e.g. D.S.

This grob interface is used in the following graphical object(s): `JumpScript` (page 422).

3.2.64 key-cancellation-interface

A key cancellation.

This grob interface is used in the following graphical object(s): `KeyCancellation` (page 424).

3.2.65 key-signature-interface

A group of accidentals, to be printed as signature sign.

User settable properties:

`alteration-alist` (list)
: List of `(`*`pitch . accidental`*`)` pairs for key signature.

`alteration-glyph-name-alist` (list)
: An alist of key-string pairs.

`flat-positions` (list)
: Flats in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`non-default` (boolean)
: Set for manually specified clefs and keys.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`padding-pairs` (list)
: An alist mapping `(`*`name . name`*`)` to distances.

`sharp-positions` (list)
: Sharps in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

Internal properties:

`c0-position` (integer)
: An integer indicating the position of middle C.

This grob interface is used in the following graphical object(s): `KeyCancellation` (page 424), and `KeySignature` (page 427).

3.2.66 kievan-ligature-interface

A kievan ligature.

User settable properties:

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

Internal properties:

`primitive` (integer)
: A pointer to a ligature primitive, i.e., an item similar to a note head that is part of a ligature.

This grob interface is used in the following graphical object(s): `KievanLigature` (page 430).

3.2.67 `ledger-line-spanner-interface`

This spanner draws the ledger lines of a staff. This is a separate grob because it has to process all potential collisions between all note heads. The thickness of ledger lines is controlled by the `ledger-line-thickness` property of the Section 3.1.116 [`StaffSymbol`], page 480, grob.

User settable properties:

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`minimum-length-fraction` (number)
: Minimum length of ledger line as fraction of note head size.

Internal properties:

`note-heads` (array of grobs)
: An array of note head grobs.

This grob interface is used in the following graphical object(s): `LedgerLineSpanner` (page 432).

3.2.68 `ledgered-interface`

Objects that need ledger lines, typically note heads. See also Section 3.2.67 [`ledger-line-spanner-interface`], page 559.

User settable properties:

`no-ledgers` (boolean)
: If set, don't draw ledger lines on this object.

This grob interface is used in the following graphical object(s): `AmbitusNoteHead` (page 355), `NoteHead` (page 455), and `TrillPitchHead` (page 506).

3.2.69 `ligature-bracket-interface`

A bracket indicating a ligature in the original edition.

User settable properties:

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`width` (dimension, in staff space)
: The width of a grob measured in staff space.

This grob interface is not used in any graphical object.

3.2.70 `ligature-head-interface`

A note head that can become part of a ligature.

This grob interface is used in the following graphical object(s): `NoteHead` (page 455).

3.2.71 `ligature-interface`

A ligature.

This grob interface is not used in any graphical object.

3.2.72 `line-interface`

Generic line objects. Any object using lines supports this. The property `style` can be `line`, `dashed-line`, `trill`, `dotted-line`, `zigzag` or `none` (a transparent line).

For `dashed-line`, the length of the dashes is tuned with `dash-fraction`. If the latter is set to 0, a dotted line is produced.

User settable properties:

`arrow-length` (number)
: Arrow length.

`arrow-width` (number)
: Arrow width.

`dash-fraction` (number)
: Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`dash-period` (number)
: The length of one dash together with whitespace. If negative, no line is drawn at all.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`zigzag-length` (dimension, in staff space)
: The length of the lines of a zigzag, relative to `zigzag-width`. A value of `1` gives 60-degree zigzags.

`zigzag-width` (dimension, in staff space)
: The width of one zigzag squiggle. This number is adjusted slightly so that the spanner line can be constructed from a whole number of squiggles.

This grob interface is used in the following graphical object(s): `DurationLine` (page 399), `DynamicTextSpanner` (page 403), `Episema` (page 405), `Glissando` (page 414), `Hairpin` (page 417), `HorizontalBracket` (page 418), `LigatureBracket` (page 434), `MeasureSpanner` (page 442), `OttavaBracket` (page 457), `PianoPedalBracket` (page 464), `TextSpanner` (page 498), `TrillSpanner` (page 507), `TupletBracket` (page 509), `VoiceFollower` (page 516), `VoltaBracket` (page 517), and `VowelTransition` (page 520).

3.2.73 `line-spanner-interface`

Generic line drawn between two objects, e.g., for use with glissandi.

`bound-details` is a nested alist. It's possible to specify settings for the sub-properties: `left`, `left-broken`, `right` and `right-broken`.

Values for the following keys may be set:

`Y`
: Sets the Y coordinate of the end point, in staff-spaces offset from the staff center line. By default, it is the center of the bound object, so a glissando points to the vertical center of the note head. For horizontal spanners, such as text spanners and trill spanners, it is hardcoded to 0.

`attach-dir`
: 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. Usually computed on the fly.

`stencil`
: Line spanners may have symbols at the beginning or end, which is contained in this sub-property. For internal use.

`text`
: This is a markup that is evaluated to yield the stencil.

`stencil-align-dir-y`
`stencil-offset`
: Without setting one of these, the stencil is simply put at the end-point, centered on the line, as defined by the `X` and `Y` sub-properties. Setting `stencil-align-dir-y` moves the symbol at the edge vertically relative to the end point of the line. With `stencil-offset`, expecting a number pair, the stencil is moved along the X axis according to the first value, the second value moves the stencil along the Y axis.

`arrow`
: Produces an arrowhead at the end-points of the line.

`padding`
: 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.

User settable properties:

`bound-details` (list)
: An alist of properties for determining attachments of spanners to edges.

`extra-dy` (number)
: Slope glissandi this much extra.

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`left-bound-info` (list)
: An alist of properties for determining attachments of spanners to edges.

`right-bound-info` (list)
: An alist of properties for determining attachments of spanners to edges.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`to-barline` (boolean)
: If true, the spanner will stop at the bar line just before it would otherwise stop.

Internal properties:

`note-columns` (array of grobs)
: An array of `NoteColumn` grobs.

This grob interface is used in the following graphical object(s): `BendSpanner` (page 369), `DurationLine` (page 399), `DynamicTextSpanner` (page 403), `Episema` (page 405), `FingerGlideSpanner` (page 406), `Glissando` (page 414), `TextSpanner` (page 498), `TrillSpanner` (page 507), `VoiceFollower` (page 516), and `VowelTransition` (page 520).

3.2.74 `lyric-extender-interface`

The extender is a simple line at the baseline of the lyric that helps show the length of a melisma (a tied or slurred note).

User settable properties:

`left-padding` (dimension, in staff space)
: The amount of space that is put left to an object (e.g., a lyric extender).

`next` (graphical (layout) object)
: Object that is next relation (e.g., the lyric syllable following an extender).

`right-padding` (dimension, in staff space)
: Space to insert on the right side of an object (e.g., between note and its accidentals).

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`heads` (array of grobs)
: An array of note heads.

This grob interface is used in the following graphical object(s): `LyricExtender` (page 436).

3.2.75 `lyric-hyphen-interface`

A centered hyphen is simply a line between lyrics used to divide syllables.

User settable properties:

`dash-period` (number)
: The length of one dash together with whitespace. If negative, no line is drawn at all.

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`length` (dimension, in staff space)
: User override for the stem length of unbeamed stems (each unit represents half a `staff-space`).

`minimum-distance` (dimension, in staff space)
: Minimum distance between rest and notes or beam.

`minimum-length` (dimension, in staff space)
: Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `LyricHyphen` (page 436), and `LyricSpace` (page 437).

3.2.76 `lyric-interface`

Any object that is related to lyrics.

This grob interface is used in the following graphical object(s): `LyricExtender` (page 436), `LyricHyphen` (page 436), and `VowelTransition` (page 520).

3.2.77 `lyric-space-interface`

An invisible object that prevents lyric words from being spaced too closely.

This grob interface is used in the following graphical object(s): `LyricSpace` (page 437).

3.2.78 `lyric-syllable-interface`

A single piece of lyrics.

This grob interface is used in the following graphical object(s): `LyricText` (page 438).

3.2.79 `mark-interface`

A rehearsal mark.

This grob interface is used in the following graphical object(s): `RehearsalMark` (page 465).

3.2.80 `measure-counter-interface`

A counter for numbering measures.

User settable properties:

`count-from` (integer)
: The first measure in a measure count receives this number. The following measures are numbered in increments from this initial value.

`left-number-text` (markup)
: For a measure counter, this is the formatted measure count. When the measure counter extends over several measures (like with compressed multi-measure rests), it is the text on the left side of the dash.

`number-range-separator` (markup)
: For a measure counter extending over several measures (like with compressed multi-measure rests), this is the separator between the two printed numbers.

`right-number-text` (markup)
: When the measure counter extends over several measures (like with compressed multi-measure rests), this is the text on the right side of the dash. Usually unset.

Internal properties:

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

This grob interface is used in the following graphical object(s): `MeasureCounter` (page 439).

3.2.81 measure-grouping-interface

This object indicates groups of beats. Valid choices for `style` are `bracket` and `triangle`.

User settable properties:

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `MeasureGrouping` (page 441).

3.2.82 measure-spanner-interface

A bracket aligned to a measure or measures.

User settable properties:

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`bracket-visibility` (boolean or symbol)
: This controls the visibility of the tuplet bracket. Setting it to false prevents printing of the bracket. Setting the property to `if-no-beam` makes it print only if there is no beam associated with this tuplet bracket.

`connect-to-neighbor` (pair)
: Pair of booleans, indicating whether this grob looks as a continued break.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`spacing-pair` (pair)
: A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.

  For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`staff-padding` (dimension, in staff space)
: Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `MeasureSpanner` (page 442).

3.2.83 `melody-spanner-interface`

Context dependent typesetting decisions.

User settable properties:

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

Internal properties:

`stems` (array of grobs)
: An array of stem objects.

This grob interface is used in the following graphical object(s): `MelodyItem` (page 443).

3.2.84 `mensural-ligature-interface`

A mensural ligature.

User settable properties:

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`add-join` (boolean)
: Is this ligature head-joined with the next one by a vertical line?

`delta-position` (number)
: The vertical position difference.

`flexa-interval` (integer)
: The interval spanned by the two notes of a flexa shape (1 is a second, 7 is an octave).

`head-width` (dimension, in staff space)
: The width of this ligature head.

`ligature-flexa` (boolean)
: request joining note to the previous one in a flexa.

`primitive` (integer)
: A pointer to a ligature primitive, i.e., an item similar to a note head that is part of a ligature.

This grob interface is used in the following graphical object(s): `MensuralLigature` (page 444), and `NoteHead` (page 455).

3.2.85 `metronome-mark-interface`

A metronome mark.

This grob interface is used in the following graphical object(s): `MetronomeMark` (page 444).

3.2.86 `multi-measure-interface`

Multi measure rest, and the text or number that is printed over it.

User settable properties:

`bound-padding` (number)
: The amount of padding to insert around spanner bounds.

This grob interface is used in the following graphical object(s): `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), and `MultiMeasureRestText` (page 450).

3.2.87 multi-measure-rest-interface

A rest that spans a whole number of measures.

User settable properties:

`bound-padding` (number)
: The amount of padding to insert around spanner bounds.

`expand-limit` (integer)
: Maximum number of measures expanded in church rests.

`hair-thickness` (number)
: Thickness of the thin line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *Staff*`.StaffSymbol.thickness`).

`max-symbol-separation` (number)
: The maximum distance between symbols making up a church rest.

`measure-count` (integer)
: The number of measures for a multi-measure rest.

`minimum-length` (dimension, in staff space)
: Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`round-up-exceptions` (list)
: A list of pairs where car is the numerator and cdr the denominator of a moment. Each pair in this list means that the multi-measure rests of the corresponding length will be rounded up to the longer rest. See *round-up-to-longer-rest*.

`round-up-to-longer-rest` (boolean)
: Displays the longer multi-measure rest when the length of a measure is between two values of `usable-duration-logs`. For example, displays a breve instead of a whole in a 3/2 measure.

`spacing-pair` (pair)
: A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.

  For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`thick-thickness` (number)
: Thickness of the thick line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *Staff*`.StaffSymbol.thickness`).

`usable-duration-logs` (list)
: List of `duration-log`s that can be used in typesetting the grob.

Internal properties:

`space-increment` (dimension, in staff space)
: The amount by which the total duration of a multimeasure rest affects horizontal spacing. Each doubling of the duration adds `space-increment` to the length of the bar.

This grob interface is used in the following graphical object(s): `MultiMeasureRest` (page 446), and `PercentRepeat` (page 460).

3.2.88 `multi-measure-rest-number-interface`

Multi measure rest number that is printed over a rest.

This grob interface is used in the following graphical object(s): `MultiMeasureRestNumber` (page 447).

3.2.89 `note-collision-interface`

An object that handles collisions between notes with different stem directions and horizontal shifts. Most of the interesting properties are to be set in Section 3.2.90 [`note-column-interface`], page 568: these are `force-hshift` and `horizontal-shift`.

User settable properties:

`merge-differently-dotted` (boolean)
: Merge note heads in collisions, even if they have a different number of dots. This is normal notation for some types of polyphonic music.

  `merge-differently-dotted` only applies to opposing stem directions (i.e., voice 1 & 2).

`merge-differently-headed` (boolean)
: Merge note heads in collisions, even if they have different note heads. The smaller of the two heads is rendered invisible. This is used in polyphonic guitar notation. The value of this setting is used by Section "note-collision-interface" in *Internals Reference*.

  `merge-differently-headed` only applies to opposing stem directions (i.e., voice 1 & 2).

`note-collision-threshold` (dimension, in staff space)
: Simultaneous notes that are this close or closer in units of `staff-space` will be identified as vertically colliding. Used by `Stem` grobs for notes in the same voice, and `NoteCollision` grobs for notes in different voices. Default value 1.

`prefer-dotted-right` (boolean)
: For note collisions, prefer to shift dotted up-note to the right, rather than shifting just the dot.

Internal properties:

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): `NoteCollision` (page 453).

3.2.90 `note-column-interface`

Stem and noteheads combined.

User settable properties:

`force-hshift` (number)
: This specifies a manual shift for notes in collisions. The unit is the note head width of the first voice note. This is used by Section "note-collision-interface" in *Internals Reference*.

`glissando-skip` (boolean)
: Should this `NoteHead` be skipped by glissandi?

`horizontal-shift` (integer)
: An integer that identifies ranking of `NoteColumn`s for horizontal shifting. This is used by Section "note-collision-interface" in *Internals Reference*.

`ignore-collision` (boolean)
: If set, don't do note collision resolution on this `NoteColumn`.

`main-extent` (pair of numbers)
: The horizontal extent of a `NoteColumn` grob without taking suspended `NoteHead` grobs into account (i.e., `NoteHead`s forced into the unnatural direction of the `Stem` because of a chromatic clash).

Internal properties:

`note-heads` (array of grobs)
: An array of note head grobs.

`rest` (graphical (layout) object)
: A pointer to a `Rest` object.

`rest-collision` (graphical (layout) object)
: A rest collision that a rest is in.

`stem` (graphical (layout) object)
: A pointer to a `Stem` object.

This grob interface is used in the following graphical object(s): `NoteColumn` (page 454).

3.2.91 `note-head-interface`

A note head. There are many possible values for `style`. For a complete list, see Section "Note head styles" in *Notation Reference*.

User settable properties:

`duration-log` (integer)
: The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`glyph-name` (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`ignore-ambitus` (boolean)
: If set, don't consider this notehead for ambitus calculation.

`ledger-positions` (list)
: Vertical positions of ledger lines. When set on a `StaffSymbol` grob it defines a repeating pattern of ledger lines and any parenthesized groups will always be shown together.

`note-names` (vector)
: Vector of strings containing names for easy-notation note heads.

`stem-attachment` (pair of numbers)
: An `(x . y)` pair where the stem attaches to the notehead.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

Internal properties:

`accidental-grob` (graphical (layout) object)
: The accidental for this note.

This grob interface is used in the following graphical object(s): `AmbitusNoteHead` (page 355), `NoteHead` (page 455), and `TabNoteHead` (page 494).

3.2.92 note-name-interface

Note names.

This grob interface is used in the following graphical object(s): `NoteName` (page 456).

3.2.93 note-spacing-interface

This object calculates spacing wishes for individual voices.

User settable properties:

`knee-spacing-correction` (number)
: Factor for the optical correction amount for kneed beams. Set between `0` for no correction and `1` for full correction.

`same-direction-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. This amount is used for stems with the same direction to compensate for note head to stem distance.

`space-to-barline` (boolean)
: If set, the distance between a note and the following non-musical column will be measured to the bar line instead of to the beginning of the non-musical column. If there is a clef change followed by a bar line, for example, this means that we will try to space the non-musical column as though the clef is not there.

`stem-spacing-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. For opposite directions, this amount is the correction for two normal sized stems that overlap completely.

Internal properties:

`left-items` (array of grobs)
: Grobs organized on the left by a spacing object.

`right-items` (array of grobs)
: Grobs organized on the right by a spacing object.

This grob interface is used in the following graphical object(s): `NoteSpacing` (page 456).

3.2.94 number-interface

Numbers.

User settable properties:

`number-type` (symbol)
: Numbering style. Choices include `roman-lower`, `roman-upper` and `arabic`.

This grob interface is used in the following graphical object(s): `StringNumber` (page 485).

3.2.95 only-prebreak-interface

Kill this grob after the line breaking process.

This grob interface is not used in any graphical object.

3.2.96 ottava-bracket-interface

An ottava bracket.

User settable properties:

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`minimum-length` (dimension, in staff space)
: Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

This grob interface is used in the following graphical object(s): `OttavaBracket` (page 457).

3.2.97 outside-staff-axis-group-interface

A vertical axis group on which outside-staff skyline calculations are done.

User settable properties:

`outside-staff-placement-directive` (symbol)
: One of four directives telling how outside staff objects should be placed.
  - `left-to-right-greedy` – Place each successive grob from left to right.
  - `left-to-right-polite` – Place a grob from left to right only if it does not potentially overlap with another grob that has been placed on a pass through a grob array. If there is overlap, do another pass to determine placement.
  - `right-to-left-greedy` – Same as `left-to-right-greedy`, but from right to left.
  - `right-to-left-polite` – Same as `left-to-right-polite`, but from right to left.

Internal properties:

`vertical-skyline-elements` (array of grobs)
: An array of grobs used to create vertical skylines.

This grob interface is used in the following graphical object(s): `BassFigureLine` (page 366), `System` (page 490), and `VerticalAxisGroup` (page 514).

3.2.98 outside-staff-interface

A grob that could be placed outside staff.

User settable properties:

`outside-staff-horizontal-padding` (number)
: By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-padding` (number)
: The padding to place between grobs when spacing according to `outside-staff-priority`. Two grobs with different `outside-staff-padding` values have the larger value of padding between them.

`outside-staff-priority` (number)
: If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

This grob interface is used in the following graphical object(s): `AccidentalSuggestion` (page 350), `BarNumber` (page 361), `BassFigureAlignmentPositioning` (page 364), `BendSpanner` (page 369), `BreathingSign` (page 373), `CenteredBarNumberLineSpanner` (page 376), `ChordName` (page 377), `ClefModifier` (page 381), `CombineTextScript` (page 383), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `DynamicLineSpanner` (page 401), `DynamicText` (page 402), `Fingering` (page 408), `FretBoard` (page 412), `Hairpin` (page 417), `HorizontalBracket` (page 418), `HorizontalBracketText` (page 419), `InstrumentSwitch` (page 421), `JumpScript` (page 422), `MeasureCounter` (page 439), `MeasureGrouping` (page 441), `MeasureSpanner` (page 442), `MetronomeMark` (page 444), `MultiMeasureRest` (page 446), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `OttavaBracket` (page 457), `PercentRepeatCounter` (page 461), `PhrasingSlur` (page 462), `RehearsalMark` (page 465), `Script` (page 470), `Slur` (page 472), `SostenutoPedalLineSpanner` (page 475), `StringNumber` (page 485), `StrokeFinger` (page 487), `SustainPedalLineSpanner` (page 489), `TextScript` (page 496), `TextSpanner` (page 498), `TrillSpanner` (page 507), `TupletBracket` (page 509), `TupletNumber` (page 510), `UnaCordaPedalLineSpanner` (page 512), and `VoltaBracketSpanner` (page 518).

3.2.99 paper-column-interface

`Paper_column` objects form the top-most X parents for items. There are two types of columns: musical and non-musical, to which musical and non-musical objects are attached respectively. The spacing engine determines the X positions of these objects.

They are numbered, the first (leftmost) is column 0. Numbering happens before line breaking, and columns are not renumbered after line breaking. Since many columns go unused, you should only use the rank field to get ordering information. Two adjacent columns may have non-adjacent numbers.

The `paper-column-interface` implies the `item-interface` (page 557).

User settable properties:

`between-cols` (pair)
: Where to attach a loose column to.

`full-measure-extra-space` (number)
: Extra space that is allocated at the beginning of a measure with only one note. This property is read from the NonMusicalPaperColumn that begins the measure.

`labels` (list)
: List of labels (symbols) placed on a column.

`line-break-penalty` (number)
: Penalty for a line break at this column. This affects the choices of the line breaker; it avoids a line break at a column with a positive penalty and prefers a line break at a column with a negative penalty.

`line-break-permission` (symbol)
: Instructs the line breaker on whether to put a line break at this column. Can be `force` or `allow`.

`line-break-system-details` (list)
: An alist of properties to use if this column is the start of a system.

`page-break-penalty` (number)
: Penalty for page break at this column. This affects the choices of the page breaker; it avoids a page break at a column with a positive penalty and prefers a page break at a column with a negative penalty.

`page-break-permission` (symbol)
: Instructs the page breaker on whether to put a page break at this column. Can be `force` or `allow`.

`page-turn-penalty` (number)
: Penalty for a page turn at this column. This affects the choices of the page breaker; it avoids a page turn at a column with a positive penalty and prefers a page turn at a column with a negative penalty.

`page-turn-permission` (symbol)
: Instructs the page breaker on whether to put a page turn at this column. Can be `force` or `allow`.

`rhythmic-location` (rhythmic location)
: Where (bar number, measure position) in the score.

`shortest-playing-duration` (moment)
: The duration of the shortest note playing here.

`shortest-starter-duration` (moment)
: The duration of the shortest note that starts here.

`used` (boolean)
: If set, this spacing column is kept in the spacing problem.

`when` (moment)
: Global time step associated with this column.

Internal properties:

`bounded-by-me` (array of grobs)
: An array of spanners that have this column as start/begin point. Only columns that have grobs or act as bounds are spaced.

`grace-spacing` (graphical (layout) object)
: A run of grace notes.

`maybe-loose` (boolean)
: Used to mark a breakable column that is loose if and only if it is in the middle of a line.

`spacing` (graphical (layout) object)
: The spacing spanner governing this section.

This grob interface is added dynamically to grobs of class `Paper_column`.

3.2.100 parentheses-interface

Parentheses for other objects.

User settable properties:

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`stencils` (list)
: Multiple stencils, used as intermediate value.

This grob interface is used in the following graphical object(s): `Parentheses` (page 459), and `TrillPitchParentheses` (page 507).

3.2.101 percent-repeat-interface

Beat, Double and single measure repeats.

User settable properties:

`dot-negative-kern` (number)
: The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

`slash-negative-kern` (number)
: The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

`slope` (number)
: The slope of this object.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), `PercentRepeat` (page 460), `PercentRepeatCounter` (page 461), and `RepeatSlash` (page 467).

3.2.102 percent-repeat-item-interface

Repeats that look like percent signs.

User settable properties:

`dot-negative-kern` (number)
: The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

`slash-negative-kern` (number)
: The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

`slope` (number)
: The slope of this object.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `DoublePercentRepeat` (page 395), `DoublePercentRepeatCounter` (page 396), `DoubleRepeatSlash` (page 398), and `RepeatSlash` (page 467).

3.2.103 `piano-pedal-bracket-interface`

The bracket of the piano pedal. It can be tuned through the regular bracket properties.

User settable properties:

`bound-padding` (number)
: The amount of padding to insert around spanner bounds.

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

Internal properties:

`pedal-text` (graphical (layout) object)
: A pointer to the text of a mixed-style piano pedal.

This grob interface is used in the following graphical object(s): `PianoPedalBracket` (page 464).

3.2.104 `piano-pedal-interface`

A piano pedal sign.

This grob interface is used in the following graphical object(s): `PianoPedalBracket` (page 464), `SostenutoPedalLineSpanner` (page 475), `SustainPedal` (page 488), `SustainPedalLineSpanner` (page 489), and `UnaCordaPedalLineSpanner` (page 512).

3.2.105 piano-pedal-script-interface

A piano pedal sign, fixed size.

This grob interface is used in the following graphical object(s): SostenutoPedal (page 474), SustainPedal (page 488), and UnaCordaPedal (page 511).

3.2.106 pitched-trill-interface

A note head to indicate trill pitches.

Internal properties:

accidental-grob (graphical (layout) object)
: The accidental for this note.

This grob interface is used in the following graphical object(s): TrillPitchHead (page 506), and TrillPitchParentheses (page 507).

3.2.107 pure-from-neighbor-interface

A collection of routines to allow for objects' pure heights and heights to be calculated based on the heights of the objects' neighbors.

Internal properties:

neighbors (array of grobs)
: The X-axis neighbors of a grob. Used by the pure-from-neighbor-interface to determine various grob heights.

pure-relevant-grobs (array of grobs)
: All the grobs (items and spanners) that are relevant for finding the pure-Y-extent

pure-Y-common (graphical (layout) object)
: A cache of the common_refpoint_of_array of the elements grob set.

This grob interface is used in the following graphical object(s): BarLine (page 358), Clef (page 378), CueClef (page 387), CueEndClef (page 389), KeyCancellation (page 424), KeySignature (page 427), SpanBarStub (page 478), and TimeSignature (page 502).

3.2.108 rest-collision-interface

Move ordinary rests (not multi-measure nor pitched rests) to avoid conflicts.

User settable properties:

minimum-distance (dimension, in staff space)
: Minimum distance between rest and notes or beam.

Internal properties:

elements (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

positioning-done (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

This grob interface is used in the following graphical object(s): RestCollision (page 470).

3.2.109 rest-interface

A rest symbol. The property style can be default, mensural, neomensural or classical.

User settable properties:

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`minimum-distance` (dimension, in staff space)
: Minimum distance between rest and notes or beam.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`voiced-position` (number)
: The staff-position of a voiced `Rest`, negative if the rest has `direction DOWN`.

This grob interface is used in the following graphical object(s): `MultiMeasureRest` (page 446), and `Rest` (page 469).

3.2.110 rhythmic-grob-interface

Any object with a duration. Used to determine which grobs are interesting enough to maintain a hara-kiri staff.

This grob interface is used in the following graphical object(s): `BassFigure` (page 363), `ChordName` (page 377), `ClusterSpannerBeacon` (page 383), `DoubleRepeatSlash` (page 398), `FretBoard` (page 412), `LyricText` (page 438), `NoteHead` (page 455), `RepeatSlash` (page 467), `Rest` (page 469), and `TabNoteHead` (page 494).

3.2.111 rhythmic-head-interface

Note head or rest.

User settable properties:

`duration-log` (integer)
: The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`glissando-skip` (boolean)
: Should this `NoteHead` be skipped by glissandi?

Internal properties:

`dot` (graphical (layout) object)
: A reference to a `Dots` object.

`stem` (graphical (layout) object)
: A pointer to a `Stem` object.

This grob interface is used in the following graphical object(s): `AmbitusNoteHead` (page 355), `NoteHead` (page 455), `Rest` (page 469), `TabNoteHead` (page 494), and `TrillPitchHead` (page 506).

3.2.112 script-column-interface

An interface that sorts scripts according to their `script-priority` and `outside-staff-priority`.

Internal properties:

`scripts` (array of grobs)
: An array of `Script` objects.

This grob interface is used in the following graphical object(s): `ScriptColumn` (page 471), and `ScriptRow` (page 472).

3.2.113 `script-interface`

An object that is put above or below a note.

User settable properties:

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`script-priority` (number)
: A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`side-relative-direction` (direction)
: Multiply direction of `direction-source` with this to get the direction of this object.

`slur-padding` (number)
: Extra distance between slur and script.

`toward-stem-shift` (number)
: Amount by which scripts are shifted toward the stem if their direction coincides with the stem direction. `0.0` means centered on the note head (the default position of most scripts); `1.0` means centered on the stem. Interpolated values are possible.

`toward-stem-shift-in-column` (number)
: Amount by which a script is shifted toward the stem if its direction coincides with the stem direction and it is associated with a `ScriptColumn` object. `0.0` means centered on the note head (the default position of most scripts); `1.0` means centered on the stem. Interpolated values are possible.

Internal properties:

`direction-source` (graphical (layout) object)
: In case `side-relative-direction` is set, which grob to get the direction from.

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

`script-column` (graphical (layout) object)
: A `ScriptColumn` associated with a `Script` object.

`script-stencil` (pair)
: A pair `(type . arg)` which acts as an index for looking up a `Stencil` object.

`slur` (graphical (layout) object)
: A pointer to a `Slur` object.

This grob interface is used in the following graphical object(s): `AccidentalSuggestion` (page 350), `DynamicText` (page 402), `MultiMeasureRestScript` (page 449), and `Script` (page 470).

3.2.114 self-alignment-interface

Position this object on itself and/or on its parent. To this end, the following functions are provided:

`Self_alignment_interface::[xy]_aligned_on_self`
: Align self on reference point, using `self-alignment-X` and `self-alignment-Y`.

`Self_alignment_interface::aligned_on_[xy]_parent`
`Self_alignment_interface::centered_on_[xy]_parent`
: Shift the object so its own reference point is centered on the extent of the parent

User settable properties:

`parent-alignment-X` (number)
: Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`parent-alignment-Y` (number)
: Like `parent-alignment-X` but for the Y axis.

`self-alignment-X` (number)
: Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number)
: Like `self-alignment-X` but for the Y axis.

`X-align-on-main-noteheads` (boolean)
: If true, this grob will ignore suspended noteheads when aligning itself on NoteColumn.

This grob interface is used in the following graphical object(s): `AccidentalSuggestion` (page 350), `BarNumber` (page 361), `ClefModifier` (page 381), `CombineTextScript` (page 383), `DoublePercentRepeatCounter` (page 396), `DynamicText` (page 402), `Fingering` (page 408), `GridLine` (page 416), `Hairpin` (page 417), `HorizontalBracketText` (page 419), `InstrumentName` (page 420), `InstrumentSwitch` (page 421), `JumpScript` (page 422), `LyricText` (page 438), `MeasureCounter` (page 439), `MeasureSpanner` (page 442), `MetronomeMark` (page 444), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `PercentRepeatCounter` (page 461), `RehearsalMark` (page 465), `Script` (page 470), `SostenutoPedal` (page 474), `StemTremolo` (page 484), `StringNumber` (page 485), `StrokeFinger` (page 487), `SustainPedal` (page 488), `TextScript` (page 496), and `UnaCordaPedal` (page 511).

3.2.115 semi-tie-column-interface

The interface for a column of l.v. (laissez vibrer) ties.

User settable properties:

`head-direction` (direction)
: Are the note heads left or right in a semitie?

`tie-configuration` (list)
: List of (*position . dir*) pairs, indicating the desired tie configuration, where *position* is the offset from the center of the staff in staff space and *dir* indicates the direction of the tie (`1`=>up, `-1`=>down, `0`=>center). A non-pair entry in the list causes the corresponding tie to be formatted automatically.

Internal properties:

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

`ties` (array of grobs)
: A grob array of `Tie` objects.

This grob interface is used in the following graphical object(s): `LaissezVibrerTieColumn` (page 431), and `RepeatTieColumn` (page 469).

3.2.116 semi-tie-interface

A tie which is only connected to a note head on one side. The following properties may be set in the `details` list:

`height-limit`
: Maximum tie height: The longer the tie, the closer it is to this height.

`ratio`
: Parameter for tie shape. The higher this number, the quicker the tie attains its `height-limit`.

User settable properties:

`control-points` (list of number pairs)
: List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`head-direction` (direction)
: Are the note heads left or right in a semitie?

`line-thickness` (number)
: For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.`StaffSymbol.thickness`).

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`annotation` (string)
: Annotate a grob for debug purposes.

`note-head` (graphical (layout) object)
: A single note head.

This grob interface is used in the following graphical object(s): `LaissezVibrerTie` (page 430), and `RepeatTie` (page 467).

3.2.117 `separation-item-interface`

Item that computes widths to generate spacing rods.

User settable properties:

`horizontal-skylines` (pair of skylines)
: Two skylines, one to the left and one to the right of this grob.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`skyline-vertical-padding` (number)
: The amount by which the left and right skylines of a column are padded vertically, beyond the `Y-extent`s and `extra-spacing-height`s of the constituent grobs in the column. Increase this to prevent interleaving of grobs from adjacent columns.

`X-extent` (pair of numbers)
: Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

Internal properties:

`conditional-elements` (array of grobs)
: Internal use only.

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

This grob interface is used in the following graphical object(s): `NonMusicalPaperColumn` (page 452), `NoteColumn` (page 454), and `PaperColumn` (page 458).

3.2.118 `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?)

The routine also takes the size of the staff into account if `staff-padding` is set. If undefined, the staff symbol is ignored.

User settable properties:

`add-stem-support` (boolean)
: If set, the `Stem` object is included in this script's support.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`horizon-padding` (number)
: The amount to pad the axis along which a `Skyline` is built for the `side-position-interface`.

`minimum-space` (dimension, in staff space)
: Minimum distance that the victim should move (after padding).

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`side-axis` (number)
: If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`slur-padding` (number)
: Extra distance between slur and script.

`staff-padding` (dimension, in staff space)
: Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`use-skylines` (boolean)
: Should skylines be used for side positioning?

Internal properties:

`quantize-position` (boolean)
: If set, a vertical alignment is aligned to be within staff spaces.

`side-support-elements` (array of grobs)
: The side support, an array of grobs.

This grob interface is used in the following graphical object(s): `AccidentalSuggestion` (page 350), `Arpeggio` (page 355), `BarNumber` (page 361), `BassFigureAlignmentPositioning` (page 364), `CenteredBarNumberLineSpanner` (page 376), `ClefModifier` (page 381), `CombineTextScript` (page 383), `DoublePercentRepeatCounter` (page 396), `DynamicLineSpanner` (page 401), `Episema` (page 405), `Fingering` (page 408), `HorizontalBracket` (page 418), `HorizontalBracketText` (page 419), `InstrumentName` (page 420), `InstrumentSwitch` (page 421), `JumpScript` (page 422), `MeasureCounter` (page 439), `MeasureGrouping` (page 441), `MeasureSpanner` (page 442), `MetronomeMark` (page 444), `MultiMeasureRestNumber` (page 447), `MultiMeasureRestScript` (page 449), `MultiMeasureRestText` (page 450), `OttavaBracket` (page 457), `PercentRepeatCounter` (page 461), `RehearsalMark` (page 465), `Script` (page 470), `SostenutoPedalLineSpanner` (page 475), `StanzaNumber` (page 481), `StringNumber` (page 485), `StrokeFinger` (page 487), `SustainPedalLineSpanner` (page 489), `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), `SystemStartSquare` (page 494), `TextScript` (page 496), `TextSpanner` (page 498), `TrillPitchAccidental` (page 504), `TrillPitchGroup` (page 505), `TrillSpanner` (page 507), `UnaCordaPedalLineSpanner` (page 512), `VoltaBracket` (page 517), and `VoltaBracketSpanner` (page 518).

3.2.119 slur-interface

A slur. Slurs are formatted by trying a number of combinations of left/right end point, and then picking the slur with the lowest demerit score. The combinations are generated by going from the base attachments (i.e., note heads) in the direction in half space increments until we have covered `region-size` staff spaces. The following properties may be set in the `details` list.

`region-size`
: Size of region (in staff spaces) for determining potential endpoints in the Y direction.

`head-encompass-penalty`
: Demerit to apply when note heads collide with a slur.

`stem-encompass-penalty`
: Demerit to apply when stems collide with a slur.

`edge-attraction-factor`
: Factor used to calculate the demerit for distances between slur endpoints and their corresponding base attachments.

`same-slope-penalty`
: Demerit for slurs with attachment points that are horizontally aligned.

`steeper-slope-factor`
: Factor used to calculate demerit only if this slur is not broken.

`non-horizontal-penalty`
: Demerit for slurs with attachment points that are not horizontally aligned.

`max-slope`
: The maximum slope allowed for this slur.

`max-slope-factor`
: Factor that calculates demerit based on the max slope.

`free-head-distance`
: The amount of vertical free space that must exist between a slur and note heads.

`absolute-closeness-measure`
: Factor to calculate demerit for variance between a note head and slur.

`extra-object-collision-penalty`
: Factor to calculate demerit for extra objects that the slur encompasses, including accidentals, fingerings, and tuplet numbers.

`accidental-collision`
: Factor to calculate demerit for `Accidental` objects that the slur encompasses. This property value replaces the value of `extra-object-collision-penalty`.

`extra-encompass-free-distance`
: The amount of vertical free space that must exist between a slur and various objects it encompasses, including accidentals, fingerings, and tuplet numbers.

`extra-encompass-collision-distance`
: This detail is currently unused.

`head-slur-distance-factor`
: Factor to calculate demerit for variance between a note head and slur.

`head-slur-distance-max-ratio`
: The maximum value for the ratio of distance between a note head and slur.

`gap-to-staffline-inside`
: Minimum gap inside the curve of the slur where the slur is parallel to a staffline.

`gap-to-staffline-outside`
: Minimum gap outside the curve of the slur where the slur is parallel to a staffline.

`free-slur-distance`
: The amount of vertical free space that must exist between adjacent slurs. This subproperty only works for `PhrasingSlur`.

`edge-slope-exponent`
: Factor used to calculate the demerit for the slope of a slur near its endpoints; a larger value yields a larger demerit.

User settable properties:

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`control-points` (list of number pairs)
: List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`dash-definition` (pair)
: List of `dash-elements` defining the dash structure. Each `dash-element` has a starting t value, an ending t-value, a `dash-fraction`, and a `dash-period`.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`eccentricity` (number)
: How asymmetrical to make a slur. Positive means move the center to the right.

`height-limit` (dimension, in staff space)
: Maximum slur height: The longer the slur, the closer it is to this height.

`inspect-quants` (pair of numbers)
: If debugging is set, set beam and slur position to a (quantized) position that is as close as possible to this value, and print the demerits for the inspected position in the output.

`line-thickness` (number)
: For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property

is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

`positions` (pair of numbers)
: Pair of staff coordinates (*start* . *end*), where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`ratio` (number)
: Parameter for slur shape. The higher this number, the quicker the slur attains its `height-limit`.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *Staff*.StaffSymbol.thickness).

Internal properties:

`annotation` (string)
: Annotate a grob for debug purposes.

`encompass-objects` (array of grobs)
: Objects that a slur should avoid in addition to notes and stems.

`note-columns` (array of grobs)
: An array of `NoteColumn` grobs.

This grob interface is used in the following graphical object(s): `PhrasingSlur` (page 462), and `Slur` (page 472).

3.2.120 spaceable-grob-interface

A layout object that takes part in the spacing problem.

User settable properties:

`allow-loose-spacing` (boolean)
: If set, column can be detached from main spacing.

`keep-inside-line` (boolean)
: If set, this column cannot have objects sticking into the margin.

`measure-length` (moment)
: Length of a measure. Used in some spacing situations.

Internal properties:

`ideal-distances` (list)
: (*obj* . (*dist* . *strength*)) pairs.

`left-neighbor` (graphical (layout) object)
: The right-most column that has a spacing-wish for this column.

`minimum-distances` (list)
: A list of rods that have the format (*obj* . *dist*).

`right-neighbor` (graphical (layout) object)
: See `left-neighbor`.

`spacing-wishes` (array of grobs)
: An array of note spacing or staff spacing objects.

This grob interface is used in the following graphical object(s): `NonMusicalPaperColumn` (page 452), and `PaperColumn` (page 458).

3.2.121 spacing-interface

This object calculates the desired and minimum distances between two columns.

Internal properties:

`left-items` (array of grobs)
: Grobs organized on the left by a spacing object.

`right-items` (array of grobs)
: Grobs organized on the right by a spacing object.

This grob interface is used in the following graphical object(s): `NoteSpacing` (page 456), and `StaffSpacing` (page 479).

3.2.122 spacing-options-interface

Supports setting of spacing variables.

User settable properties:

`shortest-duration-space` (number)
: Start with this multiple of `spacing-increment` space for the shortest duration. See also Section "spacing-spanner-interface" in *Internals Reference*.

`spacing-increment` (dimension, in staff space)
: The unit of length for note-spacing. Typically, the width of a note head. See also Section "spacing-spanner-interface" in *Internals Reference*.

This grob interface is used in the following graphical object(s): `GraceSpacing` (page 415), and `SpacingSpanner` (page 476).

3.2.123 spacing-spanner-interface

The space taken by a note is dependent on its duration. Doubling a duration adds `spacing-increment` to the space. The most common shortest note gets `shortest-duration-space`. Notes that are even shorter are spaced proportonial to their duration.

Typically, the increment is the width of a black note head. In a piece with lots of 8th notes, and some 16th notes, the eighth note gets a 2 note heads width (i.e., the space following a note is a 1 note head width). A 16th note is followed by 0.5 note head width. The quarter note is followed by 3 NHW, the half by 4 NHW, etc.

User settable properties:

`average-spacing-wishes` (boolean)
: If set, the spacing wishes are averaged over staves.

`base-shortest-duration` (moment)
: Spacing is based on the shortest notes in a piece. Normally, pieces are spaced as if notes at least as short as this are present.

`common-shortest-duration` (moment)
: The most common shortest note length. This is used in spacing. Enlarging this sets the score tighter.

`packed-spacing` (boolean)
: If set, the notes are spaced as tightly as possible.

`shortest-duration-space` (number)
: Start with this multiple of `spacing-increment` space for the shortest duration. See also Section "spacing-spanner-interface" in *Internals Reference*.

`spacing-increment` (dimension, in staff space)
: The unit of length for note-spacing. Typically, the width of a note head. See also Section "spacing-spanner-interface" in *Internals Reference*.

`strict-grace-spacing` (boolean)
: If set, main notes are spaced normally, then grace notes are put left of the musical columns for the main notes.

`strict-note-spacing` (boolean)
: If set, unbroken columns with non-musical material (clefs, bar lines, etc.) are not spaced separately, but put before musical columns.

`uniform-stretching` (boolean)
: If set, items stretch proportionally to their natural separation based on durations. This looks better in complex polyphonic patterns.

This grob interface is used in the following graphical object(s): `SpacingSpanner` (page 476).

3.2.124 span-bar-interface

A bar line that is spanned between other bar lines. This interface is used for bar lines that connect different staves.

User settable properties:

`glyph-name` (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

Internal properties:

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

`pure-relevant-grobs` (array of grobs)
: All the grobs (items and spanners) that are relevant for finding the `pure-Y-extent`

`pure-relevant-items` (array of grobs)
: A subset of elements that are relevant for finding the `pure-Y-extent`.

`pure-relevant-spanners` (array of grobs)
: A subset of elements that are relevant for finding the `pure-Y-extent`.

`pure-Y-common` (graphical (layout) object)
: A cache of the `common_refpoint_of_array` of the `elements` grob set.

This grob interface is used in the following graphical object(s): `SpanBar` (page 477).

3.2.125 spanner-interface

Some objects are horizontally spanned between objects. For example, slurs, beams, ties, etc. These grobs form a subtype called `Spanner`. All spanners have two span points (these must be `Item` objects), one on the left and one on the right. The left bound is also the X reference point of the spanner.

User settable properties:

`minimum-length` (dimension, in staff space)
: Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`minimum-length-after-break` (dimension, in staff space)
: If set, try to make a broken spanner starting a line this long. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance to the notehead.

`normalized-endpoints` (pair)
: Represents left and right placement over the total spanner, where the width of the spanner is normalized between 0 and 1.

`spanner-id` (index or symbol)
: An identifier to distinguish concurrent spanners.

`to-barline` (boolean)
: If true, the spanner will stop at the bar line just before it would otherwise stop.

Internal properties:

`spanner-broken` (boolean)
: Indicates whether spanner alignment should be broken after the current spanner.

This grob interface is added dynamically to grobs of class `Spanner`.

3.2.126 staff-grouper-interface

A grob that collects staves together.

User settable properties:

`staff-staff-spacing` (list)
: When applied to a staff-group's `StaffGrouper` grob, this spacing alist controls the distance between consecutive staves within the staff-group. When applied to a staff's `VerticalAxisGroup` grob, it controls the distance between the staff and the nearest staff below it in the same system, replacing any settings inherited from the `StaffGrouper` grob of the containing staff-group, if there is one. This property remains in effect even when non-staff lines appear between staves. The alist can contain the following 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.
  - `minimum-distance` – the smallest allowable vertical distance, measured in staff-spaces, between the reference points of the two items, when compressing is in effect.

- `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).

`staffgroup-staff-spacing` (list)
: The spacing alist controlling 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 exist between the two staves. If the `staff-staff-spacing` property of the staff's `VerticalAxisGroup` grob is set, that is used instead. See `staff-staff-spacing` for a description of the alist structure.

This grob interface is used in the following graphical object(s): `StaffGrouper` (page 479).

3.2.127 staff-spacing-interface

This object calculates spacing details from a breakable symbol (left) to another object. For example, it takes care of optical spacing from a bar line to a note.

User settable properties:

`stem-spacing-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. For opposite directions, this amount is the correction for two normal sized stems that overlap completely.

This grob interface is used in the following graphical object(s): `StaffSpacing` (page 479).

3.2.128 staff-symbol-interface

This spanner draws the lines of a staff. A staff symbol defines a vertical unit, the *staff space*. Quantities that go by a half staff space are called *positions*. The center (i.e., middle line or space) is position 0. The length of the symbol may be set by hand through the `width` property.

User settable properties:

`break-align-symbols` (list)
: A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`ledger-extra` (dimension, in staff space)
: Extra distance from staff line to draw ledger lines for.

`ledger-line-thickness` (pair of numbers)
: The thickness of ledger lines. It is the sum of 2 numbers: The first is the factor for line thickness, and the second for staff space. Both contributions are added.

`ledger-positions` (list)
: Vertical positions of ledger lines. When set on a `StaffSymbol` grob it defines a repeating pattern of ledger lines and any parenthesized groups will always be shown together.

`ledger-positions-function` (any type)
: A quoted Scheme procedure that takes a `StaffSymbol` grob and the vertical position of a note head as arguments and returns a list of ledger line positions.

`line-count` (integer)
: The number of staff lines.

`line-positions` (list)
: Vertical positions of staff lines.

`staff-space` (dimension, in staff space)
: Amount of space between staff lines, expressed in global `staff-space`.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`width` (dimension, in staff space)
: The width of a grob measured in staff space.

This grob interface is used in the following graphical object(s): `StaffSymbol` (page 480).

3.2.129 `staff-symbol-referencer-interface`

An object whose Y position is meant relative to a staff symbol. These usually have `Staff_symbol_referencer::callback` in their `Y-offset-callbacks`.

User settable properties:

`staff-position` (number)
: Vertical position, measured in half staff spaces, counted from the middle line.

This grob interface is used in the following graphical object(s): `AmbitusNoteHead` (page 355), `Arpeggio` (page 355), `Beam` (page 366), `Clef` (page 378), `CueClef` (page 387), `CueEndClef` (page 389), `Custos` (page 392), `Dots` (page 394), `KeyCancellation` (page 424), `KeySignature` (page 427), `MultiMeasureRest` (page 446), `NoteHead` (page 455), `Rest` (page 469), `TabNoteHead` (page 494), and `TrillPitchHead` (page 506).

3.2.130 `stanza-number-interface`

A stanza number, to be put in from of a lyrics line.

This grob interface is used in the following graphical object(s): `StanzaNumber` (page 481).

3.2.131 `stem-interface`

The stem represents the graphical stem. In addition, it internally connects note heads, beams, and tremolos. Rests and whole notes have invisible stems.

The following properties may be set in the `details` list.

`beamed-lengths`
: List of stem lengths given beam multiplicity.

`beamed-minimum-free-lengths`
: List of normal minimum free stem lengths (chord to beams) given beam multiplicity.

`beamed-extreme-minimum-free-lengths`
: List of extreme minimum free stem lengths (chord to beams) given beam multiplicity.

`lengths` Default stem lengths. The list gives a length for each flag count.

`stem-shorten`
: How much a stem in a forced direction should be shortened. The list gives an amount depending on the number of flags and beams.

User settable properties:

`avoid-note-head` (boolean)
: If set, the stem of a chord does not pass through all note heads, but starts at the last note head.

`beaming` (pair)
: Pair of number lists. Each number list specifies which beams to make. `0` is the central beam, `1` is the next beam toward the note, etc. This information is used to determine how to connect the beaming patterns from stem to stem inside a beam.

`beamlet-default-length` (pair)
: A pair of numbers. The first number specifies the default length of a beamlet that sticks out of the left hand side of this stem; the second number specifies the default length of the beamlet to the right. The actual length of a beamlet is determined by taking either the default length or the length specified by `beamlet-max-length-proportion`, whichever is smaller.

`beamlet-max-length-proportion` (pair)
: The maximum length of a beamlet, as a proportion of the distance between two adjacent stems.

`default-direction` (direction)
: Direction determined by note head positions.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`double-stem-separation` (number)
: The distance between the two stems of a half note in tablature when using `\tabFullNotation`, not counting the width of the stems themselves, expressed as a multiple of the default height of a staff-space in the traditional five-line staff.

`duration-log` (integer)
: The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`french-beaming` (boolean)
: Use French beaming style for this stem. The stem stops at the innermost beams.

`length` (dimension, in staff space)
: User override for the stem length of unbeamed stems (each unit represents half a `staff-space`).

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`max-beam-connect` (integer)
: Maximum number of beams to connect to beams from this stem. Further beams are typeset as beamlets.

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

`no-stem-extend` (boolean)
: If set, notes with ledger lines do not get stems extending to the middle staff line.

`note-collision-threshold` (dimension, in staff space)
: Simultaneous notes that are this close or closer in units of `staff-space` will be identified as vertically colliding. Used by `Stem` grobs for notes in the same voice, and `NoteCollision` grobs for notes in different voices. Default value 1.

`stem-begin-position` (number)
: User override for the begin position of a stem.

`stemlet-length` (number)
: How long should be a stem over a rest?

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`beam` (graphical (layout) object)
: A pointer to the beam, if applicable.

`flag` (graphical (layout) object)
: A pointer to a `Flag` object.

`french-beaming-stem-adjustment` (dimension, in staff space)
: Stem will be shortened by this amount of space in case of French beaming style.

`melody-spanner` (graphical (layout) object)
: The `MelodyItem` object for a stem.

`note-heads` (array of grobs)
: An array of note head grobs.

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

`rests` (array of grobs)
: An array of rest objects.

`stem-info` (pair)
: A cache of stem parameters.

`tremolo-flag` (graphical (layout) object)
: The tremolo object on a stem.

`tuplet-start` (boolean)
: Is stem at the start of a tuplet?

This grob interface is used in the following graphical object(s): `Stem` (page 481).

3.2.132 stem-tremolo-interface

A beam slashing a stem to indicate a tremolo. The property `shape` can be `beam-like` or `rectangle`.

User settable properties:

`beam-thickness` (dimension, in staff space)
: Beam thickness, measured in `staff-space` units.

`beam-width` (dimension, in staff space)
: Width of the tremolo sign.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`flag-count` (number)
: The number of tremolo beams.

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`shape` (symbol)
: This setting determines what shape a grob has. Valid choices depend on the `stencil` callback reading this property.

`slope` (number)
: The slope of this object.

Internal properties:

`stem` (graphical (layout) object)
: A pointer to a `Stem` object.

This grob interface is used in the following graphical object(s): `StemTremolo` (page 484).

3.2.133 sticky-grob-interface

A grob that is attached to another grob. Grobs type having this interface can be either items or spanners, depending on the class of their host. Sticky spanners implicitly take their bounds from the host.

Internal properties:

`sticky-host` (graphical (layout) object)
: The grob that a sticky grob attaches to.

This grob interface is used in the following graphical object(s): `BalloonText` (page 357), `ControlPoint` (page 385), `ControlPolygon` (page 386), `Footnote` (page 411), and `Parentheses` (page 459).

3.2.134 string-number-interface

A string number instruction.

This grob interface is used in the following graphical object(s): `StringNumber` (page 485).

3.2.135 stroke-finger-interface

A right hand finger instruction.

User settable properties:

`digit-names` (vector)
: Names for string finger digits.

This grob interface is used in the following graphical object(s): `StrokeFinger` (page 487).

3.2.136 `system-interface`

This is the top-level object: Each object in a score ultimately has a `System` object as its X and Y parent.

The `system-interface` implies the `spanner-interface` (page 588).

User settable properties:

`labels` (list)
: List of labels (symbols) placed on a column.

`page-number` (number)
: Page number on which this system ends up.

`rank-on-page` (number)
: 0-based index of the system on a page.

Internal properties:

`all-elements` (array of grobs)
: An array of all grobs in this line. Its function is to protect objects from being garbage collected.

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

`footnote-stencil` (stencil)
: The stencil of a system's footnotes.

`footnotes-after-line-breaking` (array of grobs)
: Footnote grobs of a broken system.

`footnotes-before-line-breaking` (array of grobs)
: Footnote grobs of a whole system.

`in-note-direction` (direction)
: Direction to place in-notes above a system.

`in-note-padding` (number)
: Padding between in-notes.

`in-note-stencil` (stencil)
: The stencil of a system's in-notes.

`pure-Y-extent` (pair of numbers)
: The estimated height of a system.

`vertical-alignment` (graphical (layout) object)
: The VerticalAlignment in a System.

This grob interface is added dynamically to grobs of class `System`.

3.2.137 system-start-delimiter-interface

The brace, bracket or bar in front of the system. The following values for `style` are recognized:

`bracket` A thick bracket, normally used to group similar instruments in a score. Default for `StaffGroup`. `SystemStartBracket` uses this style.

`brace` A 'piano style' brace normally used for an instrument that uses two staves. The default style for `GrandStaff`. `SystemStartBrace` uses this style.

`bar-line` A simple line between the staves in a score. Default for staves enclosed in `<<` and `>>`. `SystemStartBar` uses this style.

`line-bracket`
A simple square, normally used for subgrouping instruments in a score. `SystemStartSquare` uses this style.

See also `input/regression/system-start-nesting.ly`.

User settable properties:

`collapse-height` (dimension, in staff space)
Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

`style` (symbol)
This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`thickness` (number)
For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

This grob interface is used in the following graphical object(s): `SystemStartBar` (page 491), `SystemStartBrace` (page 492), `SystemStartBracket` (page 493), and `SystemStartSquare` (page 494).

3.2.138 system-start-text-interface

Text in front of the system.

User settable properties:

`long-text` (markup)
Text markup. See Section "Formatting text" in *Notation Reference*.

`self-alignment-X` (number)
Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number)
Like `self-alignment-X` but for the Y axis.

`text` (markup)
Text markup. See Section "Formatting text" in *Notation Reference*.

This grob interface is used in the following graphical object(s): `InstrumentName` (page 420).

3.2.139 tab-note-head-interface

A note head in tablature.

User settable properties:

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

Internal properties:

`display-cautionary` (boolean)
: Should the grob be displayed as a cautionary grob?

`span-start` (boolean)
: Is the note head at the start of a spanner?

This grob interface is used in the following graphical object(s): `TabNoteHead` (page 494).

3.2.140 text-interface

A Scheme markup text, see Section "Formatting text" in *Notation Reference* and Section "New markup command definition" in *Extending*.

There are two important commands: `ly:text-interface::print`, which is a grob callback, and `ly:text-interface::interpret-markup`.

User settable properties:

`baseline-skip` (dimension, in staff space)
: Distance between base lines of multiple lines of text.

`flag-style` (symbol)
: The style of the flag to be used with `MetronomeMark`. Available are `'modern-straight-flag`, `'old-straight-flag`, `flat-flag`, `mensural` and `'default`

`replacement-alist` (list)
: Alist of strings. The key is a string of the pattern to be replaced. The value is a string of what should be displayed. Useful for ligatures.

`text` (markup)
: Text markup. See Section "Formatting text" in *Notation Reference*.

`text-direction` (direction)
: This controls the ordering of the words. The default `RIGHT` is for roman text. Arabic or Hebrew should use `LEFT`.

`word-space` (dimension, in staff space)
: Space to insert between words in texts.

This grob interface is used in the following graphical object(s): `BalloonText` (page 357), `BarNumber` (page 361), `BassFigure` (page 363), `BendSpanner` (page 369), `BreathingSign` (page 373), `CenteredBarNumber` (page 375), `ChordName` (page 377), `ClefModifier` (page 381), `CombineTextScript` (page 383), `ControlPoint` (page 385), `ControlPolygon` (page 386), `DoublePercentRepeatCounter` (page 396), `DynamicText` (page 402), `DynamicTextSpanner` (page 403), `Fingering` (page 408), `Footnote` (page 411), `HorizontalBracketText` (page 419), `InstrumentName` (page 420), `InstrumentSwitch` (page 421), `JumpScript` (page 422), `LyricText` (page 438), `MeasureCounter` (page 439), `MeasureSpanner` (page 442),

MetronomeMark (page 444), MultiMeasureRestNumber (page 447), MultiMeasureRestText (page 450), NoteName (page 456), OttavaBracket (page 457), PercentRepeatCounter (page 461), RehearsalMark (page 465), SostenutoPedal (page 474), StanzaNumber (page 481), StringNumber (page 485), StrokeFinger (page 487), SustainPedal (page 488), TabNoteHead (page 494), TextScript (page 496), TupletNumber (page 510), UnaCordaPedal (page 511), and VoltaBracket (page 517).

3.2.141 text-script-interface

An object that is put above or below a note.

User settable properties:

avoid-slur (symbol)
: Method of handling slur collisions. Choices are inside, outside, around, and ignore. inside adjusts the slur if needed to keep the grob inside the slur. outside moves the grob vertically to the outside of the slur. around moves the grob vertically to the outside of the slur only if there is a collision. ignore does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), outside and around behave like ignore.

script-priority (number)
: A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

Internal properties:

slur (graphical (layout) object)
: A pointer to a Slur object.

This grob interface is used in the following graphical object(s): BendSpanner (page 369), CombineTextScript (page 383), Fingering (page 408), StringNumber (page 485), StrokeFinger (page 487), and TextScript (page 496).

3.2.142 tie-column-interface

Object that sets directions of multiple ties in a tied chord.

User settable properties:

tie-configuration (list)
: List of (*position . dir*) pairs, indicating the desired tie configuration, where *position* is the offset from the center of the staff in staff space and *dir* indicates the direction of the tie (1=>up, -1=>down, 0=>center). A non-pair entry in the list causes the corresponding tie to be formatted automatically.

Internal properties:

positioning-done (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

ties (array of grobs)
: A grob array of Tie objects.

This grob interface is used in the following graphical object(s): TieColumn (page 501).

3.2.143 tie-interface

A tie - a horizontal curve connecting two noteheads.

The following properties may be set in the `details` list.

`height-limit`
: The maximum height allowed for this tie.

`ratio`
: Parameter for tie shape. The higher this number, the quicker the slur attains its height-limit.

`between-length-limit`
: This detail is currently unused.

`wrong-direction-offset-penalty`
: Demerit for ties that are offset in the wrong direction.

`min-length`
: If the tie is shorter than this amount (in staff-spaces) an increasingly large length penalty is incurred.

`min-length-penalty-factor`
: Demerit factor for tie lengths shorter than `min-length`.

`center-staff-line-clearance`
: If the center of the tie is closer to a staff line than this amount, an increasingly large staff line collision penalty is incurred.

`tip-staff-line-clearance`
: If the tips of the tie are closer to a staff line than this amount, an increasingly large staff line collision penalty is incurred.

`staff-line-collision-penalty`
: Demerit factor for ties whose tips or center come close to staff lines.

`dot-collision-clearance`
: If the tie comes closer to a dot than this amount, an increasingly large dot collision penalty is incurred.

`dot-collision-penalty`
: Demerit factor for ties which come close to dots.

`note-head-gap`
: The distance (in staff-spaces) by which the ends of the tie are offset horizontally from the center line through the note head.

`stem-gap`
: The distance (in staff-spaces) by which the ends of the tie are offset horizontally from a stem which is on the same side of the note head as the tie.

`tie-column-monotonicity-penalty`
: Demerit if the y-position of this tie in the set of ties being considered is less than the y-position of the previous tie.

`tie-tie-collision-distance`
: If this tie is closer than this amount to the previous tie in the set being considered, an increasingly large tie-tie collision penalty is incurred.

`tie-tie-collision-penalty`
: Demerit factor for a tie in the set being considered which is close to the previous one.

`horizontal-distance-penalty-factor`
: Demerit factor for ties in the set being considered which are horizontally distant from the note heads.

`vertical-distance-penalty-factor`
: Demerit factor for ties in the set being considered which are vertically distant from the note heads.

`same-dir-as-stem-penalty`
: Demerit if tie is on the same side as a stem or on the opposite side to the one specified.

`intra-space-threshold`
: If the tie's height (in half staff-spaces) is less than this it is positioned between two adjacent staff lines; otherwise it is positioned to straddle a staff line further from the note heads.

`outer-tie-length-symmetry-penalty-factor`
: Demerit factor for ties horizontally positioned unsymmetrically with respect to the two note heads.

`outer-tie-vertical-distance-symmetry-penalty-factor`
: Demerit factor for ties vertically positioned unsymmetrically with respect to the two note heads.

`outer-tie-vertical-gap`
: Amount (in half staff-spaces) by which a tie is moved away from the note heads if it is closer to either of them than 0.25 half staff-spaces.

`skyline-padding`
: Padding of the skylines around note heads in chords.

`single-tie-region-size`
: The number of candidate ties to generate when only a single tie is required. Successive candidates differ in their initial vertical position by half a staff-space.

`multi-tie-region-size`
: The number of variations that are tried for the extremal ties in a chord. Variations differ in their initial vertical position by half a staff-space.

User settable properties:

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`control-points` (list of number pairs)
: List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`dash-definition` (pair)
: List of `dash-elements` defining the dash structure. Each `dash-element` has a starting t value, an ending t-value, a `dash-fraction`, and a `dash-period`.

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`head-direction` (direction)
: Are the note heads left or right in a semitie?

`line-thickness` (number)
: For slurs and ties, this is the diameter of the virtual "pen" that draws the two arcs of the curve's outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

`staff-position` (number)
: Vertical position, measured in half staff spaces, counted from the middle line.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`annotation` (string)
: Annotate a grob for debug purposes.

This grob interface is used in the following graphical object(s): `LaissezVibrerTie` (page 430), `RepeatTie` (page 467), and `Tie` (page 500).

3.2.144 `time-signature-interface`

A time signature, in different styles. The following values for `style` are are recognized:

`C` 4/4 and 2/2 are typeset as C and struck C, respectively. All other time signatures are written with two digits. The value `default` is equivalent to `C`.

`neomensural`
: 2/2, 3/2, 2/4, 3/4, 4/4, 6/4, 9/4, 4/8, 6/8, and 9/8 are typeset with neo-mensural style mensuration marks. All other time signatures are written with two digits.

`mensural` 2/2, 3/2, 2/4, 3/4, 4/4, 6/4, 9/4, 4/8, 6/8, and 9/8 are typeset with mensural style mensuration marks. All other time signatures are written with two digits.

`single-digit`
: All time signatures are typeset with a single digit, e.g., 3/2 is written as 3.

`numbered` All time signatures are typeset with two digits.

User settable properties:

`fraction` (fraction, as pair)
: Numerator and denominator of a time signature object.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

This grob interface is used in the following graphical object(s): `TimeSignature` (page 502).

3.2.145 `trill-pitch-accidental-interface`

An accidental for trill pitch.

This grob interface is used in the following graphical object(s): `TrillPitchAccidental` (page 504).

3.2.146 `trill-spanner-interface`

A trill spanner.

This grob interface is used in the following graphical object(s): `TrillSpanner` (page 507).

3.2.147 `tuplet-bracket-interface`

A bracket with a number in the middle, used for tuplets. When the bracket spans a line break, the value of `break-overshoot` determines how far it extends beyond the staff. At a line break, the markups in the `edge-text` are printed at the edges.

User settable properties:

`avoid-scripts` (boolean)
: If set, a tuplet bracket avoids the scripts associated with the note heads it encompasses.

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`bracket-visibility` (boolean or symbol)
: This controls the visibility of the tuplet bracket. Setting it to false prevents printing of the bracket. Setting the property to `if-no-beam` makes it print only if there is no beam associated with this tuplet bracket.

`break-overshoot` (pair of numbers)
: How much does a broken spanner stick out of its bounds?

`connect-to-neighbor` (pair)
: Pair of booleans, indicating whether this grob looks as a continued break.

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`edge-text` (pair)
: A pair specifying the texts to be set at the edges: `(left-text . right-text)`.

`full-length-padding` (number)
: How much padding to use at the right side of a full-length tuplet bracket.

`full-length-to-extent` (boolean)
: Run to the extent of the column for a full-length tuplet bracket.

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`positions` (pair of numbers)
: Pair of staff coordinates `(start . end)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`staff-padding` (dimension, in staff space)
: Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to `Staff.StaffSymbol.thickness`).

`tuplet-slur` (boolean)
: Draw a slur instead of a bracket for tuplets.

`X-positions` (pair of numbers)
: Pair of X staff coordinates of a spanner in the form `(left . right)`, where both *left* and *right* are in `staff-space` units of the current staff.

Internal properties:

`note-columns` (array of grobs)
: An array of `NoteColumn` grobs.

`scripts` (array of grobs)
: An array of `Script` objects.

`tuplet-number` (graphical (layout) object)
: The number for a bracket.

`tuplets` (array of grobs)
: An array of smaller tuplet brackets.

This grob interface is used in the following graphical object(s): `LigatureBracket` (page 434), and `TupletBracket` (page 509).

3.2.148 tuplet-number-interface

The number for a bracket.

User settable properties:

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside` moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP=1`, `DOWN=-1`, `LEFT=-1`, `RIGHT=1`, `CENTER=0`.

`knee-to-beam` (boolean)
: Determines whether a tuplet number will be positioned next to a kneed beam.

Internal properties:

`bracket` (graphical (layout) object)
: The bracket for a number.

This grob interface is used in the following graphical object(s): `TupletNumber` (page 510).

3.2.149 unbreakable-spanner-interface

A spanner that should not be broken across line breaks. Override with `breakable=##t`.

User settable properties:

`breakable` (boolean)
: Allow breaks here.

This grob interface is used in the following graphical object(s): `Beam` (page 366), `DurationLine` (page 399), and `Glissando` (page 414).

3.2.150 vaticana-ligature-interface

A vaticana style Gregorian ligature.

User settable properties:

`glyph-name` (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`add-cauda` (boolean)
: Does this flexa require an additional cauda on the left side?

`add-join` (boolean)
: Is this ligature head-joined with the next one by a vertical line?

`add-stem` (boolean)
: Is this ligature head a virga and therefore needs an additional stem on the right side?

`delta-position` (number)
: The vertical position difference.

`flexa-height` (dimension, in staff space)
: The height of a flexa shape in a ligature grob (in `staff-space` units).

`flexa-width` (dimension, in staff space)
: The width of a flexa shape in a ligature grob (in `staff-space` units).

`x-offset` (dimension, in staff space)
: Extra horizontal offset for ligature heads.

This grob interface is used in the following graphical object(s): `NoteHead` (page 455), and `VaticanaLigature` (page 513).

3.2.151 volta-bracket-interface

Volta bracket with number.

User settable properties:

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

Internal properties:

`bars` (array of grobs)
: An array of bar line pointers.

This grob interface is used in the following graphical object(s): `VoltaBracket` (page 517).

3.2.152 volta-interface

A volta repeat.

This grob interface is used in the following graphical object(s): `VoltaBracket` (page 517), and `VoltaBracketSpanner` (page 518).

3.3 User backend properties

`add-stem-support` (boolean)
: If set, the `Stem` object is included in this script's support.

`after-line-breaking` (boolean)
: Dummy property, used to trigger callback for `after-line-breaking`.

`align-dir` (direction)
: Which side to align? `-1`: left side, `0`: around center of width, `1`: right side.

`allow-loose-spacing` (boolean)
: If set, column can be detached from main spacing.

`allow-span-bar` (boolean)
: If false, no inter-staff bar line will be created below this bar line.

`alteration` (number)
: Alteration numbers for accidental.

`alteration-alist` (list)
: List of `(`*`pitch . accidental`*`)` pairs for key signature.

`alteration-glyph-name-alist` (list)
: An alist of key-string pairs.

`annotation-balloon` (boolean)
: Print the balloon around an annotation.

`annotation-line` (boolean)
: Print the line from an annotation to the grob that it annotates.

`arpeggio-direction` (direction)
: If set, put an arrow on the arpeggio squiggly line.

`arrow-length` (number)
: Arrow length.

`arrow-width` (number)
: Arrow width.

`auto-knee-gap` (dimension, in staff space)
: If a gap is found between note heads where a horizontal beam fits and it is larger than this number, make a kneed beam.

`automatically-numbered` (boolean)
: If set, footnotes are automatically numbered.

`average-spacing-wishes` (boolean)
: If set, the spacing wishes are averaged over staves.

`avoid-note-head` (boolean)
: If set, the stem of a chord does not pass through all note heads, but starts at the last note head.

`avoid-scripts` (boolean)
: If set, a tuplet bracket avoids the scripts associated with the note heads it encompasses.

`avoid-slur` (symbol)
: Method of handling slur collisions. Choices are `inside`, `outside`, `around`, and `ignore`. `inside` adjusts the slur if needed to keep the grob inside the slur. `outside`

moves the grob vertically to the outside of the slur. `around` moves the grob vertically to the outside of the slur only if there is a collision. `ignore` does not move either. In grobs whose notational significance depends on vertical position (such as accidentals, clefs, etc.), `outside` and `around` behave like `ignore`.

`axes` (list) List of axis numbers. In the case of alignment grobs, this should contain only one number.

`bar-extent` (pair of numbers)
The Y-extent of the actual bar line. This may differ from `Y-extent` because it does not include the dots in a repeat bar line.

`base-shortest-duration` (moment)
Spacing is based on the shortest notes in a piece. Normally, pieces are spaced as if notes at least as short as this are present.

`baseline-skip` (dimension, in staff space)
Distance between base lines of multiple lines of text.

`beam-thickness` (dimension, in staff space)
Beam thickness, measured in `staff-space` units.

`beam-width` (dimension, in staff space)
Width of the tremolo sign.

`beamed-stem-shorten` (list)
How much to shorten beamed stems, when their direction is forced. It is a list, since the value is different depending on the number of flags and beams.

`beaming` (pair)
Pair of number lists. Each number list specifies which beams to make. `0` is the central beam, `1` is the next beam toward the note, etc. This information is used to determine how to connect the beaming patterns from stem to stem inside a beam.

`beamlet-default-length` (pair)
A pair of numbers. The first number specifies the default length of a beamlet that sticks out of the left hand side of this stem; the second number specifies the default length of the beamlet to the right. The actual length of a beamlet is determined by taking either the default length or the length specified by `beamlet-max-length-proportion`, whichever is smaller.

`beamlet-max-length-proportion` (pair)
The maximum length of a beamlet, as a proportion of the distance between two adjacent stems.

`before-line-breaking` (boolean)
Dummy property, used to trigger a callback function.

`bend-me` (boolean)
Decide whether this grob is bent.

`between-cols` (pair)
Where to attach a loose column to.

`bound-details` (list)
An alist of properties for determining attachments of spanners to edges.

`bound-padding` (number)
The amount of padding to insert around spanner bounds.

`bracket-flare` (pair of numbers)
: A pair of numbers specifying how much edges of brackets should slant outward. Value `0.0` means straight edges.

`bracket-visibility` (boolean or symbol)
: This controls the visibility of the tuplet bracket. Setting it to false prevents printing of the bracket. Setting the property to `if-no-beam` makes it print only if there is no beam associated with this tuplet bracket.

`break-align-anchor` (number)
: Grobs aligned to this breakable item will have their X-offsets shifted by this number. In bar lines, for example, this is used to position grobs relative to the (visual) center of the bar line.

`break-align-anchor-alignment` (number)
: Read by `ly:break-aligned-interface::calc-extent-aligned-anchor` for aligning an anchor to a grob's extent.

`break-align-orders` (vector)
: This is a vector of 3 lists: `#(`*`end-of-line unbroken start-of-line`*`)`. Each list contains *break-align symbols* that specify an order of breakable items (see Section "break-alignment-interface" in *Internals Reference*).

  For example, this places time signatures before clefs:

```
\override Score.BreakAlignment.break-align-orders =
  #(make-vector 3 '(left-edge
                    cue-end-clef
                    ambitus
                    breathing-sign
                    time-signature
                    clef
                    cue-clef
                    staff-bar
                    key-cancellation
                    key-signature
                    custos))
```

`break-align-symbol` (symbol)
: This key is used for aligning, ordering, and spacing breakable items. See Section "break-alignment-interface" in *Internals Reference*.

`break-align-symbols` (list)
: A list of *break-align symbols* that determines which breakable items to align this to. If the grob selected by the first symbol in the list is invisible due to `break-visibility`, we will align to the next grob (and so on). Choices are listed in Section "break-alignment-interface" in *Internals Reference*.

`break-overshoot` (pair of numbers)
: How much does a broken spanner stick out of its bounds?

`break-visibility` (vector)
: A vector of 3 booleans, `#(`*`end-of-line unbroken begin-of-line`*`)`. `#t` means visible, `#f` means killed.

`breakable` (boolean)
: Allow breaks here.

`broken-bound-padding` (number)
: The amount of padding to insert when a spanner is broken at a line break.

`chord-dots-limit` (integer)
: Limits the column of dots on each chord to the height of the chord plus `chord-dots-limit` staff-positions.

`circled-tip` (boolean)
: Put a circle at start/end of hairpins (al/del niente).

`clef-alignments` (list)
: An alist of parent-alignments that should be used for clef modifiers with various clefs

`clip-edges` (boolean)
: Allow outward pointing beamlets at the edges of beams?

`collapse-height` (dimension, in staff space)
: Minimum height of system start delimiter. If equal or smaller, the bracket/brace/line is removed.

`collision-interfaces` (list)
: A list of interfaces for which automatic beam-collision resolution is run.

`collision-voice-only` (boolean)
: Does automatic beam collsion apply only to the voice in which the beam was created?

`color` (color)
: The color of this grob.

`common-shortest-duration` (moment)
: The most common shortest note length. This is used in spacing. Enlarging this sets the score tighter.

`concaveness` (number)
: A beam is concave if its inner stems are closer to the beam than the two outside stems. This number is a measure of the closeness of the inner stems. It is used for damping the slope of the beam.

`connect-to-neighbor` (pair)
: Pair of booleans, indicating whether this grob looks as a continued break.

`control-points` (list of number pairs)
: List of offsets (number pairs) that form control points for the tie, slur, or bracket shape. For Béziers, this should list the control points of a third-order Bézier curve.

`count-from` (integer)
: The first measure in a measure count receives this number. The following measures are numbered in increments from this initial value.

`damping` (number)
: Amount of beam slope damping.

`dash-definition` (pair)
: List of `dash-elements` defining the dash structure. Each `dash-element` has a starting t value, an ending t-value, a `dash-fraction`, and a `dash-period`.

`dash-fraction` (number)
: Size of the dashes, relative to `dash-period`. Should be between `0.1` and `1.0` (continuous line). If set to `0.0`, a dotted line is produced

`dash-period` (number)
: The length of one dash together with whitespace. If negative, no line is drawn at all.

`dashed-edge` (boolean)
: If set, the bracket edges are dashed like the rest of the bracket.

`default-direction` (direction)
: Direction determined by note head positions.

`default-staff-staff-spacing` (list)
: The settings to use for `staff-staff-spacing` when it is unset, for ungrouped staves and for grouped staves that do not have the relevant `StaffGrouper` property set (`staff-staff-spacing` or `staffgroup-staff-spacing`).

`details` (list)
: Alist of parameters for detailed grob behavior. More information on the allowed parameters for a grob can be found by looking at the top of the Internals Reference page for each interface having a `details` property.

`digit-names` (vector)
: Names for string finger digits.

`direction` (direction)
: If `side-axis` is `0` (or `X`), then this property determines whether the object is placed `LEFT`, `CENTER` or `RIGHT` with respect to the other object. Otherwise, it determines whether the object is placed `UP`, `CENTER` or `DOWN`. Numerical values may also be used: `UP`=`1`, `DOWN`=`-1`, `LEFT`=`-1`, `RIGHT`=`1`, `CENTER`=`0`.

`dot-count` (integer)
: The number of dots.

`dot-negative-kern` (number)
: The space to remove between a dot and a slash in percent repeat glyphs. Larger values bring the two elements closer together.

`dot-placement-list` (list)
: List consisting of `(`*`description string-number fret-number finger-number`*`)` entries used to define fret diagrams.

`double-stem-separation` (number)
: The distance between the two stems of a half note in tablature when using `\tabFullNotation`, not counting the width of the stems themselves, expressed as a multiple of the default height of a staff-space in the traditional five-line staff.

`duration-log` (integer)
: The 2-log of the note head duration, i.e., `0` = whole note, `1` = half note, etc.

`eccentricity` (number)
: How asymmetrical to make a slur. Positive means move the center to the right.

`edge-height` (pair)
: A pair of numbers specifying the heights of the vertical edges: `(`*`left-height . right-height`*`)`.

`edge-text` (pair)
: A pair specifying the texts to be set at the edges: `(`*`left-text . right-text`*`)`.

`endpoint-alignments` (pair of numbers)
: A pair of numbers representing the alignments of an object's endpoints. E.g., the ends of a hairpin relative to `NoteColumn` grobs.

`expand-limit` (integer)
: Maximum number of measures expanded in church rests.

`extra-dy` (number)
: Slope glissandi this much extra.

`extra-offset` (pair of numbers)
: A pair representing an offset. This offset is added just before outputting the symbol, so the typesetting engine is completely oblivious to it. The values are measured in `staff-space` units of the staff's `StaffSymbol`.

`extra-spacing-height` (pair of numbers)
: In the horizontal spacing problem, we increase the height of each item by this amount (by adding the 'car' to the bottom of the item and adding the 'cdr' to the top of the item). In order to make a grob infinitely high (to prevent the horizontal spacing problem from placing any other grobs above or below this grob), set this to `(-inf.0 . +inf.0)`.

`extra-spacing-width` (pair of numbers)
: In the horizontal spacing problem, we pad each item by this amount (by adding the 'car' on the left side of the item and adding the 'cdr' on the right side of the item). In order to make a grob take up no horizontal space at all, set this to `(+inf.0 . -inf.0)`.

`extroversion` (number)
: For polygons, how the thickness of the line is spread on each side of the exact polygon whith ideal zero thickness. If this is 0, the middle of line is on the polygon. If 1, the line sticks out of the polygon. If -1, the outer side of the line is exactly on the polygon. Other numeric values are interpolated.

`filled` (boolean)
: Whether an object is filled with ink.

`flag-count` (number)
: The number of tremolo beams.

`flag-style` (symbol)
: The style of the flag to be used with `MetronomeMark`. Available are `'modern-straight-flag`, `'old-straight-flag`, `flat-flag`, `mensural` and `'default`

`flat-positions` (list)
: Flats in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`font-encoding` (symbol)
: The font encoding is the broadest category for selecting a font. Currently, only lilypond's system fonts (Emmentaler) are using this property. Available values are `fetaMusic` (Emmentaler), `fetaBraces`, `fetaText` (Emmentaler).

`font-family` (symbol)
: The font family is the broadest category for selecting text fonts. Options include: `sans`, `roman`.

`font-features` (list)
: Opentype features.

`font-name` (string)
: Specifies a file name (without extension) of the font to load. This setting overrides selection using `font-family`, `font-series` and `font-shape`.

`font-series` (symbol)
: Select the series of a font. Choices include `medium`, `bold`, `bold-narrow`, etc.

`font-shape` (symbol)
: Select the shape of a font. Choices include `upright`, `italic`, `caps`.

`font-size` (number)
: The font size, compared to the 'normal' size. `0` is style-sheet's normal size, `-1` is smaller, `+1` is bigger. Each step of 1 is approximately 12% larger; 6 steps are exactly a factor 2 larger. If the context property `fontSize` is set, its value is added to this before the glyph is printed. Fractional values are allowed.

`footnote` (boolean)
: Should this be a footnote or in-note?

`footnote-music` (music)
: Music creating a footnote.

`footnote-text` (markup)
: A footnote for the grob.

`force-hshift` (number)
: This specifies a manual shift for notes in collisions. The unit is the note head width of the first voice note. This is used by Section "note-collision-interface" in *Internals Reference*.

`forced-spacing` (number)
: Spacing forced between grobs, used in various ligature engravers.

`fraction` (fraction, as pair)
: Numerator and denominator of a time signature object.

`french-beaming` (boolean)
: Use French beaming style for this stem. The stem stops at the innermost beams.

`fret-diagram-details` (list)
: An alist of detailed grob properties for fret diagrams. Each alist entry consists of a (*property* . *value*) pair. The properties which can be included in `fret-diagram-details` include the following:

  - `barre-type` – Type of barre indication used. Choices include `curved`, `straight`, and `none`. Default `curved`.
  - `capo-thickness` – Thickness of capo indicator, in multiples of fret-space. Default value 0.5.
  - `dot-color` – Color of dots. Options include `black` and `white`. Default `black`.
  - `dot-label-font-mag` – Magnification for font used to label fret dots. Default value 1.
  - `dot-position` – Location of dot in fret space. Default 0.6 for dots without labels, 0.95-`dot-radius` for dots with labels.
  - `dot-radius` – Radius of dots, in terms of fret spaces. Default value 0.425 for labeled dots, 0.25 for unlabeled dots.
  - `finger-code` – Code for the type of fingering indication used. Options include `none`, `in-dot`, and `below-string`. Default `none` for markup fret diagrams, `below-string` for `FretBoards` fret diagrams.

- `fret-count` – The number of frets. Default 4.
- `fret-distance` – Multiplier to adjust the distance between frets. Default 1.0.
- `fret-label-custom-format` – The format string to be used label the lowest fret number, when `number-type` equals to `custom`. Default `"~a"`.
- `fret-label-font-mag` – The magnification of the font used to label the lowest fret number. Default 0.5.
- `fret-label-vertical-offset` – The offset of the fret label from the center of the fret in direction parallel to strings. Default 0.
- `fret-label-horizontal-offset` – The offset of the fret label from the center of the fret in direction orthogonal to strings. Default 0.
- `handedness` – Print the fret-diagram left- or right-handed. `-1`, `LEFT` for left ; `1`, `RIGHT` for right. Default `RIGHT`.
- `paren-padding` – The padding for the parenthesis. Default 0.05.
- `label-dir` – Side to which the fret label is attached. `-1`, `LEFT`, or `DOWN` for left or down; `1`, `RIGHT`, or `UP` for right or up. Default `RIGHT`.
- `mute-string` – Character string to be used to indicate muted string. Default `"x"`.
- `number-type` – Type of numbers to use in fret label. Choices include `roman-lower`, `roman-upper`, `arabic` and `custom`. In the later case, the format string is supplied by the `fret-label-custom-format` property. Default `roman-lower`.
- `open-string` – Character string to be used to indicate open string. Default `"o"`.
- `orientation` – Orientation of fret-diagram. Options include `normal`, `landscape`, and `opposing-landscape`. Default `normal`.
- `string-count` – The number of strings. Default 6.
- `string-distance` – Multiplier to adjust the distance between strings. Default 1.0.
- `string-label-font-mag` – The magnification of the font used to label fingerings at the string, rather than in the dot. Default value 0.6 for `normal` orientation, 0.5 for `landscape` and `opposing-landscape`.
- `string-thickness-factor` – Factor for changing thickness of each string in the fret diagram. Thickness of string k is given by `thickness` * (1+`string-thickness-factor`) `^` (k-1). Default 0.
- `top-fret-thickness` – The thickness of the top fret line, as a multiple of the standard thickness. Default value 3.
- `xo-font-magnification` – Magnification used for mute and open string indicators. Default value 0.5.
- `xo-padding` – Padding for open and mute indicators from top fret. Default value 0.25.

`full-length-padding` (number)
: How much padding to use at the right side of a full-length tuplet bracket.

`full-length-to-extent` (boolean)
: Run to the extent of the column for a full-length tuplet bracket.

`full-measure-extra-space` (number)
: Extra space that is allocated at the beginning of a measure with only one note. This property is read from the NonMusicalPaperColumn that begins the measure.

`full-size-change` (boolean)
: Don't make a change clef smaller.

`gap` (dimension, in staff space)
: Size of a gap in a variable symbol.

`gap-count` (integer)
: Number of gapped beams for tremolo.

`glissando-skip` (boolean)
: Should this `NoteHead` be skipped by glissandi?

`glyph` (string)
: A string determining what 'style' of glyph is typeset. Valid choices depend on the function that is reading this property.

  In combination with (span) bar lines, it is a string resembling the bar line appearance in ASCII form.

`glyph-name` (string)
: The glyph name within the font.

  In the context of (span) bar lines, *glyph-name* represents a processed form of `glyph`, where decisions about line breaking etc. are already taken.

`graphical` (boolean)
: Display in graphical (vs. text) form.

`grow-direction` (direction)
: Crescendo or decrescendo?

`hair-thickness` (number)
: Thickness of the thin line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to `Staff.StaffSymbol.thickness`).

`harp-pedal-details` (list)
: An alist of detailed grob properties for harp pedal diagrams. Each alist entry consists of a `(property . value)` pair. The properties which can be included in harp-pedal-details include the following:

  - `box-offset` – Vertical shift of the center of flat/sharp pedal boxes above/below the horizontal line. Default value 0.8.
  - `box-width` – Width of each pedal box. Default value 0.4.
  - `box-height` – Height of each pedal box. Default value 1.0.
  - `space-before-divider` – Space between boxes before the first divider (so that the diagram can be made symmetric). Default value 0.8.
  - `space-after-divider` – Space between boxes after the first divider. Default value 0.8.
  - `circle-thickness` – Thickness (in unit of the line-thickness) of the ellipse around circled pedals. Default value 0.5.
  - `circle-x-padding` – Padding in X direction of the ellipse around circled pedals. Default value 0.15.
  - `circle-y-padding` – Padding in Y direction of the ellipse around circled pedals. Default value 0.2.

`head-direction` (direction)
: Are the note heads left or right in a semitie?

`height` (dimension, in staff space)
: Height of an object in `staff-space` units.

`height-limit` (dimension, in staff space)
: Maximum slur height: The longer the slur, the closer it is to this height.

`hide-tied-accidental-after-break` (boolean)
: If set, an accidental that appears on a tied note after a line break will not be displayed.

`horizon-padding` (number)
: The amount to pad the axis along which a `Skyline` is built for the `side-position-interface`.

`horizontal-shift` (integer)
: An integer that identifies ranking of `NoteColumn`s for horizontal shifting. This is used by Section "note-collision-interface" in *Internals Reference*.

`horizontal-skylines` (pair of skylines)
: Two skylines, one to the left and one to the right of this grob.

`id` (string)
: An id string for the grob.

`ignore-ambitus` (boolean)
: If set, don't consider this notehead for ambitus calculation.

`ignore-collision` (boolean)
: If set, don't do note collision resolution on this `NoteColumn`.

`implicit` (boolean)
: Is this an implicit bass figure?

`inspect-quants` (pair of numbers)
: If debugging is set, set beam and slur position to a (quantized) position that is as close as possible to this value, and print the demerits for the inspected position in the output.

`keep-inside-line` (boolean)
: If set, this column cannot have objects sticking into the margin.

`kern` (dimension, in staff space)
: The space between individual elements in any compound bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`knee` (boolean)
: Is this beam kneed?

`knee-spacing-correction` (number)
: Factor for the optical correction amount for kneed beams. Set between `0` for no correction and `1` for full correction.

`knee-to-beam` (boolean)
: Determines whether a tuplet number will be positioned next to a kneed beam.

`labels` (list)
: List of labels (symbols) placed on a column.

`layer` (integer)
: An integer which determines the order of printing objects. 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.

`ledger-extra` (dimension, in staff space)
: Extra distance from staff line to draw ledger lines for.

`ledger-line-thickness` (pair of numbers)
: The thickness of ledger lines. It is the sum of 2 numbers: The first is the factor for line thickness, and the second for staff space. Both contributions are added.

`ledger-positions` (list)
: Vertical positions of ledger lines. When set on a `StaffSymbol` grob it defines a repeating pattern of ledger lines and any parenthesized groups will always be shown together.

`ledger-positions-function` (any type)
: A quoted Scheme procedure that takes a `StaffSymbol` grob and the vertical position of a note head as arguments and returns a list of ledger line positions.

`left-bound-info` (list)
: An alist of properties for determining attachments of spanners to edges.

`left-number-text` (markup)
: For a measure counter, this is the formatted measure count. When the measure counter extends over several measures (like with compressed multi-measure rests), it is the text on the left side of the dash.

`left-padding` (dimension, in staff space)
: The amount of space that is put left to an object (e.g., a lyric extender).

`length` (dimension, in staff space)
: User override for the stem length of unbeamed stems (each unit represents half a `staff-space`).

`length-fraction` (number)
: Multiplier for lengths. Used for determining ledger lines and stem lengths.

`line-break-penalty` (number)
: Penalty for a line break at this column. This affects the choices of the line breaker; it avoids a line break at a column with a positive penalty and prefers a line break at a column with a negative penalty.

`line-break-permission` (symbol)
: Instructs the line breaker on whether to put a line break at this column. Can be `force` or `allow`.

`line-break-system-details` (list)
: An alist of properties to use if this column is the start of a system.

`line-count` (integer)
: The number of staff lines.

`line-positions` (list)
: Vertical positions of staff lines.

`line-thickness` (number)
: For slurs and ties, this is the diameter of the virtual “pen” that draws the two arcs of the curve’s outline, which intersect at the endpoints. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`long-text` (markup)
: Text markup. See Section "Formatting text" in *Notation Reference*.

`main-extent` (pair of numbers)
: The horizontal extent of a `NoteColumn` grob without taking suspended `NoteHead` grobs into account (i.e., `NoteHead`s forced into the unnatural direction of the `Stem` because of a chromatic clash).

`max-beam-connect` (integer)
: Maximum number of beams to connect to beams from this stem. Further beams are typeset as beamlets.

`max-symbol-separation` (number)
: The maximum distance between symbols making up a church rest.

`maximum-gap` (number)
: Maximum value allowed for `gap` property.

`measure-count` (integer)
: The number of measures for a multi-measure rest.

`measure-length` (moment)
: Length of a measure. Used in some spacing situations.

`merge-differently-dotted` (boolean)
: Merge note heads in collisions, even if they have a different number of dots. This is normal notation for some types of polyphonic music.

  `merge-differently-dotted` only applies to opposing stem directions (i.e., voice 1 & 2).

`merge-differently-headed` (boolean)
: Merge note heads in collisions, even if they have different note heads. The smaller of the two heads is rendered invisible. This is used in polyphonic guitar notation. The value of this setting is used by Section "note-collision-interface" in *Internals Reference*.

  `merge-differently-headed` only applies to opposing stem directions (i.e., voice 1 & 2).

`minimum-distance` (dimension, in staff space)
: Minimum distance between rest and notes or beam.

`minimum-length` (dimension, in staff space)
: Try to make a spanner at least this long, normally in the horizontal direction. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance between noteheads.

`minimum-length-after-break` (dimension, in staff space)
: If set, try to make a broken spanner starting a line this long. This requires an appropriate callback for the `springs-and-rods` property. If added to a `Tie`, this sets the minimum distance to the notehead.

`minimum-length-fraction` (number)
: Minimum length of ledger line as fraction of note head size.

`minimum-space` (dimension, in staff space)
: Minimum distance that the victim should move (after padding).

`minimum-X-extent` (pair of numbers)
: Minimum size of an object in X dimension, measured in `staff-space` units.

`minimum-Y-extent` (pair of numbers)
: Minimum size of an object in Y dimension, measured in `staff-space` units.

`neutral-direction` (direction)
: Which direction to take in the center of the staff.

`neutral-position` (number)
: Position (in half staff spaces) where to flip the direction of custos stem.

`next` (graphical (layout) object)
: Object that is next relation (e.g., the lyric syllable following an extender).

`no-ledgers` (boolean)
: If set, don't draw ledger lines on this object.

`no-stem-extend` (boolean)
: If set, notes with ledger lines do not get stems extending to the middle staff line.

`non-break-align-symbols` (list)
: A list of symbols that determine which NON-break-aligned interfaces to align this to.

`non-default` (boolean)
: Set for manually specified clefs and keys.

`non-musical` (boolean)
: True if the grob belongs to a `NonMusicalPaperColumn`.

`nonstaff-nonstaff-spacing` (list)
: The spacing alist controlling 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`. See `staff-staff-spacing` for a description of the alist structure.

`nonstaff-relatedstaff-spacing` (list)
: The spacing alist controlling 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. See `staff-staff-spacing` for a description of the alist structure.

`nonstaff-unrelatedstaff-spacing` (list)
: The spacing alist controlling the distance between the current non-staff line and the nearest 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`. See `staff-staff-spacing` for a description of the alist structure.

`normalized-endpoints` (pair)
: Represents left and right placement over the total spanner, where the width of the spanner is normalized between 0 and 1.

`note-collision-threshold` (dimension, in staff space)
: Simultaneous notes that are this close or closer in units of `staff-space` will be identified as vertically colliding. Used by `Stem` grobs for notes in the same voice, and `NoteCollision` grobs for notes in different voices. Default value 1.

`note-names` (vector)
: Vector of strings containing names for easy-notation note heads.

`number-range-separator` (markup)
: For a measure counter extending over several measures (like with compressed multi-measure rests), this is the separator between the two printed numbers.

`number-type` (symbol)
: Numbering style. Choices include `roman-lower`, `roman-upper` and `arabic`.

`output-attributes` (list)
: An alist of attributes for the grob, to be included in output files. When the SVG typesetting backend is used, the attributes are assigned to a group (`<g>`) containing all of the stencils that comprise a given grob. For example,

```
'((id . 123) (class . foo) (data-whatever . "bar"))
```

produces

```
<g id="123" class="foo" data-whatever="bar"> ... </g>
```

In the Postscript backend, where there is no way to group items, the setting of the `output-attributes` property has no effect.

`outside-staff-horizontal-padding` (number)
: By default, an outside-staff-object can be placed so that is it very close to another grob horizontally. If this property is set, the outside-staff-object is raised so that it is not so close to its neighbor.

`outside-staff-padding` (number)
: The padding to place between grobs when spacing according to `outside-staff-priority`. Two grobs with different `outside-staff-padding` values have the larger value of padding between them.

`outside-staff-placement-directive` (symbol)
: One of four directives telling how outside staff objects should be placed.

- `left-to-right-greedy` – Place each successive grob from left to right.
- `left-to-right-polite` – Place a grob from left to right only if it does not potentially overlap with another grob that has been placed on a pass through a grob array. If there is overlap, do another pass to determine placement.
- `right-to-left-greedy` – Same as `left-to-right-greedy`, but from right to left.
- `right-to-left-polite` – Same as `left-to-right-polite`, but from right to left.

`outside-staff-priority` (number)
: If set, the grob is positioned outside the staff in such a way as to avoid all collisions. In case of a potential collision, the grob with the smaller `outside-staff-priority` is closer to the staff.

`packed-spacing` (boolean)
: If set, the notes are spaced as tightly as possible.

`padding` (dimension, in staff space)
: Add this much extra space between objects that are next to each other.

`padding-pairs` (list)
: An alist mapping `(`*`name . name`*`)` to distances.

`page-break-penalty` (number)
: Penalty for page break at this column. This affects the choices of the page breaker; it avoids a page break at a column with a positive penalty and prefers a page break at a column with a negative penalty.

`page-break-permission` (symbol)
: Instructs the page breaker on whether to put a page break at this column. Can be `force` or `allow`.

`page-number` (number)
: Page number on which this system ends up.

`page-turn-penalty` (number)
: Penalty for a page turn at this column. This affects the choices of the page breaker; it avoids a page turn at a column with a positive penalty and prefers a page turn at a column with a negative penalty.

`page-turn-permission` (symbol)
: Instructs the page breaker on whether to put a page turn at this column. Can be `force` or `allow`.

`parent-alignment-X` (number)
: Specify on which point of the parent the object is aligned. The value `-1` means aligned on parent's left edge, `0` on center, and `1` right edge, in X direction. Other numerical values may also be specified - the unit is half the parent's width. If unset, the value from `self-alignment-X` property will be used.

`parent-alignment-Y` (number)
: Like `parent-alignment-X` but for the Y axis.

`parenthesis-friends` (list)
: A list of Grob types, as symbols. When parentheses enclose a Grob that has 'parenthesis-friends, the parentheses widen to include any child Grobs with type among 'parenthesis-friends.

`parenthesis-id` (symbol)
: When parenthesized grobs created in the same time step have this property, there is one set of parentheses for each group of grobs having the same value.

`parenthesized` (boolean)
: Parenthesize this grob.

`positions` (pair of numbers)
: Pair of staff coordinates `(`*`start . end`*`)`, where *start* and *end* are vertical positions in `staff-space` units of the current staff. For slurs, this value selects which slur candidate to use; if extreme positions are requested, the closest one is taken.

`prefer-dotted-right` (boolean)
: For note collisions, prefer to shift dotted up-note to the right, rather than shifting just the dot.

`protrusion` (number)
: In an arpeggio bracket, the length of the horizontal edges.

`rank-on-page` (number)
: 0-based index of the system on a page.

`ratio` (number)
: Parameter for slur shape. The higher this number, the quicker the slur attains its `height-limit`.

`remove-empty` (boolean)
: If set, remove group if it contains no interesting items.

`remove-first` (boolean)
: Remove the first staff of an orchestral score?

`remove-layer` (index or symbol)
: When set as a positive integer, the `Keep_alive_together_engraver` removes all `VerticalAxisGroup` grobs with a `remove-layer` larger than the smallest retained `remove-layer`. Set to `#f` to make a layer independent of the `Keep_alive_together_engraver`. Set to `'()`, the layer does not participate in the layering decisions. The property can also be set as a symbol for common behaviors: `#'any` to keep the layer alive with any other layer in the group; `#'above` or `#'below` to keep the layer alive with the context immediately before or after it, respectively.

`replacement-alist` (list)
: Alist of strings. The key is a string of the pattern to be replaced. The value is a string of what should be displayed. Useful for ligatures.

`restore-first` (boolean)
: Print a natural before the accidental.

`rhythmic-location` (rhythmic location)
: Where (bar number, measure position) in the score.

`right-bound-info` (list)
: An alist of properties for determining attachments of spanners to edges.

`right-number-text` (markup)
: When the measure counter extends over several measures (like with compressed multi-measure rests), this is the text on the right side of the dash. Usually unset.

`right-padding` (dimension, in staff space)
: Space to insert on the right side of an object (e.g., between note and its accidentals).

`rotation` (list)
: Number of degrees to rotate this object, and what point to rotate around. For example, `'(45 0 0)` rotates by 45 degrees around the center of this object.

`round-up-exceptions` (list)
: A list of pairs where car is the numerator and cdr the denominator of a moment. Each pair in this list means that the multi-measure rests of the corresponding length will be rounded up to the longer rest. See *round-up-to-longer-rest*.

`round-up-to-longer-rest` (boolean)
: Displays the longer multi-measure rest when the length of a measure is between two values of `usable-duration-logs`. For example, displays a breve instead of a whole in a 3/2 measure.

`rounded` (boolean)
: Decide whether lines should be drawn rounded or not.

`same-direction-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. This amount is used for stems with the same direction to compensate for note head to stem distance.

`script-priority` (number)
: A key for determining the order of scripts in a stack, by being added to the position of the script in the user input, the sum being the overall priority. Smaller means closer to the head.

`segno-kern` (number)
: The space between the two thin lines of the segno bar line symbol, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to *`Staff`*`.StaffSymbol.thickness`).

`self-alignment-X` (number)
: Specify alignment of an object. The value `-1` means left aligned, `0` centered, and `1` right-aligned in X direction. Other numerical values may also be specified - the unit is half the object width.

`self-alignment-Y` (number)
: Like `self-alignment-X` but for the Y axis.

`shape` (symbol)
: This setting determines what shape a grob has. Valid choices depend on the `stencil` callback reading this property.

`sharp-positions` (list)
: Sharps in key signatures are placed within the specified ranges of staff-positions. The general form is a list of pairs, with one pair for each type of clef, in order of the staff-position at which each clef places C: `(alto treble tenor soprano baritone mezzosoprano bass)`. If the list contains a single element it applies for all clefs. A single number in place of a pair sets accidentals within the octave ending at that staff-position.

`shorten-pair` (pair of numbers)
: The lengths to shorten on both sides a hairpin or text-spanner such as a pedal bracket. Positive values shorten the hairpin or text-spanner, while negative values lengthen it.

`shortest-duration-space` (number)
: Start with this multiple of `spacing-increment` space for the shortest duration. See also Section “spacing-spanner-interface” in *Internals Reference*.

`shortest-playing-duration` (moment)
: The duration of the shortest note playing here.

`shortest-starter-duration` (moment)
: The duration of the shortest note that starts here.

`show-control-points` (boolean)
: For grobs printing Bézier curves, setting this property to true causes the control points and control polygon to be drawn on the page for ease of tweaking.

`side-axis` (number)
: If the value is `X` (or equivalently `0`), the object is placed horizontally next to the other object. If the value is `Y` or `1`, it is placed vertically.

`side-relative-direction` (direction)
: Multiply direction of `direction-source` with this to get the direction of this object.

`size` (number)
: The ratio of the size of the object to its default size.

`skip-quanting` (boolean)
: Should beam quanting be skipped?

`skyline-horizontal-padding` (number)
: For determining the vertical distance between two staves, it is possible to have a configuration which would result in a tight interleaving of grobs from the top staff and the bottom staff. The larger this parameter is, the farther apart the staves are placed in such a configuration.

`skyline-vertical-padding` (number)
: The amount by which the left and right skylines of a column are padded vertically, beyond the `Y-extent`s and `extra-spacing-height`s of the constituent grobs in the column. Increase this to prevent interleaving of grobs from adjacent columns.

`slash-negative-kern` (number)
: The space to remove between slashes in percent repeat glyphs. Larger values bring the two elements closer together.

`slope` (number)
: The slope of this object.

`slur-padding` (number)
: Extra distance between slur and script.

`snap-radius` (number)
: The maximum distance between two objects that will cause them to snap to alignment along an axis.

`space-alist` (list)
: An alist that specifies distances from this grob to other breakable items, using the format:

```
'((break-align-symbol . (spacing-style . space))
  (break-align-symbol . (spacing-style . space))
  ...)
```

Standard choices for *`break-align-symbol`* are listed in Section "break-alignment-interface" in *Internals Reference*. Additionally, three special break-align symbols available to `space-alist` are:

`first-note`
: used when the grob is just left of the first note on a line

`next-note`
: used when the grob is just left of any other note; if not set, the value of `first-note` gets used

`right-edge`
: used when the grob is the last item on the line (only compatible with the `extra-space` spacing style)

Choices for *`spacing-style`* are:

`extra-space`
: Put this much space between the two grobs. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed.

`minimum-space`
: Put at least this much space between the left sides of both grobs, without allowing them to collide. The space is stretchable when paired with `first-note` or `next-note`; otherwise it is fixed. Not compatible with `right-edge`.

`fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much fixed space between the grob and the note.

`minimum-fixed-space`
: Only compatible with `first-note` and `next-note`. Put at least this much fixed space between the left side of the

grob and the left side of the note, without allowing them to collide.

`semi-fixed-space`
: Only compatible with `first-note` and `next-note`. Put this much space between the grob and the note, such that half of the space is fixed and half is stretchable.

Rules for this spacing are much more complicated than this. See [Wanske] page 126–134, [Ross] page 143–147.

`space-to-barline` (boolean)
: If set, the distance between a note and the following non-musical column will be measured to the bar line instead of to the beginning of the non-musical column. If there is a clef change followed by a bar line, for example, this means that we will try to space the non-musical column as though the clef is not there.

`spacing-increment` (dimension, in staff space)
: The unit of length for note-spacing. Typically, the width of a note head. See also Section "spacing-spanner-interface" in *Internals Reference*.

`spacing-pair` (pair)
: A pair of alignment symbols which set an object's spacing relative to its left and right `BreakAlignment`s.

For example, a `MultiMeasureRest` will ignore prefatory items at its bounds (i.e., clefs, key signatures and time signatures) using the following override:

```
\override MultiMeasureRest.spacing-pair =
                #'(staff-bar . staff-bar)
```

`spanner-id` (index or symbol)
: An identifier to distinguish concurrent spanners.

`springs-and-rods` (boolean)
: Dummy variable for triggering spacing routines.

`stacking-dir` (direction)
: Stack objects in which direction?

`staff-affinity` (direction)
: The direction of the staff to 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. 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.

`staff-padding` (dimension, in staff space)
: Maintain this much space between reference points and the staff. Its effect is to align objects of differing sizes (like the dynamics **p** and **f**) on their baselines.

`staff-position` (number)
: Vertical position, measured in half staff spaces, counted from the middle line.

`staff-space` (dimension, in staff space)
: Amount of space between staff lines, expressed in global `staff-space`.

`staff-staff-spacing` (list)
: When applied to a staff-group's `StaffGrouper` grob, this spacing alist controls the distance between consecutive staves within the staff-group. When applied to a

staff's `VerticalAxisGroup` grob, it controls the distance between the staff and the nearest staff below it in the same system, replacing any settings inherited from the `StaffGrouper` grob of the containing staff-group, if there is one. This property remains in effect even when non-staff lines appear between staves. The alist can contain the following 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.
- `minimum-distance` – the smallest allowable vertical distance, measured in staff-spaces, between the reference points of the two items, when compressing is in effect.
- `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).

`staffgroup-staff-spacing` (list)
: The spacing alist controlling 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 exist between the two staves. If the `staff-staff-spacing` property of the staff's `VerticalAxisGroup` grob is set, that is used instead. See `staff-staff-spacing` for a description of the alist structure.

`stem-attachment` (pair of numbers)
: An `(x . y)` pair where the stem attaches to the notehead.

`stem-begin-position` (number)
: User override for the begin position of a stem.

`stem-spacing-correction` (number)
: Optical correction amount for stems that are placed in tight configurations. For opposite directions, this amount is the correction for two normal sized stems that overlap completely.

`stemlet-length` (number)
: How long should be a stem over a rest?

`stencil` (stencil)
: The symbol to print.

`stencils` (list)
: Multiple stencils, used as intermediate value.

`strict-grace-spacing` (boolean)
: If set, main notes are spaced normally, then grace notes are put left of the musical columns for the main notes.

`strict-note-spacing` (boolean)
: If set, unbroken columns with non-musical material (clefs, bar lines, etc.) are not spaced separately, but put before musical columns.

`stroke-style` (string)
: Set to `"grace"` to turn stroke through flag on.

`style` (symbol)
: This setting determines in what style a grob is typeset. Valid choices depend on the `stencil` callback reading this property.

`text` (markup)
: Text markup. See Section "Formatting text" in *Notation Reference*.

`text-direction` (direction)
: This controls the ordering of the words. The default `RIGHT` is for roman text. Arabic or Hebrew should use `LEFT`.

`thick-thickness` (number)
: Thickness of the thick line in a bar line, expressed as a multiple of the default staff-line thickness (i.e. the visual output is *not* influenced by changes to `Staff.StaffSymbol.thickness`).

`thickness` (number)
: For grobs made up of lines, this is the thickness of the line. For slurs and ties, this is the distance between the two arcs of the curve's outline at its thickest point, not counting the diameter of the virtual "pen" that draws the arcs. This property is expressed as a multiple of the current staff-line thickness (i.e. the visual output is influenced by changes to `Staff.StaffSymbol.thickness`).

`tie-configuration` (list)
: List of `(position . dir)` pairs, indicating the desired tie configuration, where *position* is the offset from the center of the staff in staff space and *dir* indicates the direction of the tie (`1`=>up, `-1`=>down, `0`=>center). A non-pair entry in the list causes the corresponding tie to be formatted automatically.

`to-barline` (boolean)
: If true, the spanner will stop at the bar line just before it would otherwise stop.

`toward-stem-shift` (number)
: Amount by which scripts are shifted toward the stem if their direction coincides with the stem direction. `0.0` means centered on the note head (the default position of most scripts); `1.0` means centered on the stem. Interpolated values are possible.

`toward-stem-shift-in-column` (number)
: Amount by which a script is shifted toward the stem if its direction coincides with the stem direction and it is associated with a `ScriptColumn` object. `0.0` means centered on the note head (the default position of most scripts); `1.0` means centered on the stem. Interpolated values are possible.

`transparent` (boolean)
: This makes the grob invisible.

`tuplet-slur` (boolean)
: Draw a slur instead of a bracket for tuplets.

`uniform-stretching` (boolean)
: If set, items stretch proportionally to their natural separation based on durations. This looks better in complex polyphonic patterns.

`usable-duration-logs` (list)
: List of `duration-log`s that can be used in typesetting the grob.

`use-skylines` (boolean)
: Should skylines be used for side positioning?

`used` (boolean)
: If set, this spacing column is kept in the spacing problem.

`vertical-skylines` (pair of skylines)
: Two skylines, one above and one below this grob.

`voiced-position` (number)
: The staff-position of a voiced `Rest`, negative if the rest has `direction DOWN`.

`when` (moment)
: Global time step associated with this column.

`whiteout` (boolean-or-number)
: If a number or true, the grob is printed over a white background to white-out underlying material, if the grob is visible. A number indicates how far the white background extends beyond the bounding box of the grob as a multiple of the staff-line thickness. The `LyricHyphen` grob uses a special implementation of whiteout: A positive number indicates how far the white background extends beyond the bounding box in multiples of `line-thickness`. The shape of the background is determined by `whiteout-style`. Usually `#f` by default.

`whiteout-style` (symbol)
: Determines the shape of the `whiteout` background. Available are `'outline`, `'rounded-box`, and the default `'box`. There is one exception: Use `'special` for `LyricHyphen`.

`width` (dimension, in staff space)
: The width of a grob measured in staff space.

`word-space` (dimension, in staff space)
: Space to insert between words in texts.

`X-align-on-main-noteheads` (boolean)
: If true, this grob will ignore suspended noteheads when aligning itself on NoteColumn.

`X-extent` (pair of numbers)
: Extent (size) in the X direction, measured in staff-space units, relative to object's reference point.

`X-offset` (number)
: The horizontal amount that this object is moved relative to its X-parent.

`X-positions` (pair of numbers)
: Pair of X staff coordinates of a spanner in the form `(`*`left . right`*`)`, where both *left* and *right* are in `staff-space` units of the current staff.

`Y-extent` (pair of numbers)
: Extent (size) in the Y direction, measured in staff-space units, relative to object's reference point.

`Y-offset` (number)
: The vertical amount that this object is moved relative to its Y-parent.

`zigzag-length` (dimension, in staff space)
: The length of the lines of a zigzag, relative to `zigzag-width`. A value of `1` gives 60-degree zigzags.

`zigzag-width` (dimension, in staff space)
: The width of one zigzag squiggle. This number is adjusted slightly so that the spanner line can be constructed from a whole number of squiggles.

3.4 Internal backend properties

`accidental-grob` (graphical (layout) object)
: The accidental for this note.

`accidental-grobs` (list)
: An alist with (*notename . groblist*) entries.

`add-cauda` (boolean)
: Does this flexa require an additional cauda on the left side?

`add-join` (boolean)
: Is this ligature head-joined with the next one by a vertical line?

`add-stem` (boolean)
: Is this ligature head a virga and therefore needs an additional stem on the right side?

`adjacent-pure-heights` (pair)
: A pair of vectors. Used by a `VerticalAxisGroup` to cache the `Y-extent`s of different column ranges.

`adjacent-spanners` (array of grobs)
: An array of directly neighboring dynamic spanners.

`all-elements` (array of grobs)
: An array of all grobs in this line. Its function is to protect objects from being garbage collected.

`annotation` (string)
: Annotate a grob for debug purposes.

`ascendens` (boolean)
: Is this neume of ascending type?

`auctum` (boolean)
: Is this neume liquescentically augmented?

`axis-group-parent-X` (graphical (layout) object)
: Containing X axis group.

`axis-group-parent-Y` (graphical (layout) object)
: Containing Y axis group.

`bars` (array of grobs)
: An array of bar line pointers.

`beam` (graphical (layout) object)
: A pointer to the beam, if applicable.

`beam-segments` (list)
: Internal representation of beam segments.

`begin-of-line-visible` (boolean)
: Set to make `ChordName` or `FretBoard` be visible only at beginning of line or at chord changes.

`bezier` (graphical (layout) object)
: A pointer to a Bézier curve, for use by control points and polygons.

`bound-alignment-interfaces` (list)
: Interfaces to be used for positioning elements that align with a column.

`bounded-by-me` (array of grobs)
: An array of spanners that have this column as start/begin point. Only columns that have grobs or act as bounds are spaced.

`bracket` (graphical (layout) object)
: The bracket for a number.

`bracket-text` (graphical (layout) object)
: The text for an analysis bracket.

`c0-position` (integer)
: An integer indicating the position of middle C.

`cause` (any type)
: Any kind of causation objects (i.e., music, or perhaps translator) that was the cause for this grob.

`cavum` (boolean)
: Is this neume outlined?

`columns` (array of grobs)
: An array of grobs, typically containing `PaperColumn` or `NoteColumn` objects.

`concurrent-hairpins` (array of grobs)
: All concurrent hairpins.

`conditional-elements` (array of grobs)
: Internal use only.

`context-info` (integer)
: Within a ligature, the final glyph or shape of a head may be affected by the left and/or right neighbour head. `context-info` holds for each head such information about the left and right neighbour, encoded as a bit mask.

`covered-grobs` (array of grobs)
: Grobs that could potentially collide with a beam.

`cross-staff` (boolean)
: True for grobs whose `Y-extent` depends on inter-staff spacing. The extent is measured relative to the grobs's parent staff (more generally, its `VerticalAxisGroup`) so this boolean flags grobs that are not rigidly fixed to their parent staff. Beams that join notes from two staves are `cross-staff`. Grobs that are positioned around such beams are also `cross-staff`. Grobs that are grouping objects, however, like `VerticalAxisGroups` will not in general be marked `cross-staff` when some of the members of the group are `cross-staff`.

`delta-position` (number)
: The vertical position difference.

`deminutum` (boolean)
: Is this neume deminished?

`descendens` (boolean)
: Is this neume of descendent type?

`direction-source` (graphical (layout) object)
: In case `side-relative-direction` is set, which grob to get the direction from.

`display-cautionary` (boolean)
: Should the grob be displayed as a cautionary grob?

`dot` (graphical (layout) object)
: A reference to a `Dots` object.

`dots` (array of grobs)
: Multiple `Dots` objects.

`elements` (array of grobs)
: An array of grobs; the type is depending on the grob where this is set in.

`encompass-objects` (array of grobs)
: Objects that a slur should avoid in addition to notes and stems.

`figures` (array of grobs)
: Figured bass objects for continuation line.

`flag` (graphical (layout) object)
: A pointer to a `Flag` object.

`flexa-height` (dimension, in staff space)
: The height of a flexa shape in a ligature grob (in `staff-space` units).

`flexa-interval` (integer)
: The interval spanned by the two notes of a flexa shape (1 is a second, 7 is an octave).

`flexa-width` (dimension, in staff space)
: The width of a flexa shape in a ligature grob (in `staff-space` units).

`font` (font metric)
: A cached font metric object.

`footnote-stencil` (stencil)
: The stencil of a system's footnotes.

`footnotes-after-line-breaking` (array of grobs)
: Footnote grobs of a broken system.

`footnotes-before-line-breaking` (array of grobs)
: Footnote grobs of a whole system.

`forced` (boolean)
: Manually forced accidental.

`french-beaming-stem-adjustment` (dimension, in staff space)
: Stem will be shortened by this amount of space in case of French beaming style.

`glissando-index` (integer)
: The index of a glissando in its note column.

`grace-spacing` (graphical (layout) object)
: A run of grace notes.

`has-span-bar` (pair)
: A pair of grobs containing the span bars to be drawn below and above the staff. If no span bar is in a position, the respective element is set to `#f`.

`head-width` (dimension, in staff space)
: The width of this ligature head.

`heads` (array of grobs)
: An array of note heads.

`ideal-distances` (list)
: (*obj* . (*dist* . *strength*)) pairs.

`important-column-ranks` (vector)
: A cache of columns that contain `items-worth-living` data.

`in-note-direction` (direction)
: Direction to place in-notes above a system.

`in-note-padding` (number)
: Padding between in-notes.

`in-note-stencil` (stencil)
: The stencil of a system's in-notes.

`inclinatum` (boolean)
: Is this neume an inclinatum?

`index` (non-negative, exact integer)
: For some grobs in a group, this is a number associated with the grob.

`interfaces` (list)
: A list of symbols indicating the interfaces supported by this object. It is initialized from the `meta` field.

`items-worth-living` (array of grobs)
: An array of interesting items. If empty in a particular staff, then that staff is erased.

`keep-alive-with` (array of grobs)
: An array of other `VerticalAxisGroup`s. If any of them are alive, then we will stay alive.

`least-squares-dy` (number)
: The ideal beam slope, without damping.

`left-items` (array of grobs)
: Grobs organized on the left by a spacing object.

`left-neighbor` (graphical (layout) object)
: The right-most column that has a spacing-wish for this column.

`ligature-flexa` (boolean)
: request joining note to the previous one in a flexa.

`linea` (boolean)
: Attach vertical lines to this neume?

`make-dead-when` (array of grobs)
: An array of other `VerticalAxisGroup`s. If any of them are alive, then we will turn dead.

`maybe-loose` (boolean)
: Used to mark a breakable column that is loose if and only if it is in the middle of a line.

`melody-spanner` (graphical (layout) object)
: The `MelodyItem` object for a stem.

`meta` (list) Provide meta information. It is an alist with the entries `name` and `interfaces`.

`minimum-distances` (list)
: A list of rods that have the format `(`*`obj . dist`*`)`.

`minimum-translations-alist` (list)
: An list of translations for a given start and end point.

`neighbors` (array of grobs)
: The X-axis neighbors of a grob. Used by the pure-from-neighbor-interface to determine various grob heights.

`normal-stems` (array of grobs)
: An array of visible stems.

`note-collision` (graphical (layout) object)
: The `NoteCollision` object of a dot column.

`note-columns` (array of grobs)
: An array of `NoteColumn` grobs.

`note-head` (graphical (layout) object)
: A single note head.

`note-heads` (array of grobs)
: An array of note head grobs.

`numbering-assertion-function` (any type)
: The function used to assert that footnotes are receiving correct automatic numbers.

`oriscus` (boolean)
: Is this neume an oriscus?

`pedal-text` (graphical (layout) object)
: A pointer to the text of a mixed-style piano pedal.

`pes-or-flexa` (boolean)
: Shall this neume be joined with the previous head?

`positioning-done` (boolean)
: Used to signal that a positioning element did its job. This ensures that a positioning is only done once.

`prefix-set` (number)
: A bit mask that holds all Gregorian head prefixes, such as `\virga` or `\quilisma`.

`primitive` (integer)
: A pointer to a ligature primitive, i.e., an item similar to a note head that is part of a ligature.

`pure-relevant-grobs` (array of grobs)
: All the grobs (items and spanners) that are relevant for finding the `pure-Y-extent`

`pure-relevant-items` (array of grobs)
: A subset of elements that are relevant for finding the `pure-Y-extent`.

`pure-relevant-spanners` (array of grobs)
: A subset of elements that are relevant for finding the `pure-Y-extent`.

`pure-Y-common` (graphical (layout) object)
: A cache of the `common_refpoint_of_array` of the `elements` grob set.

`pure-Y-extent` (pair of numbers)
: The estimated height of a system.

`pure-Y-offset-in-progress` (boolean)
: A debugging aid for catching cyclic dependencies.

`quantize-position` (boolean)
: If set, a vertical alignment is aligned to be within staff spaces.

`quantized-positions` (pair of numbers)
: The beam positions after quanting.

`quilisma` (boolean)
: Is this neume a quilisma?

`rest` (graphical (layout) object)
: A pointer to a `Rest` object.

`rest-collision` (graphical (layout) object)
: A rest collision that a rest is in.

`rests` (array of grobs)
: An array of rest objects.

`right-items` (array of grobs)
: Grobs organized on the right by a spacing object.

`right-neighbor` (graphical (layout) object)
: See `left-neighbor`.

`script-column` (graphical (layout) object)
: A `ScriptColumn` associated with a `Script` object.

`script-stencil` (pair)
: A pair `(`*`type`* `.` *`arg`*`)` which acts as an index for looking up a `Stencil` object.

`scripts` (array of grobs)
: An array of `Script` objects.

`shorten` (dimension, in staff space)
: The amount of space that a stem is shortened. Internally used to distribute beam shortening over stems.

`side-support-elements` (array of grobs)
: The side support, an array of grobs.

`slur` (graphical (layout) object)
: A pointer to a `Slur` object.

`space-increment` (dimension, in staff space)
: The amount by which the total duration of a multimeasure rest affects horizontal spacing. Each doubling of the duration adds `space-increment` to the length of the bar.

`spacing` (graphical (layout) object)
: The spacing spanner governing this section.

`spacing-wishes` (array of grobs)
: An array of note spacing or staff spacing objects.

`span-start` (boolean)
: Is the note head at the start of a spanner?

`spanner-broken` (boolean)
: Indicates whether spanner alignment should be broken after the current spanner.

`spanner-placement` (direction)
: The place of an annotation on a spanner. `LEFT` is for the first spanner, and `RIGHT` is for the last. `CENTER` will place it on the broken spanner that falls closest to the center of the length of the entire spanner, although this behavior is unpredictable in situations with lots of rhythmic diversity. For predictable results, use `LEFT` and `RIGHT`.

`staff-grouper` (graphical (layout) object)
: The staff grouper we belong to.

`staff-symbol` (graphical (layout) object)
: The staff symbol grob that we are in.

`stem` (graphical (layout) object)
: A pointer to a `Stem` object.

`stem-info` (pair)
: A cache of stem parameters.

`stems` (array of grobs)
: An array of stem objects.

`sticky-host` (graphical (layout) object)
: The grob that a sticky grob attaches to.

`stropha` (boolean)
: Is this neume a stropha?

`system-Y-offset` (number)
: The Y-offset (relative to the bottom of the top-margin of the page) of the system to which this staff belongs.

`tie` (graphical (layout) object)
: A pointer to a `Tie` object.

`ties` (array of grobs)
: A grob array of `Tie` objects.

`tremolo-flag` (graphical (layout) object)
: The tremolo object on a stem.

`tuplet-number` (graphical (layout) object)
: The number for a bracket.

`tuplet-start` (boolean)
: Is stem at the start of a tuplet?

`tuplets` (array of grobs)
: An array of smaller tuplet brackets.

`vertical-alignment` (graphical (layout) object)
: The VerticalAlignment in a System.

`vertical-skyline-elements` (array of grobs)
: An array of grobs used to create vertical skylines.

`virga` (boolean)
: Is this neume a virga?

`X-common` (graphical (layout) object)
: Common reference point for axis group.

`x-offset` (dimension, in staff space)
: Extra horizontal offset for ligature heads.

`Y-common` (graphical (layout) object)
: See `X-common`.

4 Scheme functions

`add-bar-glyph-print-procedure` *glyph proc* [Function]
Specify the single glyph *glyph* that calls print procedure *proc*. The procedure *proc* has to be defined in the form `(make-...-bar-line grob extent)` even if the *extent* is not used within the routine.

`ly:add-context-mod` *contextmods modification* [Function]
Adds the given context *modification* to the list *contextmods* of context modifications.

`add-grace-property` *context-name grob sym val* [Function]
Set *sym*=*val* for *grob* in *context-name*.

`ly:add-interface` *iface desc props* [Function]
Add a new grob interface. *iface* is the interface name, *desc* is the interface description, and *props* is the list of user-settable properties for the interface.

`ly:add-listener` *callback disp cl* [Function]
Add the single-argument procedure *callback* as listener to the dispatcher *disp*. Whenever *disp* hears an event of class *cl*, it calls *callback* with it.

`add-music-fonts` *node family name brace design-size-alist factor* [Function]
Set up music fonts.

Arguments:

- *node* is the font tree to modify.
- *family* is the family name of the music font.
- *name* is the basename for the music font. *name*`-<designsize>.otf` should be the music font.
- *brace* is the basename for the brace font. *brace*`-brace.otf` should have piano braces.
- *design-size-alist* is a list of `(`*rounded*` . `*designsize*`)`. *rounded* is a suffix for font filenames, while *designsize* should be the actual design size. The latter is used for text fonts loaded through pango/fontconfig.
- *factor* is a size factor relative to the default size that is being used. This is used to select the proper design size for the text fonts.

`add-new-clef` *clef-name clef-glyph clef-position transposition c0-position* [Function]
Append the entries for a clef symbol to supported clefs and `c0-pitch-alist`.

`ly:add-option` *sym val description* [Function]
Add a program option *sym*. *val* is the default value and *description* is a string description.

`add-simple-time-signature-style` *style proc* [Function]
Specify the procedure *proc* returning markup for a time signature style *style*. The procedure is called with one argument, the pair `(`*numerator*` . `*denominator*`)`.

`add-stroke-glyph` *stencil grob dir stroke-style flag-style* [Function]
Load and add a stroke (represented by a glyph in the font) to the given flag stencil.

`add-stroke-straight` *stencil grob dir log stroke-style offset length thickness stroke-thickness* [Function]
Add the stroke for acciaccatura to the given flag stencil. The stroke starts for up-flags at 'upper-end-of-flag + (0,length/2)' and ends at '(0, vertical-center-of-flag-end) - (flag-x-width/2, flag-x-width + flag-thickness)'. Here 'length' is the whole length, while 'flag-x-width' is just the x extent and thus depends on the angle! Other combinations don't look as good.

For down-stems the y coordinates are simply mirrored.

`alist->hash-table` *lst* [Function]
Convert alist *lst* to a table.

`ly:all-grob-interfaces` [Function]
Return the hash table with all grob interface descriptions.

`ly:all-options` [Function]
Get all option settings in an alist.

`ly:all-output-backend-commands` [Function]
Return the list of extra output backend commands that are used internally in file `lily/stencil-interpret.cc`.

`ly:all-stencil-commands` [Function]
Return the list of stencil commands that can be defined in the output modules (in files `output-*.scm`).

`ly:all-stencil-expressions` [Function]
Return all symbols recognized as stencil expressions.

`allow-volta-hook` *bar-glyph* [Function]
Allow the volta bracket hook being drawn over bar line *bar-glyph*.

`alterations-in-key` *pitch-list* [Function]
Count number of sharps minus number of flats.

`ly:angle` *x y* [Function]
Calculate angle in degrees of given vector. With one argument, *x* is a number pair indicating the vector. With two arguments, *x* and *y* specify the respective coordinates.

`angle-0-2pi` *angle* [Function]
Take *angle* (in radians) and map it between 0 and 2pi.

`angle-0-360` *angle* [Function]
Take *angle* (in degrees) and map it between 0 and 360 degrees.

`arrow-stencil` *x y thick staff-space grob* [Function]
Return a right-pointing, filled arrow-head, where *x* determines the basic horizontal position and *y* determines the basic vertical position. Both values are adjusted using *staff-space*, which is `StaffSymbol`'s staff space. *thick* is the used line thickness.

`arrow-stencil-maker` *start? end?* [Function]
Return a function drawing a line from current point to `destination`, with optional arrows of `max-size` on start and end controlled by *start?* and *end?*.

`ly:assoc-get` *key alist default-value strict-checking* [Function]
Return value if *key* in *alist*, else *default-value* (or `#f` if not specified). If *strict-checking* is set to `#t` and *key* is not in *alist*, a programming error is output.

`ly:axis-group-interface::add-element` *grob grob-element* [Function]
Add *grob-element* to the axis group *grob*. In particular, *grob* becomes parent to *grob-element* on all axes supported by *grob*, unless the parents are already set.

`ly:bar-line::calc-anchor` *grob* [Function]
Calculate the anchor position of a bar line. The anchor is used for the correct placement of bar numbers, etc.

`bar-line::calc-break-visibility` *grob* [Function]
Calculate the visibility of a bar line at line breaks.

`bar-line::calc-glyph-name` *grob* [Function]
Return the name of the bar line glyph printed by *grob*. This function is a wrapper for `bar-line::calc-glyph-name-for-direction`.

`bar-line::calc-glyph-name-for-direction` *glyph dir* [Function]
Return the glyph name of the bar line glyph object for direction *dir* (`LEFT` = end of line, `CENTER` = middle of line, `RIGHT` = start of line).

`bar-line::compound-bar-line` *grob bar-glyph extent* [Function]
Build the bar line stencil.

`bar-line::draw-filled-box` *x-ext y-ext thickness extent grob* [Function]
Return a straight bar line created by `ly:round-filled-box` looking at *x-ext*, *y-ext*, *thickness*. The blot is calculated by `bar-line::calc-blot`, which needs *extent* and *grob*. *y-ext* is not necessarily of same value as *extent*.

`ly:bar-line::print` *grob* [Function]
The print routine for bar lines.

`bar-line::widen-bar-extent-on-span` *grob extent* [Function]
Widen the bar line *extent* towards span bars adjacent to grob *grob*.

`base-length` *time-signature time-signature-settings* [Function]
Get `baseMoment` rational value for *time-signature* from *time-signature-settings*.

`ly:basic-progress` *str rest* [Function]
A Scheme callable function to issue a basic progress message *str*. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`beam-exceptions` *time-signature time-signature-settings* [Function]
Get `beamExceptions` value for *time-signature* from *time-signature-settings*.

`beat-structure` *base-length time-signature time-signature-settings* [Function]
Get `beatStructure` value in *base-length* units for *time-signature* from *time-signature-settings*.

`bend::arrow-head-stencil` *thickness x-y-coords height width dir* [Function]
Return an arrow head stencil, calculated from the given dimensions *height* and *width*, and translated to *x-y-coords*, the end of the bend-spanners (curved) line.

`bend::calc-bend-x-begin` *bend-spanner bounding-noteheads factor quarter-tone-diffs* [Function]
Calculate the starting values in x direction of the bend. After a line break, the values from the right bound are taken minus 1.5 staff spaces. For bends-down or if grob property `'style` equals to `'pre-bend`, `'hold` or `'pre-bend-hold`, `interval-center` is applied the topmost note head of the starting note heads. In any other case the right edge of the starting note head is used. The value of `BendSpanner.details.horizontal-left-padding` is added, which may be changed by an appropriate override. Returns a list of the same length as the amount of bend-starting note heads.

`bend::calc-bend-x-end` *bend-spanner top-left-tab-nhd top-right-tab-nhd* [Function]
Calculate the ending x coordinate of *bend-spanner*. At the line end, take the items of `BreakAlignGroup` into account and a little bit of padding. Ends an unbroken spanner or the last of a broken one in the middle of the topmost note head of its bounding note column.

`bend::target-cautionary` *spanner* [Function]
Set `'display-cautionary` of all relevant note heads of spanners right bound to true. As a result they appear parenthesized. This procedure is the default value of `'before-line-breaking`.

`bend::text-string` *spanner* [Function]
Take a spanner grob and calculate a list with the quarter tone diffs between the pitches of starting and ending bound. Because bending to different amounts is very unlikely, only the first element of this list is returned as a string.

`bend-spanner::print` *grob* [Function]
Return the final stencil. A line and curve, an arrow head and a text representing the amount a string is bent.

`ly:book?` *x* [Function]
Is *x* a smob of class `Book`?

`ly:book-add-bookpart!` *book-smob book-part* [Function]
Add *book-part* to *book-smob* book part list.

`ly:book-add-score!` *book-smob score* [Function]
Add *score* to *book-smob* score list.

`ly:book-book-parts` *book* [Function]
Return book parts in *book*.

`book-first-page` *layout props* [Function]
Return the `'first-page-number` of the entire book.

`ly:book-header` *book* [Function]
Return header in *book*.

`ly:book-paper` *book* [Function]
Return paper in *book*.

`ly:book-process` *book-smob default-paper default-layout output* [Function]
Print book. *output* is passed to the backend unchanged. For example, it may be a string (for file based outputs) or a socket (for network based output).

`ly:book-process-to-systems` *book-smob default-paper default-layout output* [Function]
Print book. *output* is passed to the backend unchanged. For example, it may be a string (for file based outputs) or a socket (for network based output).

`ly:book-scores` *book* [Function]
Return scores in *book*.

`ly:book-set-header!` *book module* [Function]
Set the book header.

`box-grob-stencil` *grob* [Function]
Make a box of exactly the extents of the grob. The box precisely encloses the contents.

`box-stencil` *stencil thickness padding* [Function]
Add a box around *stencil*, producing a new stencil.

`ly:bp` *num* [Function]
num bigpoints (1/72th inch).

`ly:bracket` *a iv t p* [Function]
Make a bracket in direction *a*. The extent of the bracket is given by *iv*. The wings protrude by an amount of *p*, which may be negative. The thickness is given by *t*.

`bracketify-stencil` *stil axis thick protrusion padding* [Function]
Add brackets around *stil*, producing a new stencil.

`break-alignable-interface::self-alignment-of-anchor` *g* [Function]
Return a value for *g*'s `self-alignment-X` that will place *g* on the same side of the reference point defined by a `break-aligned` item such as a `Clef`.

`break-alignable-interface::self-alignment-opposite-of-anchor` *g* [Function]
Return a value for *g*'s `self-alignment-X` that will place *g* on the opposite side of the reference point defined by a `break-aligned` item such as a `Clef`.

`break-alignment-list` *end-of-line middle begin-of-line* [Function]
Return a callback that calculates a value based on a grob's break direction.

`ly:broadcast` *disp ev* [Function]
Send the stream event *ev* to the dispatcher *disp*.

`ly:cairo-output-stencil` *basename stencil paper formats* [Function]
dump a single stencil through the Cairo backend

`ly:cairo-output-stencils` *basename stencils header paper formats* [Function]
dump book through cairo backend

`calc-harmonic-pitch` *pitch music* [Function]
Calculate the harmonic pitches in *music* given *pitch* as the non-harmonic pitch.

`ly:camel-case->lisp-identifier` *name-sym* [Function]
Convert `FooBar_Bla` to `foo-bar-bla` style symbol.

`car<` *a b* [Function]
Return a comparator function that applies *key* to the two elements and compares the results using *cmp*. Especially useful for sorting.

`car<=` *a b* [Function]
Return a comparator function that applies *key* to the two elements and compares the results using *cmp*. Especially useful for sorting.

`centered-stencil` *stencil* [Function]
Center stencil *stencil* in both the x and y directions.

`centered-text-interface::print` *grob* [Function]
Print some text between two non-musical columns according to the `spacing-pair` property.

`ly:chain-assoc-get` *key achain default-value strict-checking* [Function]
Return value for *key* from a list of alists *achain*. If no entry is found, return *default-value* or `#f` if *default-value* is not specified. With *strict-checking* set to `#t`, a programming error is output in such cases.

`change-pitches` *music converter* [Function]
Recurse through *music*, applying *converter* to pitches. *converter* is typically a transposer or an inverter (see file `scm/modal-transforms.scm`), but may be user-defined. The converter function must take a single pitch as its argument and return a new pitch. These are LilyPond Scheme pitches, e.g., `(ly:make-pitch 0 2 0)`.

`check-context-path` *path . lambda*:G59* [Function]
Check a context property path specification *path*, a symbol list (or a single symbol), for validity and possibly complete it. Returns the completed specification, or `#f` when rising an error (using optionally *location*).

`ly:check-expected-warnings` [Function]
Check whether all expected warnings have really been triggered.

`check-grob-path` *path . rest* [Function]
Check a grob path specification *path*, a symbol list (or a single symbol), for validity and possibly complete it. Returns the completed specification, or `#f` if invalid, optionally using *location* for an error message. If an optional keyword argument `#:start` *`start`* is given, the parsing starts at the given index in the sequence '`Context.Grob.property.sub-property...`', with the default of '`0`' implying the full path.

If there is no valid first element of *path* fitting at the given path location, an optionally given `#:default` *`default`* is used as the respective element instead without checking it for validity at this position.

The resulting path after possibly prepending *default* can be constrained in length by optional arguments `#:min` *`min`* and `#:max` *`max`*, defaulting to '`1`' and unlimited, respectively.

`check-music-path` *path . rest* [Function]
Check a music property path specification *path*, a symbol list (or a single symbol), for validity and possibly complete it. Returns the completed specification, or `#f` when rising an error (using optionally *location*).

`circle-stencil` *stencil thickness padding* [Function]
Add a circle around *stencil*, producing a new stencil.

`clef-transposition-markup` *oct style* [Function]
The transposition sign formatting function. *oct* is supposed to be a string holding the transposition number, *style* determines the way the transposition number is displayed.

`ly:cm` *num* [Function]
num cm.

`collect-book-music-for-book` *book music* [Function]
Book music handler.

`collect-bookpart-for-book` *book-part* [Function]
Top-level book-part handler.

`collect-music-aux` *score-handler music* [Function]
Pass *music* to *score-handler*, with preprocessing for page layout instructions.

`collect-music-for-book` *music* [Function]
Top-level music handler.

`ly:command-line-code` [Function]
The Scheme code specified on the command line with option `-e`.

`ly:command-line-options` [Function]
The Scheme options specified on the command line with option `-d`.

`ly:connect-dispatchers` *to from* [Function]
Make the dispatcher *to* listen to events from *from*.

`constante-hairpin` *grob* [Function]
Create hairpin based on a list of *coords* in `(cons x y)` form. `x` is the portion of the width consumed for a given line and `y` is the portion of the height. For example, `'((0 . 0) (0.3 . 0.7) (0.8 . 0.9) (1.0 . 1.0))` means that at the point where the hairpin has consumed 30% of its width, it must be at 70% of its height. Once it is to 80% width, it must be at 90% height. It finishes at 100% width and 100% height. If *coords* does not begin with `'(0 . 0)` the final hairpin may have an open tip. For example '(0 . 0.5) will cause an open end of 50% of the usual height.

mirrored? indicates if the hairpin is mirrored over the y axis or if just the upper part is drawn.

Returns a function that accepts a hairpin grob as an argument and draws the stencil based on its coordinates.

```
#(define simple-hairpin
  (elbowed-hairpin '((0 . 0)(1.0 . 1.0)) #t))

\relative c' {
  \override Hairpin #'stencil = #simple-hairpin
  a\p\< a a a\f
}
```

`construct-chord-elements` *root duration modifications* [Function]
Build a chord on *root* using modifiers in *modifications*. `NoteEvents` have duration *duration*.

Notes: Natural 11 is left from chord if not explicitly specified.

Entry point for the parser.

`ly:context?` *x* [Function]
Is *x* a smob of class `Context`?

`ly:context-current-moment` *context* [Function]
Return the current moment of *context*.

`ly:context-def?` *x* [Function]
Is *x* a smob of class `Context_def`?

`ly:context-def-lookup` *def sym val* [Function]
Return the value of *sym* in context definition *def* (e.g., `\Voice`). If no value is found, return *val* or `'()` if *val* is undefined. *sym* can be any of '`default-child`', '`consists`', '`description`', '`aliases`', '`accepts`', '`property-ops`', '`context-name`', '`group-type`'.

`ly:context-def-modify` *def mod* [Function]
Return the result of applying the context-mod *mod* to the context definition *def*. Does not change *def*.

`ly:context-event-source` *context* [Function]
Return `event-source` of context *context*.

`ly:context-events-below` *context* [Function]
Return a `stream-distributor` that distributes all events from *context* and all its subcontexts.

`ly:context-find` *context name* [Function]
Find a parent of *context* that has name or alias *name*. Return `#f` if not found.

`ly:context-grob-definition` *context name* [Function]
Return the definition of *name* (a symbol) within *context* as an alist.

`ly:context-id` *context* [Function]
Return the ID string of *context*, i.e., for `\context Voice = "one" ...` return the string `one`.

`ly:context-matched-pop-property` *context grob cell* [Function]
This undoes a particular `\override`, `\once \override` or `\once \revert` when given the specific alist pair to undo.

`ly:context-mod?` *x* [Function]
Is *x* a smob of class `Context_mod`?

`ly:context-mod-apply!` *context mod* [Function]
Apply the context modification *mod* to *context*.

`ly:context-name` *context* [Function]
Return the name of *context*, i.e., for `\context Voice = "one" ...` return the symbol `Voice`.

`ly:context-output-def` *context* [Function]
Return the output definition of *context*.

`ly:context-parent` *context* [Function]
Return the parent of *context*, `#f` if none.

`ly:context-property` *context sym def* [Function]
Return the value for property *sym* in *context*. If *def* is given, and property value is `'()`, return *def*.

`ly:context-property-where-defined` *context name* [Function]
Return the context above *context* where *name* is defined.

`ly:context-pushpop-property` *context grob eltprop val* [Function]
Do `\temporary \override` or `\revert` operation in *context*. The grob definition *grob* is extended with *eltprop* (if *val* is specified) or reverted (if unspecified).

`ly:context-set-property!` *context name val* [Function]
Set value of property *name* in context *context* to *val*.

`context-spec-music` *m context . lambda*:G70* [Function]
Add `\context` *`context`* `=` *`id`* `\with` *`mods`* to *m*.

`ly:context-unset-property` *context name* [Function]
Unset value of property *name* in context *context*.

`copy-repeat-chord` *original-chord repeat-chord duration event-types* [Function]
Copy all events in *event-types* (be sure to include `rhythmic-events`) from *original-chord* over to *repeat-chord* with their articulations filtered as well. Any duration is replaced with the specified *duration*.

`count-list` *lst* [Function]
Given *lst* as `(E1 E2 . .)`, return `((E1 . 1) (E2 . 2) ...)`.

`create-glyph-flag` *flag-style dir-modifier grob* [Function]
Create a flag stencil by looking up the glyph from the font.

`cross-staff-connect` *stem* [Function]
Set cross-staff property of the stem to this function to connect it to other stems automatically

`cue-substitute` *quote-music* [Function]
Must happen after `quote-substitute`.

`cyclic-base-value` *value cycle* [Function]
Take *value* (for example, an angle) and modulo-maps it between 0 and base *cycle*.

`ly:debug` *str rest* [Function]
A Scheme callable function to issue a debug message *str*. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`default-flag` *grob* [Function]
Create a flag stencil for the stem. Its style is derived from the `'style` Flag property. By default, `lilypond` uses a C++ Function (which is slightly faster) to do exactly the same as this function. However, if one wants to modify the default flags, this function can be used to obtain the default flag stencil, which can then be modified at will. The correct way to do this is:

```
\override Flag #'stencil = #default-flag
\override Flag #'style = #'mensural
```

`ly:default-scale` [Function]
Get the global default scale.

`define-bar-line` *bar-glyph eol-glyph bol-glyph span-glyph* [Function]
Define a bar glyph *bar-glyph* and its substitute at the end of a line (*eol-glyph*), at the beginning of a new line (*bol-glyph*) and as a span bar (*span-glyph*), respectively.

`define-event-class` *class parent* [Function]
Defines a new event `class` derived from `parent`, a previously defined event class.

`define-fonts` *paper define-font define-pango-pf* [Function]
Return a string of all fonts used in *paper*, invoking the functions *define-font* and *define-pango-pf* for producing the actual font definition.

`define-tag-group` *tags* [Function]
Define a tag group consisting of the given *tags*, a list of symbols. Returns `#f` if successful, and an error message if there is a conflicting tag group definition.

`degrees->radians` *angle-degrees* [Function]
Convert the given angle from degrees to radians.

`descend-to-context` *m context . lambda*:G72* [Function]
Like `context-spec-music`, but only descending.

`determine-split-list` *evl1 evl2 chord-range* [Function]
Event lists *evl1* and *evl2* should be ascending. *chord-range* is a pair of numbers `(min . max)` defining the distance in steps between notes that may be combined into a chord or unison.

`determine-string-fret-finger` *context notes specified-info rest* [Function]
Determine string numbers and frets for playing *notes* as a chord, given specified information *specified-info*. *specified-info* is a list with two list elements, specified strings `defined-strings` and specified fingerings `defined-fingers`. Only a fingering of 0 will affect the fret selection, as it specifies an open string. If `defined-strings` is `'()`, the context property `defaultStrings` is used as a list of defined strings. Looks for predefined fretboards if `predefinedFretboardTable` is not `#f`. If *rest* is present, it contains the `FretBoard` grob, and a fretboard gets created. Otherwise, a list of `(string fret finger)` lists is returned.

If the context-property `supportNonIntegerFret` is set `#t`, micro-tones are supported for `TabStaff`, but not not for `FretBoards`.

`ly:dimension?` *d* [Function]
Is *d* a dimension? Used to distinguish length variables from normal numbers.

`ly:dir?` *s* [Function]
Is *s* a direction? Valid directions are `-1`, `0`, or `1`, where `-1` represents left or down, `1` represents right or up, and `0` represents a neutral direction.

`dir-basename` *file . rest* [Function]
Strip suffixes in *rest*, but leave directory component for *file*.

`ly:directed` *direction magnitude* [Function]
Calculate an `(x . y)` pair with optional *magnitude* (defaulting to `1.0`) and *direction* specified either as an angle in degrees or a coordinate pair giving the direction. If *magnitude* is a pair, the respective coordinates are scaled independently, useful for ellipse drawings.

`ly:disconnect-dispatchers` *to from* [Function]
Stop the dispatcher *to* listening to events from *from*.

`ly:dispatcher?` *x* [Function]
Is *x* a smob of class `Dispatcher`?

`display-lily-music` *expr . lambda*:G54* [Function]
Display the music expression *expr* using LilyPond syntax.

`display-music` *music . lambda*:G40* [Function]
Display *music*, not done with `music-map` for clarity of presentation.

`display-scheme-music` *obj . lambda*:G42* [Function]
Display *obj*, typically a music expression, in a friendly fashion, which often can be read back in order to generate an equivalent expression.

`dodecaphonic-no-repeat-rule` *context pitch barnum* [Function]
An accidental rule that typesets an accidental before every note (just as in the dodecaphonic accidental style) *except* if the note is immediately preceded by a note with the same pitch. This is a common accidental style in contemporary notation.

`ly:duration?` *x* [Function]
Is *x* a smob of class `Duration`?

`ly:duration<?` *p1 p2* [Function]
Is *p1* shorter than *p2*?

`ly:duration->string` *dur* [Function]
Convert *dur* to a string.

`ly:duration-compress` *dur factor* [Function]
Compress *dur* by rational *factor*.

`ly:duration-dot-count` *dur* [Function]
Extract the dot count from *dur*.

`duration-dot-factor` *dotcount* [Function]
Given a count of the dots used to extend a musical duration, return the numeric factor by which they increase the duration.

`ly:duration-factor` *dur* [Function]
Extract the compression factor from *dur*. Return it as a pair.

`ly:duration-length` *dur* [Function]
The length of the duration as a `moment`.

`duration-length` *dur* [Function]
Return the overall length of a duration, as a number of whole notes. (Not to be confused with `ly:duration-length`, which returns a less useful `Moment` object.)

`duration-line::calc` *grob* [Function]
Return list of values needed to print a stencil for `DurationLine`.

`duration-line::print` *grob* [Function]
Return the stencil of `DurationLine`.

`ly:duration-log` *dur* [Function]
Extract the duration log from *dur*.

`duration-log-factor` *lognum* [Function]
Given a logarithmic duration number, return the length of the duration, as a number of whole notes.

`ly:duration-scale` *dur* [Function]
Extract the compression factor from *dur*. Return it as a rational.

`duration-visual` *dur* [Function]
Given a duration object, return the visual part of the duration (base note length and dot count), in the form of a duration object with non-visual scale factor 1.

`duration-visual-length` *dur* [Function]
Given a duration object, return the length of the visual part of the duration (base note length and dot count), as a number of whole notes.

`dynamic-text-spanner::before-line-breaking` *grob* [Function]
Monitor left bound of `DynamicTextSpanner` for absolute dynamics. If found, ensure `DynamicText` does not collide with spanner text by changing `'attach-dir` and `'padding`. Reads the `'right-padding` property of `DynamicText` to fine-tune space between the two text elements.

`ly:effective-prefix` [Function]
Return effective prefix. For example, if LilyPond Scheme files are stored in directory `/foo/bar/scm` and PS files in `/foo/bar/ps`, the effective prefix is `/foo/bar`.

`ellipse-stencil` *stencil thickness x-padding y-padding* [Function]
Add an ellipse around *stencil*, padded by the padding pair, producing a new stencil.

`ly:encode-string-for-pdf` *str* [Function]
Encode *str* as either Latin-1 (which is a subset of PDFDocEncoding) or, if that's not possible, as full UTF-16BE with a leading Byte Order Mark (BOM).

`ly:engraver-announce-end-grob` *engraver grob cause* [Function]
Announce the end of a grob (i.e., the end of a spanner) originating from given *engraver* instance, with *grob* being a grob. *cause* should either be another grob or a music event.

`ly:engraver-make-grob` *engraver grob-name cause* [Function]
Create a grob originating from given *engraver* instance, with given *grob-name*, a symbol. *cause* should either be another grob or a music event.

`ly:engraver-make-item` *engraver grob-name cause* [Function]
Same as `ly:engraver-make-grob`, but always create a grob with the `Item` class. This is useful when the same grob definition is used to create grobs of differing classes.

`ly:engraver-make-spanner` *engraver grob-name cause* [Function]
Same as `ly:engraver-make-grob`, but always create a grob with the `Spanner` class. This is useful when the same grob definition is used to create grobs of differing classes.

`ly:engraver-make-sticky` *engraver grob-name host cause* [Function]
Utility function to create a grob sticking to another grob. This acts like either `ly:engraver-make-item` or `ly:engraver-make-spanner`, depending on the class of the host. Additionally, the host is made the parent of the newly created sticky grob on the y axis and, for items, on the x axis. Sticky spanners take their bounds from their host and their end is announced with the end of the host.

Sticky grobs must have the `sticky-grob-interface` interface, see Section "sticky-grob-interface" in *Internals Reference*.

`ly:error` *str rest* [Function]
A Scheme callable function to issue the error *str*. The error is formatted with `format`; *rest* holds the formatting arguments (if any).

`eval-carefully` *symbol module . default* [Function]
Check whether all symbols in expression *symbol* are reachable in module *module*. In that case evaluate, otherwise print a warning and set an optional *default*.

`ly:event?` *obj* [Function]
Is *obj* a proper (non-rhythmic) `Event` object?

`event-chord-notes` *event-chord* [Function]
Return a list of all notes from *event-chord*.

`event-chord-pitches` *event-chord* [Function]
Return a list of all pitches from *event-chord*.

`event-chord-reduce` *music* [Function]
Reduce event chords in *music* to their first note event, retaining only the chord articulations. Returns the modified music.

`event-chord-wrap!` *music* [Function]
Wrap isolated rhythmic events and non-postevent events in *music* inside of an `EventChord`. Chord repeats '`q`' are expanded using the default settings of the parser.

`ly:event-deep-copy` *m* [Function]
Copy *m* and all sub-expressions of *m*.

`event-has-articulation?` *event-type stream-event* [Function]
Is *event-type* in the `articulations` list of *stream-event*?

`ly:event-property` *sev sym val* [Function]
Get the property *sym* of stream event *sev*. If *sym* is undefined, return *val* or `'()` if *val* is not specified.

`ly:event-set-property!` *ev sym val* [Function]
Set property *sym* in event *ev* to *val*.

`expand-repeat-chords!` *event-types music* [Function]
Walk through *music* and fill repeated chords (notable by having a duration in `duration`) with the notes from their respective predecessor chord.

`expand-repeat-notes!` *music* [Function]
Walk through *music* and give pitchless notes (not having a pitch in `pitch` or a drum type in `drum-type`) the pitch(es) from the predecessor note/chord if available.

`ly:expect-warning` *str rest* [Function]
A Scheme callable function to register a warning to be expected and subsequently suppressed. If the warning is not encountered, a warning about the missing warning is shown. The message should be translated with `(_ ...)` and changing parameters given after the format string.

`extract-beam-exceptions` *music* [Function]
Create a value useful for setting `beamExceptions` from *music*.

`extract-music` *music pred?* [Function]
Return a flat list of all music matching *pred?* inside of *music*, not recursing into matches themselves.

`extract-named-music` *music music-name* [Function]
Return a flat list of all music named *music-name* (either a single event symbol or a list of alternatives) inside of *music*, not recursing into matches themselves.

`ly:extract-subfont-from-collection` *collection-file-name idx subfont-file-name* [Function]
Extract the subfont of index *idx* in TrueType collection (TTC) or OpenType/CFF collection (OTC) file *collection-file-name* and write it to file *subfont-file-name*.

`extract-typed-music` *music type* [Function]
Return a flat list of all music with *type* (either a single type symbol or a list of alternatives) inside of *music*, not recursing into matches themselves.

`ly:find-file` *name* [Function]
Return the absolute file name of *name*, or `#f` if not found.

`find-named-props` *prop-name grob-descriptions* [Function]
Used by `\magnifyMusic` and `\magnifyStaff`. If *grob-descriptions* is equal to the `all-grob-descriptions` alist (defined in `scm/define-grobs.scm`), this finds all grobs that can have a value for the *prop-name* property, and return them as a list in the following format:

```
'((grob prop-name)
  (grob prop-name)
  ...)
```

`find-pitch-entry` *keysig pitch accept-global accept-local* [Function]
Return the first entry in *keysig* that matches *pitch* by notename and octave. Alteration is not considered. *accept-global* states whether key signature entries should be included. *accept-local* states whether local accidentals should be included. If no matching entry is found, `#f` is returned.

`finger-glide::print` *grob* [Function]
The stencil printing procedure for grob `FingerGlideSpanner`. Depending on the grob property `style` several forms of appearance are printed. Possible settings for grob property `style` are `zigzag`, `trill`, `dashed-line`, `dotted-line`, `stub-left`, `stub-right`, `stub-both`, `bow`, `none` and `line`, which is the default.

`first-assoc` *keys lst* [Function]
Return first successful assoc of key from *keys* in *lst*.

`first-member` *members lst* [Function]
Return first successful member (of member) from *members* in *lst*.

`flared-hairpin` *grob* [Function]
Create hairpin based on a list of *coords* in `(cons x y)` form. `x` is the portion of the width consumed for a given line and `y` is the portion of the height. For example, `'((0 . 0) (0.3 . 0.7) (0.8 . 0.9) (1.0 . 1.0))` means that at the point where the hairpin has consumed 30% of its width, it must be at 70% of its height. Once it is to 80% width, it must be at 90% height. It finishes at 100% width and 100% height. If *coords* does not begin with `'(0 . 0)` the final hairpin may have an open tip. For example '(0 . 0.5) will cause an open end of 50% of the usual height.

mirrored? indicates if the hairpin is mirrored over the y axis or if just the upper part is drawn.

Returns a function that accepts a hairpin grob as an argument and draws the stencil based on its coordinates.

```
#(define simple-hairpin
  (elbowed-hairpin '((0 . 0)(1.0 . 1.0)) #t))

\relative c' {
  \override Hairpin #'stencil = #simple-hairpin
  a\p\< a a a\f
}
```

`flat-flag` *grob* [Function]
Flat flag style. The angles of the flags are both 0 degrees.

`flatten-list` *x* [Function]
Unnest list.

`flip-stencil` *axis stil* [Function]
Flip stencil *stil* in the direction of *axis*. Value `X` (or `0`) for *axis* flips it horizontally. Value `Y` (or `1`) flips it vertically. *stil* is flipped in place; its position, the coordinates of its bounding box, remains the same.

`fold-some-music` *pred? proc init music* [Function]
This works recursively on music like `fold` does on a list, calling '`(`*`pred?`* `music)`' on every music element. If `#f` is returned for an element, it is processed recursively with the same initial value of '`previous`', otherwise '`(`*`proc`* `music previous)`' replaces '`previous`' and no recursion happens. The top *music* is processed using *init* for '`previous`'.

`ly:font-config-add-directory` *dir* [Function]
Add directory *dir* to FontConfig.

`ly:font-config-add-font` *font* [Function]
Add font *font* to FontConfig.

`ly:font-config-display-fonts` [Function]
Dump a list of all fonts visible to FontConfig.

`ly:font-config-get-font-file` *name* [Function]
Get the file for font *name*, as found by FontConfig.

`ly:font-design-size` *font* [Function]
Given the font metric *font*, return the design size, relative to the current `output-scale`.

`ly:font-file-name` *font* [Function]
Given the font metric *font*, return the corresponding file name.

`ly:font-get-glyph` *font name* [Function]
Return a stencil from *font* for the glyph named *name*. If the glyph is not available, return an empty stencil.

Note that this command can only be used to access glyphs from fonts loaded with `ly:system-font-load`; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings `fetaMusic` and `fetaBraces`, respectively.

`ly:font-glyph-name-to-charcode` *font name* [Function]
Return the character code for glyph *name* in *font*.

Note that this command can only be used to access glyphs from fonts loaded with `ly:system-font-load`; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings `fetaMusic` and `fetaBraces`, respectively.

`ly:font-glyph-name-to-index` *font name* [Function]
Return the index for *name* in *font*.

Note that this command can only be used to access glyphs from fonts loaded with `ly:system-font-load`; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings `fetaMusic` and `fetaBraces`, respectively.

`ly:font-index-to-charcode` *font index* [Function]
Return the character code for *index* in *font*.

Note that this command can only be used to access glyphs from fonts loaded with `ly:system-font-load`; currently, this means either the Emmentaler or Emmentaler-Brace fonts, corresponding to the font encodings `fetaMusic` and `fetaBraces`, respectively.

`ly:font-magnification` *font* [Function]
Given the font metric *font*, return the magnification, relative to the current `output-scale`.

`ly:font-metric?` *x* [Function]
Is *x* a smob of class `Font_metric`?

`ly:font-name` *font* [Function]
Given the font metric *font*, return the corresponding name.

`font-name-split` *font-name* [Function]
Return `(`*`font-name`* `.` *`design-size`*`)` from *font-name* string or `#f`.

`ly:font-sub-fonts` *font* [Function]
Given the font metric *font* of an OpenType font, return the names of the subfonts within *font*.

`for-some-music` *stop? music* [Function]
Walk through *music*, process all elements calling *stop?* and only recurse if this returns `#f`.

`ly:format` *str rest* [Function]
LilyPond specific format function, supporting `~a` and `~[0-9]f`. Basic support for `~s` is also provided.

`ly:format-output` *context* [Function]
Given a global context in its final state, process it and return the `Music_output` object in its final state.

`fret->pitch` *fret* [Function]
Calculate a pitch given *fret* for the harmonic.

`fret-parse-terse-definition-string` *props definition-string* [Function]
Parse a fret diagram string that uses terse syntax; return a pair containing *props*, modified to include the string-count determined by *definition-string*, and a fret indication list with the appropriate values.

`function-chain` *arg function-list* [Function]
Apply a list of functions in *function-list* to *arg*. Each element of *function-list* is structured `(cons function '(arg2 arg3 ...))`. If function takes arguments besides *arg*, they are provided in *function-list*. Example:

```
(function-chain 1 `((,+ 1) (,- 2) (,+ 3) (,/)))
   ⇒ 1/3
```

`generate-crop-stencil` *paper-book* [Function]
Returns a stencil for the cropped output of the given Paper_book

`generate-preview-stencil` *paper-book* [Function]
Returns a stencil for a preview of given Paper_book

`ly:generic-bound-extent` *grob common* [Function]
Determine the extent of *grob* relative to *common* along the x axis, finding its extent as a bound when it a has `bound-alignment-interfaces` property list set and otherwise the full extent.

`ly:get-all-function-documentation` [Function]
Get a hash table with all LilyPond Scheme extension functions.

`ly:get-all-translators` [Function]
Return a list of all translator objects that may be instantiated.

`get-bound-note-heads` *spanner* [Function]
Take a spanner grob and return a pair containing all note heads of the initial starting and the final `NoteColumn`.

`ly:get-cff-offset` *font-file-name idx* [Function]
Get the offset of the 'CFF' table for *font-file-name*, returning it as an integer. The optional *idx* argument is useful for OpenType/CFF collections (OTC) only; it specifies the font index within the OTC. The default value of *idx* is 0.

`get-chord-shape` *shape-code tuning base-chord-shapes* [Function]
Return the chord shape associated with *shape-code* and *tuning* in the hash-table *base-chord-shapes*.

`ly:get-context-mods` *contextmod* [Function]
Returns the list of context modifications stored in *contextmod*.

`ly:get-font-format` *font-file-name idx* [Function]
Get the font format for *font-file-name*, returning it as a symbol. The optional *idx* argument is useful for TrueType Collections (TTC) and OpenType/CFF collections (OTC) only; it specifies the font index within the TTC/OTC. The default value of *idx* is 0.

`ly:get-option` *var* [Function]
Get a global option setting.

`get-postscript-bbox` *string* [Function]
Extract the bounding box from *string*, or return `#f` if not present.

`ly:get-spacing-spec` *from-scm to-scm* [Function]
Return the spacing spec going between the two given grobs, *from-scm* and *to-scm*.

`get-tweakable-music` *mus* [Function]
When tweaking music, return a list of music expressions where the tweaks should be applied. Relevant for music wrappers and event chords.

`ly:gettext` *original* [Function]
A Scheme wrapper function for `gettext` (to translate messages).

`ly:grob?` *x* [Function]
Is *x* a smob of class `Grob`?

`grob::all-objects` *grob* [Function]
Return a list of the names and contents of all properties having type `ly:grob?` or `ly:grob-array?` for all interfaces supported by grob *grob*.

`grob::compose-function` *func data* [Function]
Create a callback entity *func* to be stored in a grob property, based on the grob property data *data* (which can be plain data, a callback itself, or an unpure-pure container).

Function or unpure-pure container *func* accepts a grob and a value and returns another value. Depending on the type of *data*, *func* is used for building a grob callback or an unpure-pure container.

`grob::display-objects` *grob* [Function]
Display all objects stored in properties of grob *grob*.

`grob::name` *grob* [Function]
Return the name of the grob *grob* as a symbol.

`grob::offset-function` *func data . rest* [Function]
Create a callback entity *func* to be stored in a grob property, based on the grob property data *data* (which can be plain data, a callback itself, or an unpure-pure container).

Function *func* accepts a grob and returns a value that is added to the value resulting from *data*. Optional argument *plus* defaults to '`+`' but may be changed to allow for using a different underlying accumulation.

If *data* is `#f` or `'()`, it is not included in the sum.

`grob::rhythmic-location` *grob* [Function]
Return a pair consisting of the measure number and moment within the measure of grob *grob*.

`grob::unpure-Y-extent-from-stencil` *pure-function* [Function]
The unpure height will come from a stencil whereas the pure height will come from `pure-function`.

`grob::when` *grob* [Function]
Return the global timestep (a `Moment`) of grob *grob*.

`ly:grob-alist-chain` *grob global* [Function]
Get an alist chain for grob *grob*, with *global* as the global default. If unspecified, `font-defaults` from the layout block is taken.

`ly:grob-array?` *x* [Function]
Is *x* a smob of class `Grob_array`?

`ly:grob-array->list` *grob-arr* [Function]
Return the elements of *grob-arr* as a Scheme list.

`ly:grob-array-length` *grob-arr* [Function]
Return the length of *grob-arr*.

`ly:grob-array-ref` *grob-arr index* [Function]
Retrieve the *index*th element of *grob-arr*.

`ly:grob-basic-properties` *grob* [Function]
Get the immutable properties of *grob*.

`ly:grob-chain-callback` *grob proc sym* [Function]
Find the callback that is stored as property *sym* of grob *grob* and chain *proc* to the head of this, meaning that it is called using *grob* and the previous callback's result.

`ly:grob-common-refpoint` *grob other axis* [Function]
Find the common refpoint of *grob* and *other* for *axis*.

`ly:grob-common-refpoint-of-array` *grob others axis* [Function]
Find the common refpoint of *grob* and *others* (a grob-array) for *axis*.

`ly:grob-default-font` *grob* [Function]
Return the default font for grob *grob*.

`ly:grob-extent` *grob refp axis* [Function]
Get the extent in *axis* direction of *grob* relative to the grob *refp*.

`ly:grob-get-vertical-axis-group-index` *grob* [Function]
Get the index of the vertical axis group the grob *grob* belongs to; return `-1` if none is found.

`ly:grob-interfaces` *grob* [Function]
Return the interfaces list of grob *grob*.

`ly:grob-layout` *grob* [Function]
Get `\layout` definition from grob *grob*.

`ly:grob-list->grob-array` *grob-list* [Function]
Convert a Scheme list of grobs to a grob array.

`ly:grob-object` *grob sym val* [Function]
Return the value of a pointer in grob *grob* of property *sym*. When *sym* is undefined in *grob*, it returns *val* if specified or `'()` (end-of-list) otherwise. The kind of properties this taps into differs from regular properties. It is used to store links between grobs, either grobs or grob arrays. For instance, a note head has a `stem` property, the stem grob it belongs to. Just after line breaking, all those grobs are scanned and replaced by their relevant broken versions when applicable.

`ly:grob-original` *grob* [Function]
Return the unbroken original grob of *grob*.

`ly:grob-parent` *grob axis* [Function]
Get the parent of *grob*. *axis* is `0` for the x axis, `1` for the y axis.

`ly:grob-pq<?` *a b* [Function]
Compare two grob priority queue entries. This is an internal function.

`ly:grob-properties?` *x* [Function]
Is *x* a smob of class `Grob_properties`?

`ly:grob-property` *grob sym val* [Function]
Return the value for property *sym* of *grob*. If no value is found, return *val* or `'()` if *val* is not specified.

`ly:grob-property-data` *grob sym* [Function]
Return the value for property *sym* of *grob*, but do not process callbacks.

`ly:grob-pure-height` *grob refp beg end val* [Function]
Return the pure height of *grob* given refpoint *refp*. If no value is found, return *val* or `'()` if *val* is not specified.

`ly:grob-pure-property` *grob sym beg end val* [Function]
Return the pure value for property *sym* of *grob*. If no value is found, return *val* or `'()` if *val* is not specified.

`ly:grob-relative-coordinate` *grob refp axis* [Function]
Get the coordinate in *axis* direction of *grob* relative to the grob *refp*.

`ly:grob-robust-relative-extent` *grob refp axis* [Function]
Get the extent in *axis* direction of *grob* relative to the grob *refp*, or `(0,0)` if empty.

`ly:grob-script-priority-less` *a b* [Function]
Compare two grobs by script priority. For internal use.

`ly:grob-set-nested-property!` *grob symlist val* [Function]
Set nested property *symlist* in grob *grob* to value *val*.

`ly:grob-set-object!` *grob sym val* [Function]
Set *sym* in grob *grob* to value *val*.

`ly:grob-set-parent!` *grob axis parent-grob* [Function]
Set *parent-grob* as the parent of grob *grob* in axis *axis*.

`ly:grob-set-property!` *grob sym val* [Function]
Set *sym* in grob *grob* to value *val*.

`ly:grob-spanned-rank-interval` *grob* [Function]
Return a pair with the `rank` of the furthest left column and the `rank` of the furthest right column spanned by `grob`.

`ly:grob-staff-position` *sg* [Function]
Return the y position of *sg* relative to the staff.

`ly:grob-suicide!` *grob* [Function]
Kill *grob*.

`ly:grob-system` *grob* [Function]
Return the system grob of *grob*.

`grob-transformer` *property func* [Function]
Create an override value good for applying *func* to either pure or unpure values. *func* is called with the respective grob as first argument and the default value (after resolving all callbacks) as the second.

`ly:grob-translate-axis!` *grob d a* [Function]
Translate *grob* on axis *a* over distance *d*.

`ly:grob-vertical<?` *a b* [Function]
Does *a* lie above *b* on the page?

`ly:gulp-file` *name size* [Function]
Read *size* characters from the file *name*, and return its contents in a string. If *size* is undefined, the entire file is read. The file is looked up using the search path.

`ly:gulp-file-utf8` *name size* [Function]
Read *size* characters from the file *name*, and return its contents in a string decoded from UTF-8. If *size* is undefined, the entire file is read. The file is looked up using the search path.

`ly:has-glyph-names?` *font-file-name idx* [Function]
Does the font for *font-file-name* have glyph names? The optional *idx* argument is useful for TrueType Collections (TTC) and OpenType/CFF collections (OTC) only; it specifies the font index within the TTC/OTC. The default value of *idx* is 0.

`ly:hash-table-keys` *tab* [Function]
Return a list of keys in *tab*.

`hook-stencil` *x y staff-space thick blot grob* [Function]
Return a hook stencil where *x* determines the horizontal position and *y* determines the basic vertical position. The final stencil is adjusted vertically using *staff-space*, which is `StaffSymbol`'s staff space, and uses *blot*, which is the current `'blot-diameter`. The stencil's thickness is usually taken from *grob* `'details`, *thick* serves as a fallback value.

`ly:in-event-class?` *ev cl* [Function]
Does event *ev* belong to event class *cl*?

`ly:inch` *num* [Function]
num inches.

`ly:input-both-locations` *sip* [Function]
Return input location in *sip* as

```
(file-name first-line first-column last-line last-column)
```

`ly:input-file-line-char-column` *sip* [Function]
Return input location in *sip* as `(file-name line char column)`.

`ly:input-location?` *x* [Function]
Is *x* a smob of class `Input`?

`ly:input-message` *sip msg rest* [Function]
Print *msg* as a GNU compliant error message, pointing to the location in *sip*. *msg* is interpreted similar to `format`'s argument, using *rest*.

`ly:input-warning` *sip msg rest* [Function]
Print *msg* as a GNU compliant warning message, pointing to the location in *sip*. *msg* is interpreted similar to `format`'s argument, using *rest*.

`ly:interpret-music-expression` *mus ctx* [Function]
Interpret the music expression *mus* in the global context *ctx*. The context is returned in its final state.

`interval-center` *x* [Function]
Center the number pair *x*, if an interval.

`interval-index` *interval dir* [Function]
Interpolate *interval* between between left (*dir*=`-1`) and right (*dir*=`+1`).

`interval-length` *x* [Function]
Length of the number pair *x*, if an interval.

`ly:intlog2` *d* [Function]
The 2-logarithm of 1/*d*.

`invalidate-alterations` *context* [Function]
Invalidate alterations in *context*.

Elements of `'localAlterations` corresponding to local alterations of the key signature have the form `'((octave . notename) . (alter barnum . end-mom))`. Replace them with a version where `alter` is set to `'clef` to force a repetition of accidentals.

Entries that conform with the current key signature are not invalidated.

`ly:item?` *g* [Function]
Is *g* an `Item` object?

`ly:item-break-dir` *it* [Function]
The break status direction of item *it*. `-1` means end of line, `0` unbroken, and `1` beginning of line.

`ly:item-get-column` *it* [Function]
Return the `PaperColumn` or `NonMusicalPaperColumn` associated with this `Item`.

`ly:iterator?` *x* [Function]
Is *x* a smob of class `Music_iterator`?

`layout-line-thickness` *grob* [Function]
Get the line thickness of the *grob*'s corresponding layout.

`layout-set-absolute-staff-size` *sz* [Function]
Set the absolute staff size inside of a `\layout{}` block. *sz* is in points.

`layout-set-staff-size` *sz* [Function]
Set the staff size inside of a `\layout{}` block. *sz* is in points.

`ly:length` *x y* [Function]
Calculate magnitude of given vector. With one argument, *x* is a number pair indicating the vector. With two arguments, *x* and *y* specify the respective coordinates.

`ly:lily-lexer?` *x* [Function]
Is *x* a smob of class `Lily_lexer`?

`ly:lily-parser?` *x* [Function]
Is *x* a smob of class `Lily_parser`?

`lilypond-main` *files* [Function]
Entry point for LilyPond.

`ly:line-interface::line` *grob startx starty endx endy* [Function]
Make a line using layout information from grob *grob*.

`list-insert-separator` *lst between* [Function]
Create new list, inserting *between* between elements of *lst*.

`list-join` *lst intermediate* [Function]
Put *intermediate* between all elements of *lst*.

`ly:listened-event-class?` *disp cl* [Function]
Does *disp* listen to any event type in the list *cl*?

`ly:listened-event-types` *disp* [Function]
Return a list of all event types that *disp* listens to.

`ly:listener?` *x* [Function]
Is *x* a smob of class `Listener`?

`lookup-markup-command` *code* [Function]
Return (*`function . signature`*) for a markup command *code*, or return `#f`.

`lyric-text::print` *grob* [Function]
Allow interpretation of tildes as lyric tieing marks.

`ly:make-book` *paper header scores* [Function]
Make a `\book` of *paper* and *header* (which may be `#f` as well) containing `\score`s.

`ly:make-book-part` *scores* [Function]
Make a `\bookpart` containing `\score`s.

`make-bow-stencil` *start stop thickness angularity bow-height orientation* [Function]
Create a bow stencil. It starts at point *start*, ends at point *stop*. *thickness* is the thickness of the bow. The higher the value of number *angularity*, the more angular the shape of the bow. *bow-height* determines the height of the bow. *orientation* determines whether the bow is concave or convex. Both variables are supplied to support independent usage.

Done by calculating a horizontal unit bow first, then moving all control points to the correct positions. Limitation: s-curves are currently not supported.

`make-c-time-signature-markup` *fraction* [Function]
Make markup for the 'C' time signature style.

`make-circle-stencil` *radius thickness fill* [Function]
Make a circle of radius *radius* and thickness *thickness*.

`make-clef-set` *clef-name* [Function]
Generate the clef setting commands for a clef with name *clef-name*.

`make-connected-line` *points grob* [Function]
Take a list of points, *points*. Return a line connecting *points*, using `ly:line-interface::line` and getting layout information from *grob*.

`make-connected-path-stencil` *pointlist thickness x-scale y-scale connect fill* [Function]
Make a connected path described by the list *pointlist*, beginning at point (0, 0), with thickness *thickness*, and scaled by *x-scale* in the x direction and *y-scale* in the y direction. *connect* and *fill* are boolean arguments that specify whether the path should be connected or filled, respectively.

`ly:make-context-mod` *mod-list* [Function]
Create a context modification, optionally initialized via the list of modifications *mod-list*.

`make-cue-clef-set` *clef-name* [Function]
Generate the clef setting commands for a cue clef with name *clef-name*.

`make-cue-clef-unset` [Function]
Reset the clef settings for a cue clef.

`ly:make-dispatcher` [Function]
Return a newly created dispatcher.

`ly:make-duration` *length dotcount num den* [Function]
Make a duration. *length* is the negative logarithm (base 2) of the duration: 1 is a half note, 2 is a quarter note, 3 is an eighth note, etc. The number of dots after the note is given by the optional argument *dotcount*.

The duration factor is optionally given by integers *num* and *den*, alternatively by a single rational number.

A duration is a musical duration, i.e., a length of time described by a power of two (whole, half, quarter, etc.) and a number of augmentation dots.

`make-duration-of-length` *moment* [Function]
Make duration of the given `moment` length.

`make-ellipse-stencil` *x-radius y-radius thickness fill* [Function]
Make an ellipse of x radius *x-radius*, y radius `y-radius`, and thickness *thickness* with fill defined by `fill`.

`make-filled-box-stencil` *xext yext* [Function]
Make a filled box.

`ly:make-global-context` *output-def* [Function]
Set up a global interpretation context, using the output block *output-def*. The context is returned.

`ly:make-global-translator` *global* [Function]
Create a translator group and connect it to the global context *global*. The translator group is returned.

`make-glyph-time-signature-markup` *style fraction* [Function]
Make markup for a symbolic time signature. If the music font does not have a glyph for the requested style and fraction, issue a warning and make a numbered time signature instead.

`ly:make-grob-properties` *alist* [Function]
Package the given property list *alist* in a grob property container stored in a context property with the name of a grob.

`make-grob-property-override` *grob gprop val* [Function]
Make a `Music` expression that overrides *gprop* to *val* in *grob*. This is a `\temporary \override`, making it possible to `\revert` to any previous value afterwards.

`make-grob-property-revert` *grob gprop* [Function]
Revert the grob property *gprop* for *grob*.

`make-grob-property-set` *grob gprop val* [Function]
Make a `Music` expression that overrides a *gprop* to *val* in *grob*. Does a pop first, i.e., this is not a `\temporary \override`.

`make-harmonic` *mus* [Function]
Convert music variable *mus* to harmonics.

`make-line-stencil` *width startx starty endx endy* [Function]
Make a line stencil of given line width and set its extents accordingly.

`ly:make-listener` *callback* [Function]
This is a compatibility wrapper for creating a 'listener' for use with `ly:add-listener` from a *callback* taking a single argument. Since listeners are equivalent to callbacks, this is no longer needed.

`make-modal-inverter` *around to scale* [Function]
Wrapper function for `inverter-factory`.

`make-modal-transposer` *from to scale* [Function]
Wrapper function for `transposer-factory`.

`ly:make-moment` *m g gn gd* [Function]
Create a moment with rational main timing *m*, and optional grace timing *g*.

A *moment* is a point in musical time. It consists of a pair of rationals (*m*, *g*), where *m* is the timing for the main notes, and *g* the timing for grace notes. In absence of grace notes, *g* is zero.

For compatibility reasons, it is possible to write two numbers specifying numerator and denominator instead of the rationals. These forms cannot be mixed, and the two-argument form is disambiguated by the sign of the second argument: if it is positive, it can only be a denominator and not a grace timing.

`ly:make-music` *props* [Function]
Make a C++ `Music` object and initialize it with *props*.

This function is for internal use and is only called by `make-music`, which is the preferred interface for creating music objects.

`make-music` *name . music-properties* [Function]
Create a music object of given name, and set its properties according to `music-properties`, a list of alternating property symbols and values. Example:

```
(make-music 'OverrideProperty
            'symbol 'Stem
            'grob-property 'thickness
            'grob-value (* 2 1.5))
```

Instead of a successive symbol and value, an entry in the list may also be an alist or a music object in which case its elements, respectively its *mutable* property list (properties not inherent to the type of the music object), are taken.

The argument list will be interpreted left to right, so later entries override earlier ones.

`ly:make-music-function` *signature func* [Function]
Make a function to process music, to be used for the parser. *func* is the function, and *signature* describes its arguments. *signature*'s cdr is a list containing either `ly:music?` predicates or other type predicates. Its car is the syntax function to call.

`ly:make-music-relative!` *music pitch* [Function]
Make *music* relative to *pitch*, return final pitch.

`ly:make-output-def` [Function]
Make an output definition.

`make-oval-stencil` *x-radius y-radius thickness fill* [Function]
Make an oval from two Bézier curves, of x radius *x-radius*, y radius *y-radius*, and thickness *thickness* with fill defined by *fill*.

`ly:make-page-label-marker` *label* [Function]
Return page marker with label *label*.

`ly:make-page-permission-marker` *symbol permission* [Function]
Return page marker with page breaking and turning permissions.

`ly:make-pango-description-string` *chain size* [Function]
Make a `PangoFontDescription` string for the property alist *chain* at size *size*.

`ly:make-paper-outputter` *port alist default-callback* [Function]
Create an outputter dumping to *port*. *alist* should map symbols to procedures. See file `output-ps.scm` for an example. If *default-callback* is given, it is called for unsupported expressions.

`make-part-combine-context-changes` *state-machine split-list* [Function]
Generate a sequence of part combiner context changes from a split list.

`make-part-combine-marks` *state-machine split-list* [Function]
Generate a sequence of part combiner events from a split list.

`make-partial-ellipse-stencil` *x-radius y-radius start-angle end-angle thick connect fill* [Function]
Create an elliptical arc. *x-radius* is the x radius of the arc. *y-radius* is the y radius of the arc. *start-angle* is the starting angle of the arc (in degrees). *end-angle* is the ending angle of the arc (in degrees). *thick* is the thickness of the line. *connect* is a boolean flag indicating whether the end should be connected to the start by a line. *fill* is a boolean flag indicating whether the shape should be filled.

`make-path-stencil` *path thickness x-scale y-scale fill* [Function]
Make a stencil based on the path described by the list *path*, with thickness *thickness*, and scaled by *x-scale* in the x direction and *y-scale* in the y direction. *fill* is a boolean argument that specifies whether the path should be filled. Valid path commands are

```
moveto rmoveto lineto rlineto curveto rcurveto closepath
```

and their standard SVG single-letter equivalents

```
M m L l C c Z z
```

`ly:make-pitch` *octave note alter* [Function]
Make a pitch. *octave* is specified by an integer, zero for the octave containing middle C. *note* is a number indexing the global default scale, with 0 corresponding to pitch C and 6 usually corresponding to pitch B. Optional *alter* is a rational number of 200-cent whole tones for alteration.

`ly:make-prob` *type init rest* [Function]
Create a `Prob` object.

`make-repeat` *name times main alts* [Function]
Create a repeat music expression, with all properties initialized properly.

`ly:make-rotation` *angle center* [Function]
Make a transform rotating by *angle* in degrees. If *center* is given as a pair of coordinates, it is the center of the rotation, otherwise the rotation is around (0, 0).

`ly:make-scale` *steps* [Function]
Create a scale. The argument is a vector of rational numbers, each of which represents the number of 200-cent tones of a pitch above the tonic.

`ly:make-scaling` *scale scaley* [Function]
Create a scaling transform from argument *scale* and optionally *scaley*. When both arguments are given, they must be real and give the scale in x and y direction. If only *scale* is given, it may also be complex to indicate a scaled rotation in the manner of complex number rotations, or a pair of reals for specifying different scales in x and y direction like with the first calling convention.

`ly:make-score` *music* [Function]
Return score with *music* encapsulated in it.

`make-semitone->pitch` *pitches* [Function]
Convert *pitches*, an unordered list of note values covering (after disregarding octaves) all absolute pitches in need of conversion, into a function converting semitone numbers (absolute pitch missing enharmonic information) back into note values.

For a key signature without accidentals

```
c cis d es e f fis g gis a bes b
```

might be a good choice, covering Bb major to A major and their parallel keys, and melodic/ harmonic C minor to A minor.

`ly:make-spring` *ideal min-dist* [Function]
Make a spring. *ideal* is the ideal distance of the spring, and *min-dist* is the minimum distance.

`ly:make-stencil` *expr xext yext* [Function]
Stencils are device independent output expressions. They carry two pieces of information:

1. A specification of how to print this object. This specification is processed by the output backends, for example `scm/output-ps.scm`.
2. The vertical and horizontal extents of the object, given as pairs. If an extent is unspecified (or if you use `empty-interval` as its value), it is taken to be empty.

`make-stencil-boxer` *thickness padding callback* [Function]
Return function that adds a box around the grob passed as argument.

`make-stencil-circler` *thickness padding callback* [Function]
Return function that adds a circle around the grob passed as argument.

`ly:make-stream-event` *cl proplist* [Function]
Create a stream event of class *cl* with the given mutable property list.

`make-tmpfile` *basename* [Function]
Return a temporary file (as a Scheme port). If *basename* is `#f`, a file in the directory given by the environment variable `$TMPDIR` is created.

`ly:make-transform` *xx yx xy yy x0 y0* [Function]
Create a transform. Without options, it is the identity transform. Given four arguments *xx*, *yx*, *xy*, and *yy*, it is a linear transform. Given six arguments (with *x0* and *y0* last), it is an affine transform.

Transforms can be called as functions on other transforms (concatening them) or on points given either as complex number or real number pair. See also `ly:make-rotation`, `ly:make-scaling`, and `ly:make-translation`.

`ly:make-translation` *x y* [Function]
Make a transform translating by *x* and *y*. If only *x* is given, it can also be a complex number or a pair of numbers indicating the offset to use.

`make-transparent-box-stencil` *xext yext* [Function]
Make a transparent box.

`ly:make-unpure-pure-container` *unpure pure* [Function]
Make an unpure-pure container. *unpure* should be an unpure expression, and *pure* should be a pure expression. If *pure* is omitted, the value of *unpure* will be used twice, except that a callback is given two extra arguments that are ignored for the sake of pure calculations.

`map-selected-alist-keys` *function keys alist* [Function]
Return *alist* with *function* applied to all of the values in list *keys*. Example:

```
(map-selected-alist-keys - '(a b) '((a . 1) (b . -2) (c . 3) (d . 4)))
   ⇒ ((a . -1) (b . 2) (c . 3) (d . 4))
```

`map-some-music` *map? music* [Function]
Walk through *music*, transform all elements calling *map?* and only recurse if this returns `#f`. `elements` or `articulations` that are not music expressions are discarded: this allows some amount of filtering.

`map-some-music` may overwrite the original *music*.

`markup-command-list?` *x* [Function]
Check whether *x* is a markup command list, i.e., a list composed of a markup list function and its arguments.

`markup-list?` *arg* [Function]
Return a true value if *x* is a list of markups or markup command lists.

`measure-counter::text` *grob* [Function]
A number for a measure count. Broken measures are numbered in parentheses. When the counter spans several measures (like with compressed multi-measure rests), it displays a measure range.

`mensural-flag` *grob* [Function]
Mensural flags: Create the flag stencil by loading the glyph from the font. Flags are always aligned with staff lines, so we need to check the end point of the stem: For stems ending on staff lines, use different flags than for notes between staff lines. The idea is that flags are always vertically aligned with the staff lines, regardless of whether the note head is on a staff line or between two staff lines. In other words, the inner end of a flag always touches a staff line.

`ly:message` *str rest* [Function]
A Scheme callable function to issue the message *str*. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`midi-program` *instrument* [Function]
Return the program of the instrument.

`ly:minimal-breaking` *pb* [Function]
Break (pages and lines) the `Paper_book` object *pb* without looking for optimal spacing: stack as many lines on a page before moving to the next one.

`ly:mm` *num* [Function]
num mm.

`mmrest-of-length` *mus* [Function]
Create a multi-measure rest of exactly the same length as *mus*.

`modern-straight-flag` *grob* [Function]
Modern straight flag style (for composers like Stockhausen, Boulez, etc.). The angles are 18 and 22 degrees and thus smaller than for the ancient style of Bach, etc.

`ly:module->alist` *mod* [Function]
Dump the contents of module *mod* as an alist.

`ly:module-copy` *dest src* [Function]
Copy all bindings from module *src* into *dest*.

`ly:modules-lookup` *modules sym def* [Function]
Look up *sym* in the list *modules*, returning the first occurrence. If not found, return *def* or `#f` if *def* isn't specified.

`ly:moment?` *x* [Function]
Is *x* a smob of class `Moment`?

`ly:moment<?` *a b* [Function]
Compare two moments.

`ly:moment-add` *a b* [Function]
Add two moments.

`ly:moment-div` *a b* [Function]
Divide two moments.

`ly:moment-grace` *mom* [Function]
Extract grace timing as a rational number from *mom*.

`ly:moment-grace-denominator` *mom* [Function]
Extract denominator from grace timing.

`ly:moment-grace-numerator` *mom* [Function]
Extract numerator from grace timing.

`ly:moment-main` *mom* [Function]
Extract main timing as a rational number from *mom*.

`ly:moment-main-denominator` *mom* [Function]
Extract denominator from main timing.

`ly:moment-main-numerator` *mom* [Function]
Extract numerator from main timing.

`ly:moment-mod` *a b* [Function]
Modulo of two moments.

`ly:moment-mul` *a b* [Function]
Multiply two moments.

`ly:moment-sub` *a b* [Function]
Subtract two moments.

`ly:music?` *obj* [Function]
Is *obj* a `Music` object?

`music->make-music` *obj* [Function]
Generate an expression that, once evaluated, may return an object equivalent to *obj*, that is, for a music expression, a `(make-music ...)` form.

`music-clone` *music . music-properties* [Function]
Clone *music* and set properties according to *music-properties*, a list of alternating property symbols and values:

```
(music-clone start-span 'span-direction STOP)
```

Only properties that are not overridden by *music-properties* are actually fully cloned.

`ly:music-compress` *mus scale* [Function]
Compress *mus* by *scale*.

`ly:music-deep-copy` *m origin* [Function]
Copy *m* and all sub expressions of *m*. *m* may be an arbitrary type; cons cells and music are copied recursively. If *origin* is given, it is used as the origin for one level of music by calling `ly:set-origin!` on the copy.

`ly:music-duration-compress` *mus fact* [Function]
Compress *mus* by factor *fact*, which is a `Moment`.

`ly:music-duration-length` *mus* [Function]
Extract the duration field from *mus* and return the length.

`music-filter` *pred? music* [Function]
Filter out music expressions that do not satisfy *pred?*.

`ly:music-function?` *x* [Function]
Is *x* a smob of class `Music_function`?

`ly:music-function-extract` *x* [Function]
Return the Scheme function inside *x*.

`ly:music-function-signature` *x* [Function]
Return the function signature inside *x*.

`music-is-of-type?` *mus type* [Function]
Does *mus* belong to the music class *type*?

`ly:music-length` *mus* [Function]
Get the length of music expression *mus* and return it as a `Moment` object.

`ly:music-list?` *lst* [Function]
Is *lst* a list of music objects?

`music-map` *function music* [Function]
Apply *function* to *music* and all of the music it contains.

First it recurses over the children, then the function is applied to *music*.

`ly:music-mutable-properties` *mus* [Function]
Return an alist containing the mutable properties of *mus*. The immutable properties are not available, since they are constant and initialized by the `make-music` function.

`ly:music-output?` *x* [Function]
Is *x* a smob of class `Music_output`?

`music-pitches` *music* [Function]
Return a list of all pitches from *music*.

`ly:music-property` *mus sym val* [Function]
Return the value for property *sym* of music expression *mus*. If no value is found, return *val* or `'()` if *val* is not specified.

`music-selective-filter` *descend? pred? music* [Function]
Recursively filter out music expressions that do not satisfy *pred?*, but refrain from filtering the subexpressions of music that does not satisfy *descend?*.

`music-selective-map` *descend? function music* [Function]
Apply *function* recursively to *music*, but refrain from mapping subexpressions of music that does not satisfy *descend?*.

`music-separator?` *m* [Function]
Is *m* a separator?

`ly:music-set-property!` *mus sym val* [Function]
Set property *sym* in music expression *mus* to *val*.

`ly:music-start` *mus* [Function]
Get the start of music expression *mus* and return it as a `Moment` object.

`ly:music-transpose` *m p* [Function]
Transpose *m* such that central C is mapped to *p*. Return *m*.

`music-type-predicate` *types* [Function]
Return a predicate function that can be used for checking music to have one of the types listed in *types*.

`neo-modern-accidental-rule` *context pitch barnum* [Function]
An accidental rule that typesets an accidental if it differs from the key signature *and* does not directly follow a note on the same staff line. This rule should not be used alone because it does neither look at bar lines nor different accidentals at the same note name.

`no-flag` *grob* [Function]
No flag: Simply return empty stencil.

`normal-flag` *grob* [Function]
Create a default flag.

`ly:note-column-accidentals` *note-column* [Function]
Return the `AccidentalPlacement` grob from *note-column* if any, or `SCM_EOL` otherwise.

`ly:note-column-dot-column` *note-column* [Function]
Return the `DotColumn` grob from *note-column* if any, or `SCM_EOL` otherwise.

`ly:note-head::stem-attachment` *font-metric glyph-name direction* [Function]
Get attachment in *font-metric* for attaching a stem to notehead *glyph-name* in the direction *direction* (default `UP`).

`note-name->markup` *pitch lowercase?* [Function]
Return pitch markup for *pitch*, including accidentals printed as glyphs. If *lowercase?* is set to false, the note names are capitalized.

`note-name->string` *pitch . language* [Function]
Return pitch string for *pitch*, without accidentals or octaves. Current input language is used for pitch names, except if an other *language* is specified.

`note-to-cluster` *music* [Function]
Replace `NoteEvents` by `ClusterNoteEvents`.

`ly:number->string` *s* [Function]
Convert *s* to a string without generating many decimals.

`number-format` *number-type num . custom-format* [Function]
Print *num* according to the requested *number-type*. Choices include `roman-lower` (the default), `roman-upper`, `arabic`, and `custom`. For `custom`, *custom-format* must be present; it gets applied to *num*.

`offset-fret` *fret-offset diagram-definition* [Function]
Add *fret-offset* to each fret indication in *diagram-definition* and return the resulting verbose `fret-diagram-definition`.

`offsetter` *property offsets* [Function]
Apply *offsets* to the default values of *property* of *grob*. Offsets are restricted to immutable properties and values of type `number`, `number-pair`, or `number-pair-list`.

`old-straight-flag` *grob* [Function]
Old straight flag style (for composers like Bach). The angles of the flags are both 45 degrees.

`ly:one-line-auto-height-breaking` *pb* [Function]
Put each score on a single line, and put each line on its own page. Modify the `paper-width` setting so that every page is wider than the widest line. Modify the `paper-height` setting to fit the height of the tallest line.

`ly:one-line-breaking` *pb* [Function]
Put each score on a single line, and put each line on its own page. Modify the `paper-width` setting so that every page is wider than the widest line.

`ly:one-page-breaking` *pb* [Function]
Put each score on a single page. The `paper-height` settings are modified so each score fits on one page, and the height of the page matches the height of the full score.

`ly:optimal-breaking` *pb* [Function]
Optimally break (pages and lines) the `Paper_book` object *pb* to minimize badness for both vertical and horizontal spacing.

`ly:option-usage` *port* [Function]
Print `ly:set-option` usage. Optional *port* argument for the destination defaults to current output port.

`ly:otf->cff` *otf-file-name idx* [Function]
Convert the contents of an OTF file to a CFF file, returning it as a string. The optional *idx* argument is useful for OpenType/CFF collections (OTC) only; it specifies the font index within the OTC. The default value of *idx* is 0.

`ly:otf-font?` *font* [Function]
Is *font* an OpenType font?

`ly:otf-font-glyph-info` *font glyph* [Function]
Given the font metric *font* of an OpenType font, return the information about named glyph *glyph* (a string).

`ly:otf-font-table-data` *font tag* [Function]
Extract a table *tag* from *font*. Return empty string for non-existent *tag*.

`ly:otf-glyph-count` *font* [Function]
Return the number of glyphs in *font*.

`ly:otf-glyph-list` *font* [Function]
Return a list of glyph names for *font*.

`ly:output-def?` *x* [Function]
Is *x* a smob of class `Output_def`?

`ly:output-def-clone` *def* [Function]
Clone output definition *def*.

`ly:output-def-lookup` *def sym val* [Function]
Return the value of *sym* in output definition *def* (e.g., `\paper`). If no value is found, return *val* or `'()` if *val* is undefined.

`ly:output-def-parent` *def* [Function]
Return the parent output definition of *def*.

`ly:output-def-scope` *def* [Function]
Return the variable scope inside *def*.

`ly:output-def-set-variable!` *def sym val* [Function]
Set an output definition *def* variable *sym* to *val*.

`ly:output-description` *output-def* [Function]
Return the description of translators in *output-def*.

`ly:output-find-context-def` *output-def context-name* [Function]
Return an alist of all context defs (matching *context-name* if given) in *output-def*.

`output-module?` *module* [Function]
Return `#t` if *module* belongs to an output module usually carrying context definitions (`\midi` or `\layout`).

`ly:outputter-close` *outputter* [Function]
Close port of *outputter*.

`ly:outputter-dump-stencil` *outputter stencil* [Function]
Dump stencil *expr* onto *outputter*.

`ly:outputter-dump-string` *outputter str* [Function]
Dump *str* onto *outputter*.

`ly:outputter-output-scheme` *outputter expr* [Function]
Output *expr* to the paper outputter.

`ly:outputter-port` *outputter* [Function]
Return output port for *outputter*.

`oval-stencil` *stencil thickness x-padding y-padding* [Function]
Add an oval around `stencil`, padded by the padding pair, producing a new stencil.

`override-head-style` *heads style* [Function]
Override style for *heads* to *style*.

`override-time-signature-setting` *time-signature setting* [Function]
Override the time signature settings for the context in *time-signature*, with the new setting alist *setting*.

`ly:page-marker?` *x* [Function]
Is *x* a smob of class `Page_marker`?

`ly:page-turn-breaking` *pb* [Function]
Optimally break (pages and lines) the `Paper_book` object *pb* such that page turns only happen in specified places, returning its pages.

`ly:pango-font?` *f* [Function]
Is *f* a Pango font?

`ly:pango-font-physical-fonts` *f* [Function]
Return alist of `(ps-name file-name font-index)` lists for Pango font *f*.

`pango-pf-file-name` *pango-pf* [Function]
Return the file name of the Pango physical font *pango-pf*.

`pango-pf-font-name` *pango-pf* [Function]
Return the font name of the Pango physical font *pango-pf*.

`pango-pf-fontindex` *pango-pf* [Function]
Return the font index of the Pango physical font *pango-pf*.

`ly:paper-book?` *x* [Function]
Is *x* a smob of class `Paper_book`?

`ly:paper-book-header` *pb* [Function]
Return the header definition (`\header`) in `Paper_book` object *pb*.

`ly:paper-book-pages` *pb* [Function]
Return pages in `Paper_book` object *pb*.

`ly:paper-book-paper` *pb* [Function]
Return the paper output definition (`\paper`) in `Paper_book` object *pb*.

`ly:paper-book-performances` *pb* [Function]
Return performances in `Paper_book` object *pb*.

`ly:paper-book-scopes` *pb* [Function]
Return scopes in `Paper_book` object *pb*.

`ly:paper-book-systems` *pb* [Function]
Return systems in `Paper_book` object *pb*.

`ly:paper-column::break-align-width` *col align-syms* [Function]
Determine the extent along the x axis of a grob used for break alignment organized by column *col*. The grob is specified by *align-syms*, which contains either a single `break-align-symbol` or a list of such symbols.

`ly:paper-column::print` [Function]
Optional stencil for `PaperColumn` or `NonMusicalPaperColumn`. Draws the *rank number* of each column, its moment in time, a blue arrow showing the ideal distance, and a red arrow showing the minimum distance between columns.

`ly:paper-fonts` *def* [Function]
Return a list containing the fonts from output definition *def* (e.g., `\paper`).

`ly:paper-get-font` *def chain* [Function]
Find a font metric in output definition *def* satisfying the font qualifiers in alist chain *chain*, and return it. (An alist chain is a list of alists, containing grob properties.)

`ly:paper-get-number` *def sym* [Function]
Return the value of variable *sym* in output definition *def* as a double.

`ly:paper-outputscale` *def* [Function]
Return the `output-scale` for output definition *def*.

`ly:paper-score-paper-systems` *paper-score* [Function]
Return vector of `paper_system` objects from *paper-score*.

`ly:paper-system?` *obj* [Function]
Is *obj* a C++ `Prob` object of type `paper-system`?

`ly:paper-system-minimum-distance` *sys1 sys2* [Function]
Measure the minimum distance between two paper system `Prob`s *sys1* and *sys2*, using their stored skylines if possible and falling back to their extents otherwise.

`parenthesize-stencil` *stencil half-thickness width angularity padding* [Function]
Add parentheses around *stencil*, returning a new stencil.

`ly:parse-file` *name* [Function]
Parse a single `.ly` file. Upon failure, throw `ly-file-failed` key.

`ly:parse-init` *name* [Function]
Parse the init file *name*.

`ly:parse-string-expression` *parser-smob ly-code filename line* [Function]
Parse the string *ly-code* with *parser-smob*. Return the contained music expression. *filename* and *line* are optional source indicators.

`parse-terse-string` *terse-definition* [Function]
Parse a `fret-diagram-terse` definition string *terse-definition* and return a marking list, which can be used with a fretboard grob.

`ly:parsed-undead-list!` [Function]
Return the list of objects that have been found alive but should have been dead, and clear that list.

`ly:parser-clear-error` *parser* [Function]
Clear error flag for *parser*, defaulting to current parser.

`ly:parser-clone` *closures location* [Function]
Return a clone of current parser. An association list of port positions to closures can be specified in *closures* in order to have `$` and `#` interpreted in their original lexical environment. If *location* is a valid location, it becomes the source of all music expressions inside.

`ly:parser-define!` *symbol val* [Function]
Bind *symbol* to *val* in current parser's module.

`ly:parser-error` *msg input* [Function]
Display an error message and make current parser fail. Without a current parser, trigger an ordinary error.

`ly:parser-has-error?` *parser* [Function]
Does *parser* (defaulting to current parser) have an error flag?

`ly:parser-include-string` *ly-code* [Function]
Include the string *ly-code* into the input stream for current parser. Can only be used in immediate Scheme expressions (`$` instead of `#`).

`ly:parser-lookup` *symbol* [Function]
Look up *symbol* in current parser's module. Return `'()` if not defined.

`ly:parser-output-name` *parser* [Function]
Return the base name of the output file. If *parser* is left off, use currently active parser.

`ly:parser-parse-string` *parser-smob ly-code* [Function]
Parse the string *ly-code* with *parser-smob*. Upon failure, throw `ly-file-failed` key.

`ly:parser-set-note-names` *names* [Function]
Replace current note names in parser. *names* is an alist of symbols. This only has effect if the current mode is notes.

`percussion?` *instrument* [Function]
Return `#t` if the instrument should use MIDI channel 9.

`ly:performance-headers` *performance* [Function]
Return the list of headers with the innermost first.

`ly:performance-write` *performance filename name* [Function]
Write *performance* to *filename* storing *name* as the name of the performance in the file metadata.

`ly:pitch?` *x* [Function]
Is *x* a smob of class `Pitch`?

`ly:pitch<?` *p1 p2* [Function]
Is *p1* lexicographically smaller than *p2*?

`ly:pitch-alteration` *pp* [Function]
Extract the alteration from pitch *pp*.

`ly:pitch-diff` *pitch root* [Function]
Return pitch *delta* such that *root* transposed by *delta* equals *pitch*.

`ly:pitch-negate` *p* [Function]
Negate pitch *p*.

`ly:pitch-notename` *pp* [Function]
Extract the note name from pitch *pp*.

`ly:pitch-octave` *pp* [Function]
Extract the octave from pitch *pp*.

`ly:pitch-quartertones` *pp* [Function]
Calculate the number of quarter tones of pitch *pp* from middle C.

`ly:pitch-semitones` *pp* [Function]
Calculate the number of semitones of pitch *pp* from middle C.

`ly:pitch-steps` *p* [Function]
Number of steps counted from middle C of the pitch *p*.

`ly:pitch-tones` *pp* [Function]
Calculate the number of tones of pitch *pp* from middle C as a rational number.

`ly:pitch-transpose` *p delta* [Function]
Transpose pitch *p* by the amount *delta*, where *delta* is relative to middle C.

`ly:pointer-group-interface::add-grob` *grob sym grob-element* [Function]
Add *grob-element* to *grob*'s *sym* grob array.

`polar->rectangular` *radius angle-in-degrees* [Function]
Return polar coordinates (*radius*, *angle-in-degrees*) as rectangular coordinates `(x-length . y-length)`.

`ly:position-on-line?` *sg spos* [Function]
Return whether *spos* is on a line of the staff associated with the grob *sg* (even on an extender line).

`ly:prob?` *x* [Function]
Is *x* a smob of class `Prob`?

`ly:prob-immutable-properties` *prob* [Function]
Retrieve an alist of immutable properties.

`ly:prob-mutable-properties` *prob* [Function]
Retrieve an alist of mutable properties.

`ly:prob-property` *prob sym val* [Function]
Return the value for property *sym* of Prob object *prob*. If no value is found, return *val* or `'()` if *val* is not specified.

`ly:prob-property?` *obj sym* [Function]
Is boolean prop *sym* of *obj* set?

`ly:prob-set-property!` *obj sym value* [Function]
Set property *sym* of *obj* to *value*.

`ly:prob-type?` *obj type* [Function]
Is *obj* the specified prob type?

`ly:programming-error` *str rest* [Function]
A Scheme callable function to issue the internal warning *str*. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`ly:progress` *str rest* [Function]
A Scheme callable function to print progress *str*. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`ly:property-lookup-stats` *sym* [Function]
Return hash table with a property access corresponding to *sym*. Choices are `prob`, `grob`, and `context`.

`ly:protects` [Function]
Return hash of protected objects.

`ly:pt` *num* [Function]
num printer points.

`ly:pure-call` *data grob start end rest* [Function]
Convert property *data* (unpure-pure container or procedure) to value in a pure context defined by *grob*, *start*, *end*, and possibly *rest* arguments.

`pure-chain-offset-callback` *grob start end prev-offset* [Function]
Sometimes, a chained offset callback is unpure and there is no way to write a pure function that estimates its behavior. In this case, we use a pure equivalent that will simply pass the previous calculated offset value.

`ly:randomize-rand-seed` [Function]
Randomize C random generator.

`ratio->fret` *ratio* [Function]
Calculate a fret number given *ratio* for the harmonic.

`ratio->pitch` *ratio* [Function]
Calculate a pitch given *ratio* for the harmonic.

`read-lily-expression` *chr port* [Function]
Read a lilypond music expression enclosed within `#{` and `#}` from *port* and return the corresponding Scheme music expression. '`$`' and '`#`' introduce immediate and normal Scheme forms.

`recording-group-emulate` *music odef* [Function]
Interpret *music* according to *odef*, but store all events in a chronological list, similar to the `Recording_group_engraver` in LilyPond version 2.8 and earlier.

`ly:register-stencil-expression` *symbol* [Function]
Add *symbol* as head of a stencil expression.

`ly:register-translator` *creator name description* [Function]
Register a translator *creator* (usually a descriptive alist or a function/closure returning one when given a context argument) with the given symbol *name* and the given *description* alist.

`ly:relative-group-extent` *elements common axis* [Function]
Determine the extent of *elements* relative to *common* in the *axis* direction.

`remove-grace-property` *context-name grob sym* [Function]
Remove all *sym* for *grob* in *context-name*.

`remove-whitespace` *strg* [Function]
Remove characters satisfying `char-whitespace?` from string *strg*.

`ly:rename-file` *oldname newname* [Function]
Rename *oldname* to *newname*. In contrast to Guile's `rename-file` function, this replaces the destination if it already exists. On Windows, fall back to copying the file contents if *newname* cannot be deleted.

`ly:reset-all-fonts` [Function]
Forget all about previously loaded fonts.

`retrieve-glyph-flag` *flag-style dir dir-modifier grob* [Function]
Load the correct flag glyph from the font.

`retrograde-music` *music* [Function]
Return *music* in retrograde (reversed) order.

`revert-fontSize` *func-name mag* [Function]
Used by `\magnifyMusic` and `\magnifyStaff`. Calculate the previous `fontSize` value (before scaling) by factoring out the magnification factor *mag* (if *func-name* is `'magnifyMusic`), or by factoring out the context property `magnifyStaffValue` (if *func-name* is `'magnifyStaff`). Revert the `fontSize` in the appropriate context accordingly.

With `\magnifyMusic`, the scaling is reverted after the music block it operates on. `\magnifyStaff` does not operate on a music block, so the scaling from a previous call (if there is one) is reverted before the new scaling takes effect.

`revert-head-style` *heads* [Function]
Revert style for *heads*.

`revert-props` *func-name mag props* [Function]
Used by `\magnifyMusic` and `\magnifyStaff`. Revert each prop in *props* in the appropriate context. *func-name* is either `'magnifyMusic` or `'magnifyStaff`. The *props* list is formatted like:

```
'((Stem thickness)
  (Slur line-thickness)
  ...)
```

`ly:round-filled-box` *xext yext blot* [Function]
Make a `Stencil` object that prints a black box of dimensions *xext*, *yext* and roundness *blot*.

`ly:round-polygon` *points blot extroversion filled-scm* [Function]
Make a `Stencil` object that prints a black polygon with corners at the points defined by *points* (list of coordinate pairs) and roundness *blot*. Optional *extroversion* shifts the outline outward, with the default of 0 keeping the middle of the line just on the polygon.

`rounded-box-stencil` *stencil thickness padding blot* [Function]
Add a rounded box around *stencil*, producing a new stencil.

`ly:run-translator` *mus output-def* [Function]
Process *mus* according to *output-def*. An interpretation context is set up, and *mus* is interpreted with it. The context is returned in its final state.

Optionally, this routine takes an object key to to uniquely identify the score block containing it.

`scale-beam-thickness` *mag* [Function]
Used by `\magnifyMusic`. Scaling `Beam.beam-thickness` exactly to the *mag* value will not work. This uses two reference values for `beam-thickness` to determine an acceptable value when scaling, then does the equivalent of a `\temporary \override` with the new value.

`scale-fontSize` *func-name mag* [Function]
Used by `\magnifyMusic` and `\magnifyStaff`. Look up the current `fontSize` in the appropriate context and scale it by the magnification factor *mag*. *func-name* is either `'magnifyMusic` or `'magnifyStaff`.

`scale-layout` *paper scale* [Function]
Return a clone of *paper*, scaled by the given scale factor.

`scale-props` *func-name mag allowed-to-shrink? props* [Function]
Used by `\magnifyMusic` and `\magnifyStaff`. For each prop in *props*, find the current value of the requested prop, scale it by the magnification factor *mag*, and do the equivalent of a `\temporary \override` with the new value in the appropriate context. If *allowed-to-shrink?* is `#f`, don't let the new value be less than the current value. *func-name* is either `'magnifyMusic` or `'magnifyStaff`. The *props* list is formatted like:

```
'((Stem thickness)
  (Slur line-thickness)
  ...)
```

`ly:score?` *x* [Function]
Is *x* a smob of class `Score`?

`ly:score-add-output-def!` *score def* [Function]
Add an output definition *def* to *score*.

`ly:score-embedded-format` *score layout* [Function]
Run *score* through *layout* (an output definition) scaled to correct `output-scale` already, returning a list of layout lines.

`ly:score-error?` *score* [Function]
Was there an error in the score?

`ly:score-header` *score* [Function]
Return score header.

`ly:score-music` *score* [Function]
Return score music.

`ly:score-output-defs` *score* [Function]
All output definitions in a score.

`ly:score-set-header!` *score module* [Function]
Set the score header.

`scorify-music` *music* [Function]
Preprocess *music*.

`seconds->moment` *s context* [Function]
Return a moment equivalent to *s* seconds at the current tempo.

`select-head-glyph` *style log* [Function]
Select a note head glyph string based on note head style *style* and duration log *log*.

`self-alignment-interface::self-aligned-on-breakable` *grob* [Function]
Return the `X-offset` that places *grob* according to its `self-alignment-X` over the reference point defined by the `break-align-anchor-alignment` of a `break-aligned` item such as a `Clef`.

`ly:separation-item::print` [Function]
Optional stencil for `PaperColumn` or `NonMusicalPaperColumn`. This function draws `horizontal-skylines` of each `PaperColumn`, showing the shapes used to determine the minimum distances between `PaperColumns` at the note spacing step, before staves have been spaced (vertically) on the page.

`sequential-music-to-chord-exceptions` *seq . rest* [Function]
Transform sequential music *seq* of type

```
<<c d e>>-\markup{ foobar }
```

to (`cons` *`cde-pitches foobar-markup`*), or to (`cons` *`de-pitches foobar-markup`*) if *omit-root* is given and non-false.

`set-accidental-style` *style . rest* [Function]
Set accidental style to *style*. Optionally take a context argument, e.g., `'Staff` or `'Voice`. The context defaults to `Staff`, except for piano styles, which use `GrandStaff` as a context.

`ly:set-color-names` *alist* [Function]
Define named colors for `ly:stencil-in-color`. *alist* has the entries of the format (*`name . color`*), where *color* is a list of length 3 (RGB) or 4 (RGB+alpha).

`ly:set-default-scale` *scale* [Function]
Set the global default scale. This determines the tuning of pitches with no accidentals or key signatures. The first pitch is C. Alterations are calculated relative to this scale. The number of pitches in this scale determines the number of scale steps that make up an octave. Usually the 7-note major scale.

`set-global-staff-size` *sz* [Function]
Set the default staff size, where *sz* is thought to be in points.

`ly:set-grob-creation-callback` *cb* [Function]
Specify a procedure that gets called every time a new grob is created. The callback receives as arguments the grob that was created, the name of the C++ source file that caused the grob to be created, and the corresponding line number in the C++ source file. Call with `#f` as argument to unset the callback.

`ly:set-grob-modification-callback` *cb* [Function]
Specify a procedure that gets called every time LilyPond modifies a grob property. The callback receives as arguments the grob that is being modified, the name of the C++ file in which the modification was requested, the line number in the C++ file in which the modification was requested, the name of the function in which the modification was requested, the property to be changed, and the new value for the property. Call with `#f` as argument to unset the callback.

`ly:set-middle-C!` *context* [Function]
Set the `middleCPosition` variable in *context* based on the variables `middleCClefPosition` and `middleCOffset`.

`set-mus-properties!` *m alist* [Function]
Set all of *alist* as properties of *m*.

`ly:set-option` *var val* [Function]
Set a program option.

`ly:set-origin!` *m origin* [Function]
Set the origin given in *origin* to *m*. *m* is typically a music expression or a list of music. List structures are searched recursively, but recursion stops at the changed music expressions themselves.

origin is generally of type `ly:input-location?`, defaulting to `(*location*)`. Other valid values for `origin` are a music expression which is then used as the source of location information, or `#f` or `'()` in which case no action is performed. The return value is *m* itself.

`ly:set-property-cache-callback` *cb* [Function]
Specify a procedure that gets called whenever LilyPond calculates a callback function and caches the result. The callback receives as arguments the grob whose property it is, the name of the property, the name of the callback that calculated the property, and the new (cached) value of the property. Call with `#f` as argument to unset the callback.

`shift-one-duration-log` *music shift dot* [Function]
Add *shift* to `duration-log` of `'duration` in *music* and optionally *dot* to any note encountered. The number of dots in the shifted music may not be less than zero.

`shift-right-at-line-begin` *g* [Function]
Shift an item to the right, but only at the start of the line.

`skip->rest` *mus* [Function]
Replace *mus* by `RestEvent` of the same duration if it is a `SkipEvent`. Useful for extracting parts from crowded scores.

`skip-of-length` *mus* [Function]
Create a skip of exactly the same length as *mus*.

`skip-of-moment-span` *start-moment end-moment* [Function]
Make skip music fitting between *start-moment* and *end-moment*. The grace part of *end-moment* matters only if *start-moment* and *end-mom* have the same main part.

`ly:skyline?` *x* [Function]
Is *x* a smob of class `Skyline`?

`ly:skyline-empty?` *sky* [Function]
Return whether skyline *sky* is empty.

`ly:skyline-pair?` *x* [Function]
Is *x* a smob of class `Skyline_pair`?

`ly:smob-protects` [Function]
Return LilyPond's internal smob protection list.

`ly:solve-spring-rod-problem` *springs rods length ragged* [Function]
Solve a spring and rod problem for *count* objects that are connected by *count*-1 *springs*, and an arbitrary number of *rods*. *count* is implicitly given by *springs* and *rods*. The *springs* argument has the format `(ideal, inverse_hook)` and *rods* is of the form `(idx1, idx2, distance)`.

length is a number, *ragged* a boolean.

The function returns a list containing the force (positive for stretching, negative for compressing and `#f` for non-satisfied constraints) followed by *spring-count*+1 positions of the objects.

`ly:source-file?` *x* [Function]
Is *x* a smob of class `Source_file`?

`ly:source-files` *parser-smob* [Function]
Return a list of input files that have been opened up to here, including the files that have been closed already. A parser, *parser-smob*, may optionally be specified.

`ly:span-bar::before-line-breaking` *grob* [Function]
A dummy callback that kills the `Grob` *grob* if it contains no elements.

`ly:span-bar::calc-glyph-name` *grob* [Function]
Return the `'glyph-name` of the corresponding `BarLine` grob. The corresponding `SpanBar` glyph is computed within `span-bar::compound-bar-line`.

`span-bar::compound-bar-line` *grob bar-glyph extent* [Function]
Build the stencil of the span bar.

`ly:span-bar::print` *grob* [Function]
The print routine for span bars.

`ly:span-bar::width` *grob* [Function]
Compute the width of the `SpanBar` stencil.

`Span_stem_engraver` *ctx* [Function]
Connect cross-staff stems to the stems above in the system.

`ly:spanner?` *g* [Function]
Is *g* a spanner object?

`ly:spanner-bound` *spanner dir* [Function]
Get one of the bounds of *spanner*. *dir* is `-1` for left, and `1` for right.

`ly:spanner-broken-into` *spanner* [Function]
Return broken-into list for *spanner*.

`ly:spanner-set-bound!` *spanner dir item* [Function]
Set grob *item* as bound in direction *dir* for *spanner*.

`ly:spawn` *command rest* [Function]
Simple Scheme interface to the GLib function `g_spawn_sync`. If an error occurs, format it with `format` and *rest*.

`split-list-by-separator` *lst pred* [Function]
Split *lst* at each element that satisfies *pred*, and return the parts (with the separators removed) as a list of lists. Example:

```
(split-list-by-separator '(a 0 b c 1 d) number?)
   ⇒ ((a) (b c) (d))
```

`ly:spring?` *x* [Function]
Is *x* a smob of class `Spring`?

`ly:spring-set-inverse-compress-strength!` *spring strength* [Function]
Set the inverse compress *strength* of *spring*.

`ly:spring-set-inverse-stretch-strength!` *spring strength* [Function]
Set the inverse stretch *strength* of *spring*.

`stack-lines` *dir padding baseline stils* [Function]
Stack stencils vertically with a baseline skip.

`stack-stencil-line` *space stencils* [Function]
Adjoin a list of *stencils* along the x axis, leaving *space* between the end of each stencil and the beginning of the following stencil. Stencils with empty y extent are not given *space* before them and don't avoid overlapping other stencils.

`stack-stencils` *axis dir padding stils* [Function]
Stack stencils *stils* in direction *axis*, *dir*, using *padding*.

`stack-stencils-padding-list` *axis dir paddings stils* [Function]
Stack stencils *stils* in direction *axis*, *dir*, using a list of *paddings*.

`ly:staff-symbol-line-thickness` *grob* [Function]
Return the current staff line thickness in the staff associated with *grob*, expressed as a multiple of the current staff space height.

`ly:staff-symbol-staff-radius` *grob* [Function]
Return the radius of the staff associated with *grob*.

`ly:staff-symbol-staff-space` *grob* [Function]
Return the current staff space height in the staff associated with *grob*, expressed as a multiple of the default height of a staff space in the traditional five-line staff.

`ly:stderr-redirect` *fd-or-file-name mode* [Function]
Redirect standard error output (stderr) to file descriptor *fd* if the first parameter is an integer, or to file *file-name*, opened with *mode*.

`ly:stencil?` *x* [Function]
Is *x* a smob of class `Stencil`?

`ly:stencil-add` *args* [Function]
Combine stencils. Takes any number of arguments.

`ly:stencil-aligned-to` *stil axis dir* [Function]
Align stencil *stil* using its own extents. *dir* is a number. `-1` and `1` are left and right, respectively. Other values are interpolated (so `0` means the center).

`ly:stencil-combine-at-edge` *first axis direction second padding* [Function]
Construct a stencil by putting *second* next to *first*. *axis* can be `0` (x axis) or `1` (y axis). *direction* can be `-1` (left or down) or `1` (right or up). The stencils are juxtaposed with *padding* as extra space. *first* and *second* may also be `'()` or `#f`.

`ly:stencil-empty?` *stil axis* [Function]
Return whether *stil* is empty. If an optional *axis* is supplied, the emptiness check is restricted to that axis.

`ly:stencil-expr` *stil* [Function]
Return the expression of stencil *stil*.

`ly:stencil-extent` *stil axis* [Function]
Return a pair of numbers signifying the extent of stencil *stil* in *axis* direction (`0` or `1` for x and y axis, respectively).

`ly:stencil-in-color` *stc r g b a* [Function]
Put stencil *stc* in a different color. Accepts either three values for *r*, *g*, *b* and an optional value for *a*, or a single CSS-like string.

`ly:stencil-outline` *stil outline* [Function]
Return a stencil with the stencil expression (inking) of stencil *stil* but with outline and dimensions from stencil *outline*.

`ly:stencil-rotate` *stil angle x y* [Function]
Return a stencil *stil* rotated by *angle* degrees around the relative offset (*x*, *y*). E.g., an offset of (-1, 1) rotates the stencil around the left upper corner.

`ly:stencil-rotate-absolute` *stil angle x y* [Function]
Return a stencil *stil* rotated by *angle* degrees around point (*x*, *y*), given in absolute coordinates.

`ly:stencil-scale` *stil x y* [Function]
Scale stencil *stil* using the horizontal and vertical scaling factors *x* and *y*. Negative values flip or mirror *stil* without changing its origin; this may result in collisions unless it is repositioned.

`ly:stencil-stack` *first axis direction second padding mindist* [Function]
Construct a stencil by stacking *second* next to *first*. *axis* can be `0` (x axis) or `1` (y axis). *direction* can be `-1` (left or down) or `1` (right or up). The stencils are juxtaposed with *padding* as extra space. *first* and *second* may also be `'()` or `#f`. As opposed to `ly:stencil-combine-at-edge`, metrics are suited for successively accumulating lines of stencils. Also, *second* stencil is drawn last.

If *mindist* is specified, reference points are placed apart at least by this distance. If either of the stencils is spacing, *padding* and *mindist* do not apply.

`ly:stencil-translate` *stil offset* [Function]
Return a copy of stencil *stil* but translated by *offset* (a pair of numbers).

`ly:stencil-translate-axis` *stil amount axis* [Function]
Return a copy of stencil *stil* but translated by *amount* in *axis* direction.

`stencil-whiteout` *stil . lambda*:G28* [Function]
White-out a stencil (i.e., add a white background around it).

style, *thickness* and *line-thickness* are optional arguments. If set, *style* determines the shape of the white background. Given `'outline` the white background is produced by `stencil-whiteout-outline`, given `'rounded-box` it is produced by `stencil-whiteout-box` with rounded corners, given other arguments (e.g., `'box`) or when unspecified it defaults to `stencil-whiteout-box` with square corners. If *thickness* is specified it determines how far, as a multiple of *line-thickness*, the white background extends past the extents of stencil *stil*. If *thickness* has not been specified, an appropriate default is chosen based on *style*.

`stencil-whiteout-box` *stil . lambda*:G26* [Function]
White-out a stencil by printing it on top of a white (or *color*) rectangle.

thickness is how far, as a multiple of `line-thickness`, the white outline extends past the extents of stencil *stil*.

`stencil-whiteout-outline` *stil . lambda*:G24* [Function]
White-out a stencil by surrounding it with white (or *color*) around its outline.

This function works by creating a series of white or *color* stencils radially offset from the original stencil with angles from 0 to 2*pi, at an increment of `angle-inc`, and with radii from `radial-inc` to *thickness*. *thickness* is how big the white outline is, as a multiple of `line-thickness`. *radial-increments* is how many copies of the white stencil we make on our way out to thickness. *angle-increments* is how many copies of the white stencil we make between 0 and 2*pi.

`straight-flag` *flag-thickness flag-spacing upflag-angle upflag-length downflag-angle downflag-length* [Function]
Create a stencil for a straight flag. *flag-thickness* and *flag-spacing* are given in staff spaces, *upflag-angle* and *downflag-angle* are given in degrees, and *upflag-length* and *downflag-length* are given in staff spaces.

All lengths are scaled according to the font size of the note.

`ly:stream-event?` *obj* [Function]
Is *obj* a `Stream_event` object?

`ly:string-percent-encode` *str* [Function]
Encode all characters in string *str* with hexadecimal percent escape sequences, with the following exceptions: characters '`-./_`' and characters in ranges `0-9`, `A-Z`, and `a-z`.

`ly:string-substitute` *a b s* [Function]
Replace string *a* by string *b* in string *s*.

`style-note-heads` *heads style music* [Function]
Set *style* for all *heads* in *music*. Works both inside of and outside of chord construct.

`symbol<?` *a b* [Function]
Return a comparator function that applies *key* to the two elements and compares the results using *cmp*. Especially useful for sorting.

`symbol-concatenate` *ame* [Function]
Like `string-concatenate`, but for symbols.

`symbol-key<?` *a b* [Function]
Return a comparator function that applies *key* to the two elements and compares the results using *cmp*. Especially useful for sorting.

`ly:system-font-load` *name* [Function]
Load the OpenType system font ***name*`.otf`**. Fonts loaded with this command must contain three additional SFNT font tables called `LILC`, `LILF`, and `LILY`, needed for typesetting musical elements. Currently, only the Emmentaler and the Emmentaler-Brace fonts fulfill these requirements.

Note that only `ly:font-get-glyph` and derived code (like `\lookup`) can access glyphs from the system fonts; text strings are handled exclusively via the Pango interface.

`tag-group-get` *tag* [Function]
Return the tag group (as a list of symbols) that the given *tag* symbol belongs to, `#f` if none.

`tags-keep-predicate` *tags* [Function]
Return a predicate that returns `#f` for any music that is to be removed by `\keepWithTag` on the given symbol or list of symbols *tags*.

`tags-remove-predicate` *tags* [Function]
Return a predicate that returns `#f` for any music that is to be removed by `\removeWithTag` on the given symbol or list of symbols *tags*.

`teaching-accidental-rule` *context pitch barnum* [Function]
An accidental rule that typesets a cautionary accidental if it is included in the key signature *and* does not directly follow a note on the same staff line.

`ly:text-interface::interpret-markup` [Function]
Convert a text markup into a stencil. Takes three arguments, *layout*, *props*, and *markup*.

layout is a `\layout` block; it may be obtained from a grob with `ly:grob-layout`. *props* is an alist chain, i.e., a list of alists. This is typically obtained with `(ly:grob-alist-chain grob (ly:output-def-lookup layout 'text-font-defaults))`. *markup* is the markup text to be processed.

`ly:time-signature::print` *grob* [Function]
Print routine for time signatures.

`ly:transform?` *x* [Function]
Is *x* a smob of class `Transform`?

`ly:transform->list` *transform* [Function]
Convert a transform matrix to a list of six values. Values are *xx*, *yx*, *xy*, *yy*, *x0*, *y0*.

`ly:translate-cpp-warning-scheme` *str* [Function]
Translate a string in C++ `printf` format and modify it to use it for Scheme formatting.

`ly:translator?` *x* [Function]
Is *x* a smob of class `Translator`?

`ly:translator-context` *trans* [Function]
Return the context of the translator object *trans*.

`ly:translator-description` *creator* [Function]
Return an alist of properties of translator definition *creator*.

`ly:translator-group?` *x* [Function]
Is *x* a smob of class `Translator_group`?

`ly:translator-name` *creator* [Function]
Return the type name of the translator definition *creator*. The name is a symbol.

`ly:transpose-key-alist` *l pit* [Function]
Make a new key alist of *l* transposed by pitch *pit*.

`ly:ttf->pfa` *ttf-file-name idx* [Function]
Convert the contents of a TrueType font file to PostScript Type 42 font, returning it as a string. The optional *idx* argument is useful for TrueType collections (TTC) only; it specifies the font index within the TTC. The default value of *idx* is 0.

`ly:ttf-ps-name` *ttf-file-name idx* [Function]
Extract the PostScript name from a TrueType font. The optional *idx* argument is useful for TrueType collections (TTC) only; it specifies the font index within the TTC. The default value of *idx* is 0.

`ly:type1->pfa` *type1-file-name* [Function]
Convert the contents of a Type 1 font in PFB format to PFA format. If the file is already in PFA format, pass it through.

`unfold-repeats` *types music* [Function]
Replace repeats of the types given by *types* with unfolded repeats. If *types* is an empty list, `repeated-music` is taken, unfolding all.

`unfold-repeats-fully` *music* [Function]
Unfold repeats and expand the resulting `unfolded-repeated-music`.

`uniq-list` *lst* [Function]
Remove doublets from list *lst* (i.e., make its elements unique), assuming that it is sorted. Uses `equal?` for comparisons.

`ly:unit` [Function]
Return the unit used for lengths as a string.

`unity-if-multimeasure` *context dur* [Function]
Given a context and a duration, return `1` if the duration is longer than the `measureLength` in that context, and `#f` otherwise. This supports historic use of `Completion_heads_engraver` to split `c1*3` into three whole notes.

`ly:unpure-call` *data grob rest* [Function]
Convert property *data* (unpure-pure container or procedure) to value in an unpure context defined by *grob* and possibly *rest* arguments.

`ly:unpure-pure-container?` *x* [Function]
Is *x* a smob of class `Unpure_pure_container`?

`ly:unpure-pure-container-pure-part` *pc* [Function]
Return the pure part of *pc*.

`ly:unpure-pure-container-unpure-part` *pc* [Function]
Return the unpure part of *pc*.

`ly:usage` [Function]
Print usage message.

`ly:verbose-output?` [Function]
Was verbose output requested, i.e., is the log level at least `DEBUG`?

`ly:version` [Function]
Return the current LilyPond version as a list, e.g., `(1 3 127 uu1)`.

`ly:version?` *op ver* [Function]
Use operator *op* to compare the currently executed LilyPond version with a given version *ver*, which is passed as a list of numbers.

`voicify-music` *m . lambda*:G79* [Function]
Recursively split chords that are separated with `\\`. Optional *id* can be a list of context ids to use. If numeric, they also indicate a voice type override. If *id* is just a single number, that's where numbering starts.

`volta-bracket::calc-hook-visibility` *bar-glyph* [Function]
Determine the visibility of the volta bracket end hook, returning `#t` if *no* hook should be drawn.

`ly:volta-bracket::calc-shorten-pair` *grob* [Function]
Calculate the `shorten-pair` values for an ideal placement of the volta brackets relative to the bar lines.

`volta-spec-music` *number-list music* [Function]
Add \volta *number-list* to *music*.

`ly:warning` *str rest* [Function]
A Scheme callable function to issue the warning *str*. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`ly:warning-located` *location str rest* [Function]
A Scheme callable function to issue the warning *str* at the specified location in an input file. The message is formatted with `format`; *rest* holds the formatting arguments (if any).

`ly:wide-char->utf-8` *wc* [Function]
Encode the Unicode codepoint *wc*, an integer, as UTF-8.

`write-me` *message x* [Function]
Return *x*. Display *message* and write *x*. Handy for debugging, possibly turned off.

Appendix A Indices

A.1 Concept index

(Index is nonexistent)

A.2 Function index

A

add-bar-glyph-print-procedure 634
add-grace-property 634
add-music-fonts 634
add-new-clef 634
add-simple-time-signature-style 634
add-stroke-glyph 634
add-stroke-straight 634
alist->hash-table 635
allow-volta-hook 635
alterations-in-key 635
angle-0-2pi 635
angle-0-360 635
arrow-stencil 635
arrow-stencil-maker 635

B

bar-line::calc-break-visibility 636
bar-line::calc-glyph-name 636
bar-line::calc-glyph-name-for-direction 636
bar-line::compound-bar-line 636
bar-line::draw-filled-box 636
bar-line::widen-bar-extent-on-span 636
base-length 636
beam-exceptions 636
beat-structure 636
bend-spanner::print 637
bend::arrow-head-stencil 636
bend::calc-bend-x-begin 636
bend::calc-bend-x-end 636
bend::target-cautionary 637
bend::text-string 637
book-first-page 637
box-grob-stencil 637
box-stencil 637
bracketify-stencil 638
break-alignable-interface::
  self-alignment-of-anchor 638
break-alignable-interface::
  self-alignment-opposite-of-anchor 638
break-alignment-list 638

C

calc-harmonic-pitch 638
car< 638
car<= 638
centered-stencil 638
centered-text-interface::print 638
change-pitches 638
check-context-path 639
check-grob-path 639
check-music-path 639
circle-stencil 639
clef-transposition-markup 639
collect-book-music-for-book 639
collect-bookpart-for-book 639
collect-music-aux 639
collect-music-for-book 639
constante-hairpin 640
construct-chord-elements 640
context-spec-music 641
copy-repeat-chord 641
count-list 641
create-glyph-flag 641
cross-staff-connect 641
cue-substitute 641
cyclic-base-value 642

D

default-flag 642
define-bar-line 642
define-event-class 642
define-fonts 642
define-tag-group 642
degrees->radians 642
descend-to-context 642
determine-split-list 642
determine-string-fret-finger 642
dir-basename 643
display-lily-music 643
display-music 643
display-scheme-music 643
dodecaphonic-no-repeat-rule 643
duration-dot-factor 643
duration-length 644
duration-line::calc 644
duration-line::print 644
duration-log-factor 644
duration-visual 644
duration-visual-length 644
dynamic-text-spanner::
  before-line-breaking 644

E

ellipse-stencil ... 644
eval-carefully ... 645
event-chord-notes ... 645
event-chord-pitches ... 645
event-chord-reduce ... 645
event-chord-wrap! ... 645
event-has-articulation? ... 645
expand-repeat-chords! ... 646
expand-repeat-notes! ... 646
extract-beam-exceptions ... 646
extract-music ... 646
extract-named-music ... 646
extract-typed-music ... 646

F

find-named-props ... 646
find-pitch-entry ... 646
finger-glide::print ... 646
first-assoc ... 647
first-member ... 647
flared-hairpin ... 647
flat-flag ... 647
flatten-list ... 647
flip-stencil ... 647
fold-some-music ... 647
font-name-split ... 648
for-some-music ... 648
fret->pitch ... 649
fret-parse-terse-definition-string ... 649
function-chain ... 649

G

generate-crop-stencil ... 649
generate-preview-stencil ... 649
get-bound-note-heads ... 649
get-chord-shape ... 649
get-postscript-bbox ... 650
get-tweakable-music ... 650
grob-transformer ... 653
grob::all-objects ... 650
grob::compose-function ... 650
grob::display-objects ... 650
grob::name ... 650
grob::offset-function ... 650
grob::rhythmic-location ... 650
grob::unpure-Y-extent-from-stencil ... 650
grob::when ... 650

H

hook-stencil ... 653

I

interval-center ... 654
interval-index ... 654
interval-length ... 654
invalidate-alterations ... 654

L

layout-line-thickness ... 654
layout-set-absolute-staff-size ... 654
layout-set-staff-size ... 654
lilypond-main ... 654
list-insert-separator ... 655
list-join ... 655
lookup-markup-command ... 655
ly:add-context-mod ... 634
ly:add-interface ... 634
ly:add-listener ... 634
ly:add-option ... 634
ly:all-grob-interfaces ... 635
ly:all-options ... 635
ly:all-output-backend-commands ... 635
ly:all-stencil-commands ... 635
ly:all-stencil-expressions ... 635
ly:angle ... 635
ly:assoc-get ... 635
ly:axis-group-interface::add-element ... 635
ly:bar-line::calc-anchor ... 635
ly:bar-line::print ... 636
ly:basic-progress ... 636
ly:book-add-bookpart! ... 637
ly:book-add-score! ... 637
ly:book-book-parts ... 637
ly:book-header ... 637
ly:book-paper ... 637
ly:book-process ... 637
ly:book-process-to-systems ... 637
ly:book-scores ... 637
ly:book-set-header! ... 637
ly:book? ... 637
ly:bp ... 637
ly:bracket ... 638
ly:broadcast ... 638
ly:cairo-output-stencil ... 638
ly:cairo-output-stencils ... 638
ly:camel-case->lisp-identifier ... 638
ly:chain-assoc-get ... 638
ly:check-expected-warnings ... 639
ly:cm ... 639
ly:command-line-code ... 639
ly:command-line-options ... 639
ly:connect-dispatchers ... 639
ly:context-current-moment ... 640
ly:context-def-lookup ... 640
ly:context-def-modify ... 640
ly:context-def? ... 640
ly:context-event-source ... 640
ly:context-events-below ... 640
ly:context-find ... 640
ly:context-grob-definition ... 640
ly:context-id ... 641
ly:context-matched-pop-property ... 641
ly:context-mod-apply! ... 641
ly:context-mod? ... 641
ly:context-name ... 641
ly:context-output-def ... 641
ly:context-parent ... 641
ly:context-property ... 641
ly:context-property-where-defined ... 641
ly:context-pushpop-property ... 641
ly:context-set-property! ... 641

ly:context-unset-property 641
ly:context? 640
ly:debug 642
ly:default-scale 642
ly:dimension? 643
ly:dir? 643
ly:directed 643
ly:disconnect-dispatchers 643
ly:dispatcher? 643
ly:duration->string 643
ly:duration-compress 643
ly:duration-dot-count 643
ly:duration-factor 643
ly:duration-length 644
ly:duration-log 644
ly:duration-scale 644
ly:duration<? 643
ly:duration? 643
ly:effective-prefix 644
ly:encode-string-for-pdf 644
ly:engraver-announce-end-grob 644
ly:engraver-make-grob 644
ly:engraver-make-item 645
ly:engraver-make-spanner 645
ly:engraver-make-sticky 645
ly:error 645
ly:event-deep-copy 645
ly:event-property 645
ly:event-set-property! 645
ly:event? 645
ly:expect-warning 646
ly:extract-subfont-from-collection 646
ly:find-file 646
ly:font-config-add-directory 647
ly:font-config-add-font 647
ly:font-config-display-fonts 647
ly:font-config-get-font-file 647
ly:font-design-size 648
ly:font-file-name 648
ly:font-get-glyph 648
ly:font-glyph-name-to-charcode 648
ly:font-glyph-name-to-index 648
ly:font-index-to-charcode 648
ly:font-magnification 648
ly:font-metric? 648
ly:font-name 648
ly:font-sub-fonts 648
ly:format 648
ly:format-output 649
ly:generic-bound-extent 649
ly:get-all-function-documentation 649
ly:get-all-translators 649
ly:get-cff-offset 649
ly:get-context-mods 649
ly:get-font-format 649
ly:get-option 650
ly:get-spacing-spec 650
ly:gettext 650
ly:grob-alist-chain 651
ly:grob-array->list 651
ly:grob-array-length 651
ly:grob-array-ref 651
ly:grob-array? 651
ly:grob-basic-properties 651
ly:grob-chain-callback 651
ly:grob-common-refpoint 651
ly:grob-common-refpoint-of-array 651
ly:grob-default-font 651
ly:grob-extent 651
ly:grob-get-vertical-axis-group-index 651
ly:grob-interfaces 651
ly:grob-layout 651
ly:grob-list->grob-array 651
ly:grob-object 651
ly:grob-original 651
ly:grob-parent 652
ly:grob-pq<? 652
ly:grob-properties? 652
ly:grob-property 652
ly:grob-property-data 652
ly:grob-pure-height 652
ly:grob-pure-property 652
ly:grob-relative-coordinate 652
ly:grob-robust-relative-extent 652
ly:grob-script-priority-less 652
ly:grob-set-nested-property! 652
ly:grob-set-object! 652
ly:grob-set-parent! 652
ly:grob-set-property! 652
ly:grob-spanned-rank-interval 652
ly:grob-staff-position 652
ly:grob-suicide! 652
ly:grob-system 652
ly:grob-translate-axis! 653
ly:grob-vertical<? 653
ly:grob? 650
ly:gulp-file 653
ly:gulp-file-utf8 653
ly:has-glyph-names? 653
ly:hash-table-keys 653
ly:in-event-class? 653
ly:inch 653
ly:input-both-locations 653
ly:input-file-line-char-column 653
ly:input-location? 653
ly:input-message 653
ly:input-warning 653
ly:interpret-music-expression 654
ly:intlog2 654
ly:item-break-dir 654
ly:item-get-column 654
ly:item? 654
ly:iterator? 654
ly:length 654
ly:lily-lexer? 654
ly:lily-parser? 654
ly:line-interface::line 655
ly:listened-event-class? 655
ly:listened-event-types 655
ly:listener? 655
ly:make-book 655
ly:make-book-part 655
ly:make-context-mod 656
ly:make-dispatcher 656
ly:make-duration 656
ly:make-global-context 656
ly:make-global-translator 656
ly:make-grob-properties 656
ly:make-listener 657
ly:make-moment 657

ly:make-music ... 657
ly:make-music-function ... 657
ly:make-music-relative! ... 657
ly:make-output-def ... 657
ly:make-page-label-marker ... 658
ly:make-page-permission-marker ... 658
ly:make-pango-description-string ... 658
ly:make-paper-outputter ... 658
ly:make-pitch ... 658
ly:make-prob ... 658
ly:make-rotation ... 658
ly:make-scale ... 659
ly:make-scaling ... 659
ly:make-score ... 659
ly:make-spring ... 659
ly:make-stencil ... 659
ly:make-stream-event ... 659
ly:make-transform ... 659
ly:make-translation ... 660
ly:make-unpure-pure-container ... 660
ly:message ... 660
ly:minimal-breaking ... 660
ly:mm ... 660
ly:module->alist ... 661
ly:module-copy ... 661
ly:modules-lookup ... 661
ly:moment-add ... 661
ly:moment-div ... 661
ly:moment-grace ... 661
ly:moment-grace-denominator ... 661
ly:moment-grace-numerator ... 661
ly:moment-main ... 661
ly:moment-main-denominator ... 661
ly:moment-main-numerator ... 661
ly:moment-mod ... 661
ly:moment-mul ... 661
ly:moment-sub ... 661
ly:moment<? ... 661
ly:moment? ... 661
ly:music-compress ... 662
ly:music-deep-copy ... 662
ly:music-duration-compress ... 662
ly:music-duration-length ... 662
ly:music-function-extract ... 662
ly:music-function-signature ... 662
ly:music-function? ... 662
ly:music-length ... 662
ly:music-list? ... 662
ly:music-mutable-properties ... 662
ly:music-output? ... 662
ly:music-property ... 663
ly:music-set-property! ... 663
ly:music-start ... 663
ly:music-transpose ... 663
ly:music? ... 661
ly:note-column-accidentals ... 663
ly:note-column-dot-column ... 663
ly:note-head::stem-attachment ... 663
ly:number->string ... 664
ly:one-line-auto-height-breaking ... 664
ly:one-line-breaking ... 664
ly:one-page-breaking ... 664
ly:optimal-breaking ... 664
ly:option-usage ... 664
ly:otf->cff ... 664
ly:otf-font-glyph-info ... 664
ly:otf-font-table-data ... 665
ly:otf-font? ... 664
ly:otf-glyph-count ... 665
ly:otf-glyph-list ... 665
ly:output-def-clone ... 665
ly:output-def-lookup ... 665
ly:output-def-parent ... 665
ly:output-def-scope ... 665
ly:output-def-set-variable! ... 665
ly:output-def? ... 665
ly:output-description ... 665
ly:output-find-context-def ... 665
ly:outputter-close ... 665
ly:outputter-dump-stencil ... 665
ly:outputter-dump-string ... 665
ly:outputter-output-scheme ... 665
ly:outputter-port ... 665
ly:page-marker? ... 666
ly:page-turn-breaking ... 666
ly:pango-font-physical-fonts ... 666
ly:pango-font? ... 666
ly:paper-book-header ... 666
ly:paper-book-pages ... 666
ly:paper-book-paper ... 666
ly:paper-book-performances ... 666
ly:paper-book-scopes ... 666
ly:paper-book-systems ... 666
ly:paper-book? ... 666
ly:paper-column::break-align-width ... 666
ly:paper-column::print ... 666
ly:paper-fonts ... 667
ly:paper-get-font ... 667
ly:paper-get-number ... 667
ly:paper-outputscale ... 667
ly:paper-score-paper-systems ... 667
ly:paper-system-minimum-distance ... 667
ly:paper-system? ... 667
ly:parse-file ... 667
ly:parse-init ... 667
ly:parse-string-expression ... 667
ly:parsed-undead-list! ... 667
ly:parser-clear-error ... 667
ly:parser-clone ... 667
ly:parser-define! ... 667
ly:parser-error ... 667
ly:parser-has-error? ... 668
ly:parser-include-string ... 668
ly:parser-lookup ... 668
ly:parser-output-name ... 668
ly:parser-parse-string ... 668
ly:parser-set-note-names ... 668
ly:performance-headers ... 668
ly:performance-write ... 668
ly:pitch-alteration ... 668
ly:pitch-diff ... 668
ly:pitch-negate ... 668
ly:pitch-notename ... 668
ly:pitch-octave ... 668
ly:pitch-quartertones ... 668
ly:pitch-semitones ... 668
ly:pitch-steps ... 669
ly:pitch-tones ... 669
ly:pitch-transpose ... 669
ly:pitch<? ... 668

ly:pitch? ... 668
ly:pointer-group-interface::add-grob ... 669
ly:position-on-line? ... 669
ly:prob-immutable-properties ... 669
ly:prob-mutable-properties ... 669
ly:prob-property ... 669
ly:prob-property? ... 669
ly:prob-set-property! ... 669
ly:prob-type? ... 669
ly:prob? ... 669
ly:programming-error ... 669
ly:progress ... 669
ly:property-lookup-stats ... 669
ly:protects ... 669
ly:pt ... 670
ly:pure-call ... 670
ly:randomize-rand-seed ... 670
ly:register-stencil-expression ... 670
ly:register-translator ... 670
ly:relative-group-extent ... 670
ly:rename-file ... 670
ly:reset-all-fonts ... 670
ly:round-filled-box ... 671
ly:round-polygon ... 671
ly:run-translator ... 671
ly:score-add-output-def! ... 672
ly:score-embedded-format ... 672
ly:score-error? ... 672
ly:score-header ... 672
ly:score-music ... 672
ly:score-output-defs ... 672
ly:score-set-header! ... 672
ly:score? ... 672
ly:separation-item::print ... 672
ly:set-color-names ... 673
ly:set-default-scale ... 673
ly:set-grob-creation-callback ... 673
ly:set-grob-modification-callback ... 673
ly:set-middle-C! ... 673
ly:set-option ... 673
ly:set-origin! ... 673
ly:set-property-cache-callback ... 674
ly:skyline-empty? ... 674
ly:skyline-pair? ... 674
ly:skyline? ... 674
ly:smob-protects ... 674
ly:solve-spring-rod-problem ... 674
ly:source-file? ... 674
ly:source-files ... 674
ly:span-bar::before-line-breaking ... 674
ly:span-bar::calc-glyph-name ... 675
ly:span-bar::print ... 675
ly:span-bar::width ... 675
ly:spanner-bound ... 675
ly:spanner-broken-into ... 675
ly:spanner-set-bound! ... 675
ly:spanner? ... 675
ly:spawn ... 675
ly:spring-set-inverse-compress-strength! ... 675
ly:spring-set-inverse-stretch-strength! ... 675
ly:spring? ... 675
ly:staff-symbol-line-thickness ... 676
ly:staff-symbol-staff-radius ... 676
ly:staff-symbol-staff-space ... 676
ly:stderr-redirect ... 676
ly:stencil-add ... 676
ly:stencil-aligned-to ... 676
ly:stencil-combine-at-edge ... 676
ly:stencil-empty? ... 676
ly:stencil-expr ... 676
ly:stencil-extent ... 676
ly:stencil-in-color ... 676
ly:stencil-outline ... 676
ly:stencil-rotate ... 676
ly:stencil-rotate-absolute ... 677
ly:stencil-scale ... 677
ly:stencil-stack ... 677
ly:stencil-translate ... 677
ly:stencil-translate-axis ... 677
ly:stencil? ... 676
ly:stream-event? ... 678
ly:string-percent-encode ... 678
ly:string-substitute ... 678
ly:system-font-load ... 678
ly:text-interface::interpret-markup ... 678
ly:time-signature::print ... 678
ly:transform->list ... 679
ly:transform? ... 679
ly:translate-cpp-warning-scheme ... 679
ly:translator-context ... 679
ly:translator-description ... 679
ly:translator-group? ... 679
ly:translator-name ... 679
ly:translator? ... 679
ly:transpose-key-alist ... 679
ly:ttf->pfa ... 679
ly:ttf-ps-name ... 679
ly:type1->pfa ... 679
ly:unit ... 679
ly:unpure-call ... 680
ly:unpure-pure-container-pure-part ... 680
ly:unpure-pure-container-unpure-part ... 680
ly:unpure-pure-container? ... 680
ly:usage ... 680
ly:verbose-output? ... 680
ly:version ... 680
ly:version? ... 680
ly:volta-bracket::calc-shorten-pair ... 680
ly:warning ... 680
ly:warning-located ... 680
ly:wide-char->utf-8 ... 680
lyric-text::print ... 655

M

make-bow-stencil ... 655
make-c-time-signature-markup ... 655
make-circle-stencil ... 655
make-clef-set ... 655
make-connected-line ... 655
make-connected-path-stencil ... 655
make-cue-clef-set ... 656
make-cue-clef-unset ... 656
make-duration-of-length ... 656
make-ellipse-stencil ... 656
make-filled-box-stencil ... 656
make-glyph-time-signature-markup ... 656
make-grob-property-override ... 656
make-grob-property-revert ... 656

make-grob-property-set 656
make-harmonic 657
make-line-stencil 657
make-modal-inverter 657
make-modal-transposer 657
make-music 657
make-oval-stencil 658
make-part-combine-context-changes 658
make-part-combine-marks 658
make-partial-ellipse-stencil 658
make-path-stencil 658
make-repeat 658
make-semitone->pitch 659
make-stencil-boxer 659
make-stencil-circler 659
make-tmpfile 659
make-transparent-box-stencil 660
map-selected-alist-keys 660
map-some-music 660
markup-command-list? 660
markup-list? 660
measure-counter::text 660
mensural-flag 660
midi-program 660
mmrest-of-length 661
modern-straight-flag 661
music->make-music 662
music-clone 662
music-filter 662
music-is-of-type? 662
music-map 662
music-pitches 663
music-selective-filter 663
music-selective-map 663
music-separator? 663
music-type-predicate 663

N

neo-modern-accidental-rule 663
no-flag 663
normal-flag 663
note-name->markup 663
note-name->string 664
note-to-cluster 664
number-format 664

O

offset-fret 664
offsetter 664
old-straight-flag 664
output-module? 665
oval-stencil 665
override-head-style 665
override-time-signature-setting 666

P

pango-pf-file-name 666
pango-pf-font-name 666
pango-pf-fontindex 666
parenthesize-stencil 667
parse-terse-string 667
percussion? 668
polar->rectangular 669
pure-chain-offset-callback 670

R

ratio->fret 670
ratio->pitch 670
read-lily-expression 670
recording-group-emulate 670
remove-grace-property 670
remove-whitespace 670
retrieve-glyph-flag 670
retrograde-music 671
revert-fontSize 671
revert-head-style 671
revert-props 671
rounded-box-stencil 671

S

scale-beam-thickness 671
scale-fontSize 671
scale-layout 671
scale-props 672
scorify-music 672
seconds->moment 672
select-head-glyph 672
self-alignment-interface::
  self-aligned-on-breakable 672
sequential-music-to-chord-exceptions 673
set-accidental-style 673
set-global-staff-size 673
set-mus-properties! 673
shift-one-duration-log 674
shift-right-at-line-begin 674
skip->rest 674
skip-of-length 674
skip-of-moment-span 674
span-bar::compound-bar-line 675
Span_stem_engraver 675
split-list-by-separator 675
stack-lines 675
stack-stencil-line 675
stack-stencils 675
stack-stencils-padding-list 676
stencil-whiteout 677
stencil-whiteout-box 677
stencil-whiteout-outline 677
straight-flag 677
style-note-heads 678
symbol-concatenate 678
symbol-key<? 678
symbol<? 678

T

tag-group-get 678
tags-keep-predicate 678
tags-remove-predicate 678
teaching-accidental-rule 678

U

unfold-repeats 679
unfold-repeats-fully 679
uniq-list 679
unity-if-multimeasure 679

V

voicify-music 680
volta-bracket::calc-hook-visibility 680
volta-spec-music 680

W

write-me 680