

HARVARD UNIVERSITY  
Graduate School of Arts and Sciences



DISSERTATION ACCEPTANCE CERTIFICATE

The undersigned, appointed by the

Department of Music

have examined a dissertation entitled

*"A Computational Model of Music Composition"*

presented by

Josiah Wolf Oberholtzer

candidate for the degree of Doctor of Philosophy and hereby  
certify that it is worthy of acceptance.

Signature \_\_\_\_\_

Prof. Chaya Czernowin

Signature \_\_\_\_\_

Prof. Hans Tutschku

Signature \_\_\_\_\_

Prof. Christopher Hasty

Date: March 27, 2015



# A Computational Model of Music Composition

A DISSERTATION PRESENTED  
BY  
JOSIAH WOLF OBERHOLTZER  
TO  
THE DEPARTMENT OF MUSIC

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
IN THE SUBJECT OF  
MUSIC COMPOSITION

HARVARD UNIVERSITY  
CAMBRIDGE, MASSACHUSETTS  
MAY 2015

©2015 – JOSIAH WOLF OBERHOLTZER  
ALL RIGHTS RESERVED.

# A Computational Model of Music Composition

## ABSTRACT

This thesis documents my research into formalized score control, in order to demonstrate a computational model of music composition. When working computationally, models provide an explicit formal description of what objects exist within a given domain, how they behave, and what transformations they afford. The clearer the model becomes, the easier it is to extend and to construct increasingly higher-order abstractions around that model. In other words, a clear computational model of music notation affords the development of a clear model of music composition. The Abjad API for Formalized Score Control, an open-source software library written in the Python programming language and making use of the LilyPond automated typesetting system for graphical output, is presented as such a computational model of music notation. My own compositional modeling work, extending Abjad, is introduced and analyzed in the Python library Consort. A collection of five scores, each implemented as Python packages extending these software libraries, are included. Three of these scores, “Zaira”, “Armilla” and “Ersilia”, rely on Consort as their compositional engine, and are presented along with their complete sources. These scores demonstrate my development as a composer investigating the role of computation in music, and display a variety of large-scale structures and musical textures made possible when working with such modeling tools.

This page intentionally left blank.

# Contents

<b>I Theory</b>	<b>1</b>
1 INTRODUCTION	3
1.1 Background & Motivation . . . . .	5
1.2 Overview of the dissertation . . . . .	9
2 <i>ABJAD</i> : A MODEL OF NOTATION	11
2.1 Representing objects . . . . .	14
2.2 Components . . . . .	18
2.2.1 Containers . . . . .	19
2.2.2 Parentage . . . . .	22
2.2.3 Durations . . . . .	25
2.2.4 Named components . . . . .	30
2.3 Indicators . . . . .	33
2.3.1 Indicator scope . . . . .	35
2.3.2 Annotations . . . . .	39
2.4 Spanners . . . . .	41
2.5 Typographic overrides . . . . .	48
2.6 Selecting components . . . . .	52
2.6.1 Logical ties . . . . .	54
2.6.2 Iteration . . . . .	56
2.6.3 Selectors . . . . .	61
2.7 Templating & persistence . . . . .	65
2.8 Other tools . . . . .	71
2.8.1 Mutation . . . . .	71
2.8.2 Pitch modeling . . . . .	71
2.8.3 Parsers . . . . .	72
2.8.4 Notation factories . . . . .	73
2.9 Conclusion . . . . .	74
3 MODELING TIME, RHYTHM AND METER	77
3.1 Timespans . . . . .	78
3.1.1 Annotated timespans . . . . .	80
3.1.2 Time relations . . . . .	82
3.1.3 Operations on timespans . . . . .	85
3.2 Timespan inventories . . . . .	91
3.2.1 Operations on timespan inventories . . . . .	95

3.3	Annotated timespans in Consort . . . . .	101
3.4	Timespan-makers . . . . .	103
3.4.1	Flooded timespan-makers . . . . .	103
3.4.2	Talea timespan-makers . . . . .	106
3.4.3	Dependent timespan-makers . . . . .	113
3.5	Rhythm-makers . . . . .	118
3.5.1	Note rhythm-makers . . . . .	119
3.5.2	Talea rhythm-makers . . . . .	120
3.5.3	Incised rhythm-makers . . . . .	123
3.5.4	Even-division rhythm-makers . . . . .	125
3.5.5	Composite rhythm-makers . . . . .	127
3.6	Meter . . . . .	128
3.7	Rewriting meters . . . . .	132
3.7.1	Dot count . . . . .	134
3.7.2	Boundary depth . . . . .	135
3.7.3	Recursive meter rewriting . . . . .	136
3.7.4	Examples . . . . .	137
3.8	Finding meters . . . . .	138
3.8.1	Offset counters . . . . .	138
3.8.2	Metric accent kernels . . . . .	140
3.8.3	Meter fitting . . . . .	142
3.8.4	Examples . . . . .	143
3.9	Synthesizing time techniques . . . . .	147
3.9.1	Organizing meter . . . . .	150
3.9.2	Organizing timespans . . . . .	150
3.9.3	Populating voices . . . . .	151
3.9.4	Rewriting meters . . . . .	152
3.9.5	Score post-processing . . . . .	153
3.9.6	Examples . . . . .	154
4	<i>CONSORT: A MODEL OF COMPOSITION</i>	159
4.1	Specification . . . . .	160
4.1.1	Segment-makers . . . . .	160
4.1.2	Music settings . . . . .	161
4.1.3	Music specifiers . . . . .	162
4.2	Interpretation . . . . .	168
4.3	Rhythmic interpretation . . . . .	168
4.3.1	Populating the maquette . . . . .	169
4.3.2	Resolving cascading overlap . . . . .	175
4.3.3	Finding meters, revisited . . . . .	180
4.3.4	Splitting, pruning & consolidation . . . . .	181
4.3.5	Inscription . . . . .	183
4.3.6	Meter pruning . . . . .	188
4.3.7	Populating dependent timespans . . . . .	188
4.3.8	Populating silent timespans . . . . .	189
4.3.9	Rewriting meters, revisited . . . . .	190
4.3.10	Populating the score . . . . .	194
4.4	Non-rhythmic interpretation . . . . .	194

4.4.1	Score traversal . . . . .	194
4.4.2	Grace-handlers . . . . .	200
4.4.3	Pitch-handlers . . . . .	201
4.4.4	Attachment-handlers . . . . .	210
4.4.5	Expressive attachments . . . . .	215
4.4.6	Post-processing . . . . .	216
4.5	Persistence & visualization . . . . .	217
<b>5</b>	<b>COMPOSITION AS SOFTWARE DEVELOPMENT</b>	<b>219</b>
5.1	Score directory layout . . . . .	220
5.1.1	Makers . . . . .	221
5.1.2	Material packages . . . . .	222
5.1.3	Segment packages . . . . .	223
5.1.4	The stylesheets/ directory . . . . .	224
5.1.5	The build/ directory . . . . .	225
5.1.6	The etc/ and distribution/ directories . . . . .	227
5.1.7	The test/ directory . . . . .	227
5.1.8	Python packaging . . . . .	228
5.2	Document preparation in detail . . . . .	228
5.2.1	Build tools . . . . .	228
5.2.2	Illustrating & persisting segments . . . . .	229
5.2.3	Collecting and concatenating segment illustrations . . . . .	230
5.2.4	Organizing and typesetting LaTeX assets . . . . .	233
5.2.5	Part extraction . . . . .	236
5.3	Project maintenance . . . . .	239
5.3.1	Automated regression testing . . . . .	239
5.3.2	Version control . . . . .	240
<b>6</b>	<b>CONCLUSION</b>	<b>243</b>
6.1	Concerns & Implications . . . . .	245
6.2	Future work . . . . .	248
6.3	Parting words . . . . .	248
<b>II</b>	<b>Practice</b>	<b>251</b>
<b>7</b>	<i>AURORA</i> (2011)	<b>253</b>
<b>8</b>	<i>PLAGUE WATER</i> (2014)	<b>307</b>
<b>9</b>	<i>INVISIBLE CITIES (I): ZAIRA</i> (2014)	<b>343</b>
<b>10</b>	<i>INVISIBLE CITIES (II): ARMILLA</i> (2015)	<b>369</b>
<b>11</b>	<i>INVISIBLE CITIES (III): ERSILIA</i> (2015)	<b>385</b>

### III Appendices 435

APPENDIX A <i>CONSORT SOURCE</i>	<b>437</b>
A.1 consort.tools.AbsolutePitchHandler . . . . .	437
A.2 consort.tools.AfterGraceSelectorCallback . . . . .	439
A.3 consort.tools.annotate . . . . .	439
A.4 consort.tools.AttachmentExpression . . . . .	441
A.5 consort.tools.AttachmentHandler . . . . .	446
A.6 consort.tools.AttackPointSignature . . . . .	449
A.7 consort.tools.BoundaryTimespanMaker . . . . .	453
A.8 consort.tools.ChordExpression . . . . .	459
A.9 consort.tools.ChordSpecifier . . . . .	465
A.10 consort.tools.ClefSpanner . . . . .	465
A.11 consort.tools.ClefSpannerExpression . . . . .	469
A.12 consort.tools.ComplexPianoPedalSpanner . . . . .	470
A.13 consort.tools.ComplexTextSpanner . . . . .	471
A.14 consort.tools.CompositeMusicSpecifier . . . . .	478
A.15 consort.tools.CompositeRhythmMaker . . . . .	484
A.16 consort.tools.ConsortTrillSpanner . . . . .	490
A.17 consort.tools.Cursor . . . . .	493
A.18 consort.tools.DependentTimespanMaker . . . . .	496
A.19 consort.tools.DynamicExpression . . . . .	502
A.20 consort.tools.FloodedTimespanMaker . . . . .	517
A.21 consort.tools.GraceHandler . . . . .	519
A.22 consort.tools.HarmonicExpression . . . . .	522
A.23 consort.tools.HashCachingObject . . . . .	524
A.24 consort.tools.KeyClusterExpression . . . . .	525
A.25 consort.tools.LeafExpression . . . . .	528
A.26 consort.tools.LogicalTieExpression . . . . .	529
A.27 consort.tools.MusicSetting . . . . .	530
A.28 consort.tools.MusicSpecifier . . . . .	539
A.29 consort.tools.MusicSpecifierSequence . . . . .	545
A.30 consort.tools.OctavationExpression . . . . .	548
A.31 consort.tools.PerformedTimespan . . . . .	549
A.32 consort.tools.PhrasedSelectorCallback . . . . .	553
A.33 consort.tools.PitchClassPitchHandler . . . . .	554
A.34 consort.tools.PitchHandler . . . . .	560
A.35 consort.tools.PitchOperationSpecifier . . . . .	572
A.36 consort.tools.PitchSpecifier . . . . .	579
A.37 consort.tools.Proportions . . . . .	585
A.38 consort.tools.RatioPartsExpression . . . . .	586
A.39 consort.tools.RegisterInflection . . . . .	588
A.40 consort.tools.RegisterInflectionInventory . . . . .	596
A.41 consort.tools.RegisterSpecifier . . . . .	596
A.42 consort.tools.RegisterSpecifierInventory . . . . .	601
A.43 consort.tools.ScoreTemplate . . . . .	602
A.44 consort.tools.ScoreTemplateManager . . . . .	604
A.45 consort.tools.SeedSession . . . . .	609

A.46	consort.tools.SegmentMaker . . . . .	612
A.47	consort.tools.SilentTimespan . . . . .	667
A.48	consort.tools.SimpleDynamicExpression . . . . .	669
A.49	consort.tools.StopTrillSpan . . . . .	672
A.50	consort.tools.StringContactSpanner . . . . .	673
A.51	consort.tools.StringQuartetScoreTemplate . . . . .	683
A.52	consort.tools.TaleaTimespanMaker . . . . .	690
A.53	consort.tools.TextSpannerExpression . . . . .	700
A.54	consort.tools.TimespanCollection . . . . .	702
A.55	consort.tools.TimespanCollectionNode . . . . .	720
A.56	consort.tools.TimespanInventoryMapping . . . . .	724
A.57	consort.tools.TimespanMaker . . . . .	724
A.58	consort.tools.TimespanSimultaneity . . . . .	728
A.59	consort.tools.TimespanSpecifier . . . . .	730
<b>APPENDIX B</b>	<b><i>ZAIRA</i> SOURCE CODE</b>	<b>733</b>
B.1	<i>zaira</i> makers source . . . . .	733
B.1.1	zaira.makers.Percussion . . . . .	733
B.1.2	zaira.makers.ZairaScoreTemplate . . . . .	733
B.1.3	zaira.makers.ZairaSegmentMaker . . . . .	739
B.2	<i>zaira</i> materials source . . . . .	740
B.2.1	zaira.materials.background_dynamic_attachment_expression . . . . .	740
B.2.2	zaira.materials.brazil_nut_music_specifier . . . . .	741
B.2.3	zaira.materials.cello_solo_music_specifier . . . . .	742
B.2.4	zaira.materials.dense_timespan_maker . . . . .	743
B.2.5	zaira.materials.drum_agitation_music_specifier . . . . .	743
B.2.6	zaira.materials.drum_brushed_music_specifier . . . . .	744
B.2.7	zaira.materials.drum_heartbeat_music_specifier . . . . .	745
B.2.8	zaira.materials.drum_storm_music_specifier . . . . .	745
B.2.9	zaira.materials.drum_tranquilo_music_specifier . . . . .	746
B.2.10	zaira.materials.erratic_dynamic_attachment_expression . . . . .	747
B.2.11	zaira.materials.flourish_rhythm_maker . . . . .	748
B.2.12	zaira.materials.foreground_dynamic_attachment_expression . . . . .	748
B.2.13	zaira.materials.glissando_rhythm_maker . . . . .	749
B.2.14	zaira.materials.granular_timespan_maker . . . . .	749
B.2.15	zaira.materials.laissez_vibrer_attachment_expression . . . . .	750
B.2.16	zaira.materials.legato_rhythm_maker . . . . .	750
B.2.17	zaira.materials.legato_timespan_maker . . . . .	751
B.2.18	zaira.materials.metal_agitation_music_specifier . . . . .	751
B.2.19	zaira.materials.metal_brushed_music_specifier . . . . .	752
B.2.20	zaira.materials.metal_tranquilo_music_specifier . . . . .	753
B.2.21	zaira.materials.midground_dynamic_attachment_expression . . . . .	754
B.2.22	zaira.materials.oboe_solo_music_specifier . . . . .	754
B.2.23	zaira.materials.overpressure_attachment_expression . . . . .	755
B.2.24	zaira.materials.pedals_timespan_maker . . . . .	756
B.2.25	zaira.materials_percussion_brushed_music_specifier . . . . .	756
B.2.26	zaira.materials_percussion_brushed_tremolo_music_specifier . . . . .	756
B.2.27	zaira.materials_percussion_fanfare_music_specifier . . . . .	757

B.2.28	<i>zaira.materials.percussion_reiteration_music_specifier</i>	757
B.2.29	<i>zaira.materials.percussion_superball_music_specifier</i>	758
B.2.30	<i>zaira.materials.piano_clusters_music_specifier</i>	758
B.2.31	<i>zaira.materials.piano_drone_music_specifier</i>	759
B.2.32	<i>zaira.materials.piano_fanfare_music_specifier</i>	760
B.2.33	<i>zaira.materials.piano_flourish_music_specifier</i>	761
B.2.34	<i>zaira.materials.piano_guero_music_specifier</i>	762
B.2.35	<i>zaira.materials.piano_pedals_music_specifier</i>	763
B.2.36	<i>zaira.materials.piano_prepared_bass_music_specifier</i>	763
B.2.37	<i>zaira.materials.piano_prepared_treble_music_specifier</i>	764
B.2.38	<i>zaira.materials.proportions</i>	766
B.2.39	<i>zaira.materials.registerSpecifier_inventory</i>	766
B.2.40	<i>zaira.materials.reiterating_rhythm_maker</i>	769
B.2.41	<i>zaira.materials.sparse_timespan_maker</i>	769
B.2.42	<i>zaira.materials.string_chord_music_specifier</i>	770
B.2.43	<i>zaira.materials.string_flourish_music_specifier</i>	770
B.2.44	<i>zaira.materials.string_trills_music_specifier</i>	771
B.2.45	<i>zaira.materials.string_tutti_overpressure_music_specifier</i>	772
B.2.46	<i>zaira.materials.string_undergrowth_music_specifier</i>	773
B.2.47	<i>zaira.materials.stuttering_rhythm_maker</i>	774
B.2.48	<i>zaira.materials.sustained_rhythm_maker</i>	774
B.2.49	<i>zaira.materials.sustained_timespan_maker</i>	774
B.2.50	<i>zaira.materials.tempi</i>	775
B.2.51	<i>zaira.materials.time_signatures</i>	775
B.2.52	<i>zaira.materials.total_duration_in_seconds</i>	776
B.2.53	<i>zaira.materials.tutti_timespan_maker</i>	776
B.2.54	<i>zaira.materials.undergrowth_rhythm_maker</i>	776
B.2.55	<i>zaira.materials.wind_airtone_music_specifier</i>	777
B.2.56	<i>zaira.materials.wind_keyclick_music_specifier</i>	777
B.2.57	<i>zaira.materials.wind_slap_music_specifier</i>	778
B.2.58	<i>zaira.materials.wind_trills_music_specifier</i>	779
B.2.59	<i>zaira.materials.wood_agitation_music_specifier</i>	780
B.2.60	<i>zaira.materials.wood_bamboo_music_specifier</i>	781
B.3	<i>zaira segments source</i>	782
B.3.1	<i>zaira.segments.segment_a</i>	782
B.3.2	<i>zaira.segments.segment_b</i>	783
B.3.3	<i>zaira.segments.segment_c</i>	787
B.3.4	<i>zaira.segments.segment_d</i>	793
B.3.5	<i>zaira.segments.segment_e</i>	798
B.3.6	<i>zaira.segments.segment_f1</i>	804
B.3.7	<i>zaira.segments.segment_f2</i>	810
B.3.8	<i>zaira.segments.segment_g</i>	814
B.3.9	<i>zaira.segments.segment_h</i>	818
B.3.10	<i>zaira.segments.segment_i</i>	825
B.3.11	<i>zaira.segments.segment_j</i>	829
B.3.12	<i>zaira.segments.segment_k</i>	832
B.4	<i>zaira stylesheet source</i>	833

B.4.1	stylesheet.ily . . . . .	833
<b>APPENDIX C <i>ARMILLA</i> SOURCE CODE</b>		<b>843</b>
C.1	<i>armilla</i> makers source . . . . .	843
C.1.1	armilla.makers.ArmillaScoreTemplate . . . . .	843
C.1.2	armilla.makers.ArmillaSegmentMaker . . . . .	846
C.2	<i>armilla</i> materials source . . . . .	850
C.2.1	armilla.materials.dense_timespan_maker . . . . .	850
C.2.2	armilla.materials.intermittent_accents . . . . .	851
C.2.3	armilla.materials.intermittent_circular . . . . .	851
C.2.4	armilla.materials.intermittent_glissandi . . . . .	851
C.2.5	armilla.materials.intermittent_tremoli . . . . .	852
C.2.6	armilla.materials.intermittent_trills . . . . .	852
C.2.7	armilla.materials.left_hand_diads_music_specifier . . . . .	853
C.2.8	armilla.materials.left_hand_dietro_music_specifier . . . . .	854
C.2.9	armilla.materials.left_hand_glissandi_music_specifier . . . . .	855
C.2.10	armilla.materials.left_hand_pizzicati_music_specifier . . . . .	856
C.2.11	armilla.materials.left_hand_stasis_music_specifier . . . . .	857
C.2.12	armilla.materials.right_hand_circular_music_specifier . . . . .	857
C.2.13	armilla.materials.right_hand_jete_music_specifier . . . . .	859
C.2.14	armilla.materials.right_hand_overpressure_music_specifier . . . . .	860
C.2.15	armilla.materials.right_hand_pizzicati_music_specifier . . . . .	862
C.2.16	armilla.materials.right_hand_stasis_music_specifier . . . . .	863
C.2.17	armilla.materials.sparse_timespan_maker . . . . .	864
C.2.18	armilla.materials.sustained_timespan_maker . . . . .	864
C.2.19	armilla.materials.synchronized_timespan_maker . . . . .	865
C.2.20	armilla.materials.time_signatures . . . . .	865
C.3	<i>armilla</i> segments source . . . . .	866
C.3.1	armilla.segments.segment_a_far_sorr . . . . .	866
C.3.2	armilla.segments.segment_b_selidor_a . . . . .	868
C.3.3	armilla.segments.segment_c_wellogy . . . . .	869
C.3.4	armilla.segments.segment_d_the_long_dune_a . . . . .	871
C.3.5	armilla.segments.segment_e_selidor_b . . . . .	874
C.3.6	armilla.segments.segment_f_the_isle_of_the_ear . . . . .	875
C.3.7	armilla.segments.segment_g_selidor_c . . . . .	877
C.3.8	armilla.segments.segment_h_the_long_dune_b . . . . .	878
C.4	<i>armilla</i> stylesheet source . . . . .	881
C.4.1	stylesheet.ily . . . . .	881
<b>APPENDIX D <i>ERSILIA</i> SOURCE CODE</b>		<b>889</b>
D.1	<i>ersilia</i> makers source . . . . .	889
D.1.1	ersilia.makers.ErsiliaScoreTemplate . . . . .	889
D.1.2	ersilia.makers.ErsiliaSegmentMaker . . . . .	897
D.1.3	ersilia.makers.Percussion . . . . .	899
D.2	<i>ersilia</i> materials source . . . . .	900
D.2.1	ersilia.materials.abbreviations . . . . .	900
D.2.2	ersilia.materials.dense_timespan_maker . . . . .	902
D.2.3	ersilia.materials.guitar_agitato_music_specifier . . . . .	903

D.2.4	<code>ersilia.materials.guitar_continuo_music_specifier</code>	905
D.2.5	<code>ersilia.materials.guitar_pointillist_harmonics_music_specifier</code>	906
D.2.6	<code>ersilia.materials.guitar_strummed_music_specifier</code>	907
D.2.7	<code>ersilia.materials.guitar_tremolo_music_specifier</code>	908
D.2.8	<code>ersilia.materials.guitar_undulation_tremolo_music_specifier</code>	909
D.2.9	<code>ersilia.materials.percussion_bamboo_windchimes_music_specifier</code>	910
D.2.10	<code>ersilia.materials.percussion_crotalates_flash_music_specifier</code>	912
D.2.11	<code>ersilia.materials.percussion_crotalates_interruption_music_specifier</code>	912
D.2.12	<code>ersilia.materials.percussion_low_pedal_music_specifier</code>	914
D.2.13	<code>ersilia.materials.percussion_marimba_agitato_music_specifier</code>	916
D.2.14	<code>ersilia.materials.percussion_marimba_ostinato_music_specifier</code>	918
D.2.15	<code>ersilia.materials.percussion_marimba_tremolo_music_specifier</code>	919
D.2.16	<code>ersilia.materials.percussion_snare_interruption_music_specifier</code>	920
D.2.17	<code>ersilia.materials.percussion_temple_block_fanfare_music_specifier</code>	922
D.2.18	<code>ersilia.materials.percussion_tom_fanfare_music_specifier</code>	924
D.2.19	<code>ersilia.materials.permitted_time_signatures</code>	926
D.2.20	<code>ersilia.materials.piano_agitato_music_specifier</code>	926
D.2.21	<code>ersilia.materials.piano_arm_cluster_music_specifier</code>	928
D.2.22	<code>ersilia.materials.piano_glissando_music_specifier</code>	929
D.2.23	<code>ersilia.materials.piano_palm_cluster_music_specifier</code>	930
D.2.24	<code>ersilia.materials.piano_pedals_music_setting</code>	931
D.2.25	<code>ersilia.materials.piano_pointillist_music_specifier</code>	932
D.2.26	<code>ersilia.materials.piano_string_glissando_music_specifier</code>	933
D.2.27	<code>ersilia.materials.piano_tremolo_music_specifier</code>	934
D.2.28	<code>ersilia.materials.pitch_pipe_music_specifier</code>	935
D.2.29	<code>ersilia.materials.saxophone_agitato_music_specifier</code>	936
D.2.30	<code>ersilia.materials.shaker_decelerando_music_specifier</code>	938
D.2.31	<code>ersilia.materials.shaker_spodic_music_specifier</code>	939
D.2.32	<code>ersilia.materials.shaker_tremolo_music_specifier</code>	940
D.2.33	<code>ersilia.materials.sparse_timespan_maker</code>	941
D.2.34	<code>ersilia.materials.string_agitato_music_specifier</code>	941
D.2.35	<code>ersilia.materials.string_legato_music_specifier</code>	943
D.2.36	<code>ersilia.materials.string_low_pedal_music_specifier</code>	945
D.2.37	<code>ersilia.materials.string_ostinato_music_specifier</code>	946
D.2.38	<code>ersilia.materials.string_overpressure_music_specifier</code>	947
D.2.39	<code>ersilia.materials.string_pointillist_music_specifier</code>	948
D.2.40	<code>ersilia.materials.string_tremolo_music_specifier</code>	949
D.2.41	<code>ersilia.materials.sustained_timespan_maker</code>	950
D.2.42	<code>ersilia.materials.tutti_timespan_maker</code>	951
D.2.43	<code>ersilia.materials.wind_agitato_music_specifier</code>	951
D.2.44	<code>ersilia.materials.wind_continuo_music_specifier</code>	954
D.2.45	<code>ersilia.materials.wind_low_pedal_music_specifier</code>	955
D.2.46	<code>ersilia.materials.wind_ostinato_music_specifier</code>	955
D.2.47	<code>ersilia.materials.wind_pointillist_music_specifier</code>	956
D.2.48	<code>ersilia.materials.wind_tremolo_music_specifier</code>	957
D.3	<code>ersilia segments source</code>	958
D.3.1	<code>ersilia.segments.chemish</code>	958

D.3.2	ersilia.segments.cut_1 . . . . .	963
D.3.3	ersilia.segments.cut_2 . . . . .	964
D.3.4	ersilia.segments.komokome . . . . .	965
D.3.5	ersilia.segments.sort . . . . .	972
D.4	<i>ersilia</i> stylesheet source . . . . .	978
D.4.1	stylesheet.ily . . . . .	978
	REFERENCES	989



*The Astrologer*  
Hans Holbein  
*The Dance of Death*  
(c.1527)

FOR CINDY

A Sorcerer by the power of his magick had subdued all  
things to himself.

Would he travel? He could fly through space more  
swiftly than the stars.

Would he eat, drink, and take his pleasure? There was  
none that did not instantly obey his bidding.

In the whole system of ten million times ten million  
spheres upon the two and twenty million planes  
he had his desire.

And with all this he was but himself.

Alas!

- Aleister Crowley, *The Book of Lies*

This page intentionally left blank.

Part I

Theory

This page intentionally left blank.

# 1

## Introduction

The intent of this dissertation is to document my research into *formalized score control*<sup>4,5,18</sup> in order to demonstrate a *computational model of music composition*. The term formalized score control, inspired by Iannis Xenakis' seminal text *Formalized Music*<sup>21,3</sup>, describes the discipline of modeling and manipulating the typography of common practice notation programmatically via software. An emphasis on *modeling*, here, is crucial. When working computationally, models provide an explicit formal description of what objects exist within a given domain, how they behave, and what transformations they afford. The clearer the model becomes, the easier it is to extend and to construct increasingly higher-order abstractions around that model. That is to say, a clear model of notation – the description of symbols on the page – affords a clear model of composition – how those symbols come to be there.

At the time of this writing, a wide variety of computational models of notation exist<sup>5,18</sup>, expressing an equally wide range of explicitness or implicitness in their modeling, and providing composers with varying degrees of pro-

grammatic control over those models. For example, many composers make use of vector graphics programs for notation, working in an environment which behaves like a hyper-extended writing desk, but which provides no more semantic understanding of the contents of the page than a physical pencil does. That is to say, the software make no categorical distinction between an oval representing a note head and any other line on the page. Likewise, many composers rely on more traditional notation software which mimics the physical page while providing some model of common practice notation, rigidly enforcing a measure-based understanding of musical score on the artist, while still others make use of the idiosyncratic score-modeling systems often developed hand-in-hand with computer-assisted composition tools.<sup>1</sup> Of course it's important to understand that I don't denigrate the use of vector graphics programs to create scores. The relative presence or absence of notation modeling in a tool can be a boon or a hindrance, depending on how it aligns with the needs of the composer using it. While one tool might model notation exquisitely, it still provides programmatic control over only what it can describe, and may even forbid the manipulation of anything beyond that domain. Conversely, another tool, providing only a minimal model of notation, or even none at all, is not so constrained, and while it may not afford programmatic control over the contents of the score at all, it does allow composers to radically reinvent and extend an understanding of Western notation.

My own compositional work now makes extensive use of the *Abjad API for Formalized Score Control*.<sup>4,5,18</sup> Abjad,<sup>2</sup> which originated in 2008 as a joint effort between the composers Víctor Adán and Trevor Bača, and to which I have also been contributing significantly for over six years, extends the *Python*<sup>19</sup> programming language with a rich model of common practice notation, and visualizes the scores it models with *LilyPond*<sup>14</sup>, an automated engraving system inspired by the venerable L<sup>A</sup>T<sub>E</sub>X. On top of Abjad, I have implemented a model of composition in the Python package *Consort*, which allows for the specification of the structure of a score and the processes to be used to create it, and the interpretation of that specification first into Abjad's notation model, and subsequently into publication-quality typography via LilyPond. These twin notational and compositional models bring the col-

---

<sup>1</sup>Composers often make use of all the above categories of tool in a single score, working programmatically with a computer-assisted composition tool to generate material which is then exported into a more traditional notation editor for page layout and further editing, and finally exported as vector graphics for fine-tuning. This work-flow is not reversible. The changes made to the score as vector graphics cannot be propagated back into the semantic model of the score provided by either the traditional score editor or the computer-assisted composition tool.

<sup>2</sup><http://projectabjad.org/>

<sup>3</sup><http://python.org/>

<sup>4</sup><http://lilypond.org/>

lective knowledge of the computing disciplines to bear on a variety of musical questions: how does one arrange large-scale form? What manner of rhythmic, harmonic and timbral transformations are available? How can one describe formal structural relationships between the objects on a page? How can one quickly audition and vary dense, multi-layered orchestral textures? The chapters that follow investigate these two intertwined forms of modeling, attempting to address some of the musical questions posed here, while paying special attention to those details – often missing from treatises on computer-assisted composition – which straddle the line between high-level and low.

Importantly, this dissertation is not a general survey of the techniques used by composers working in computer-assisted composition, but the work of one composer implementing for the specifics of his own process. This work may well be applicable to others in many ways, but I do not claim universality. It is then, more of a tutorial than anything else, presented in order to explain the architectural decisions, implementations and implications behind a system designed to assist in the composition of music.

## 1.1 BACKGROUND & MOTIVATION

Computer-assisted composition has a fairly short history, dating at the time of this writing a little more than half a century. While one could certainly argue about the timeline and origins of the techniques falling under this umbrella – and providing a complete history of computer music is not the goal of this dissertation – it is generally agreed that work began in earnest in the post-war decades of the 20th century.<sup>15,21</sup> Despite its relative youth as a musical tradition, computer-assisted composition has many practitioners and a tremendous number of tools to work with, both generalized and idiosyncratic.

My personal introduction to computer-assisted composition began with Iannis Xenakis' mid-century massed string works,<sup>21</sup> and, combined with a life-long interest in electro-acoustic music dating back to my serendipitous childhood exposure to a copy of Ivan Tcherepnin's *Flores Musicales / Five Songs / Santūr Live!*, quickly progressed through study of Curtis Roads' granular explorations,<sup>16</sup> Trevor Wishart's schema for textural transformations,<sup>20</sup> and Kaaija Saariaho's analytic orchestrations. While I've turned to other music and other considerations since reading these texts and listening to these works, they instilled in me a confidence that *somewhat* computation could be useful for creating the sort of music I wanted to hear but was consistently unable to write, simply sitting at my desk.<sup>5</sup>

---

<sup>5</sup>The jury is still out, but I persist anyway.

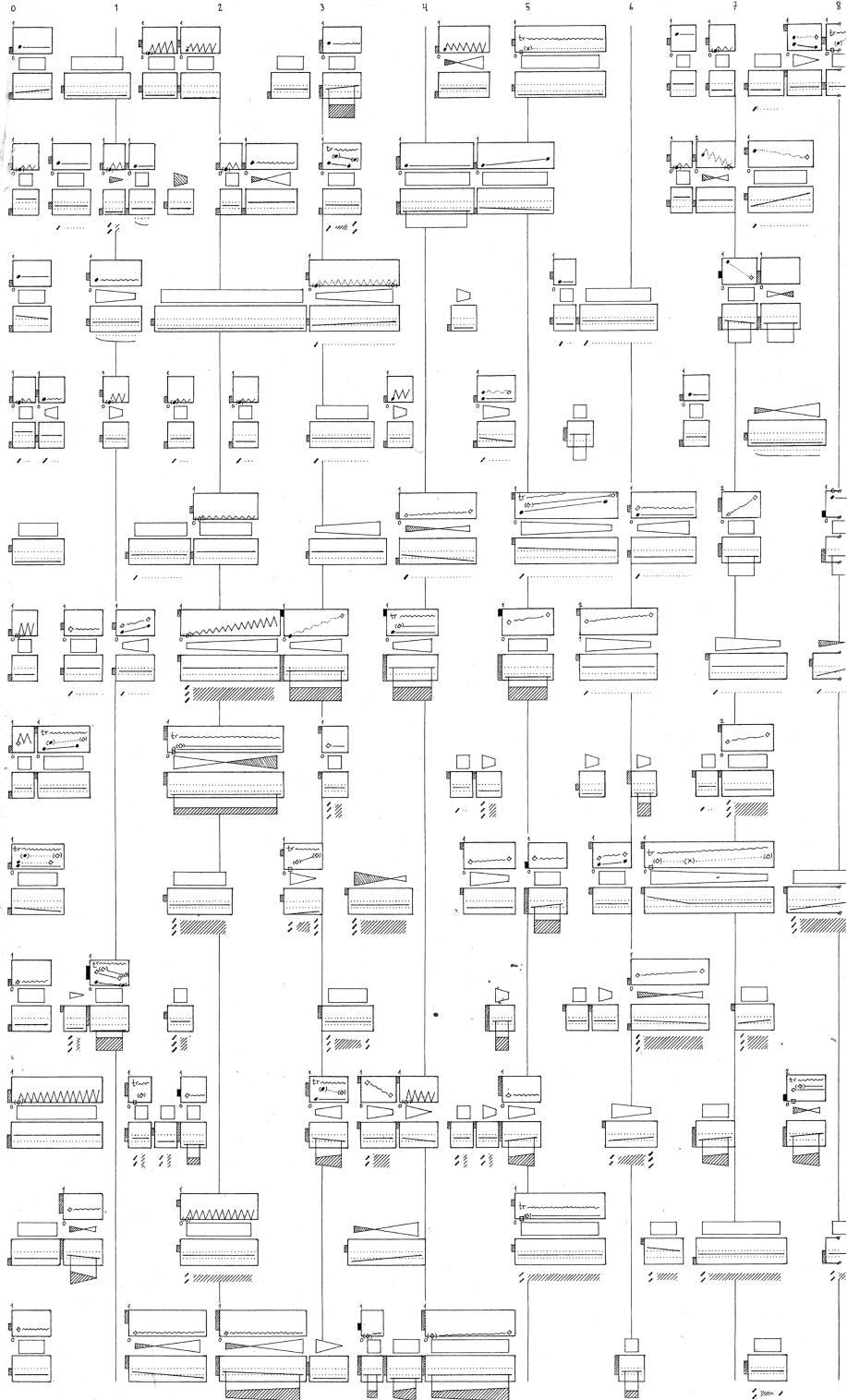


FIGURE 1.1: The first page of *mbs1* (2006), an unfinished tablature score for one to twelve string performers. This represents an early attempt of mine at computationally modeling massed performers with timespans to create an evolving “granular” texture. The instructions for the score were created through a variety of Max/MSP patches which painstakingly converted noise functions into text files which I then collated into spreadsheets. The score itself was drawn by hand on size A1 graph paper with Rapidographs. I completed seventeen of the intended fifty-three pages before stopping for the sake of my wrists and due to a general lack of faith in the project. While still incomplete, the concerns that motivated this score remain with me.

My earliest experiments with computer-assisted composition involved Max/MSP, writing patches to generate tabular data from various noise functions which I collected into spreadsheets and painstakingly notated by hand. That work culminated at the end of my undergraduate study in 2006 with the never-completed score *mbrsi* – excerpted in figure 1.1, with one spreadsheet “source” in figure 1.2 –, whose textural fabric of massed strings attempted to crystallize the possibilities I felt in granular synthesis at the time: a kind of sparkling, nervous energy hovering between a flame and the strange Bezier-cloud outlined by starling swarms. Unfortunately, the process of simply notating the piece by hand was gruelling, and effectively forbade any effort at compositional exploration, let alone revising. In short, while this early work did take lengths to model a compositional process, it lacked any notational model beyond my own hands. And in the absence of any automated typesetting tool, the project collapsed under its inherent labor difficulties. I set *mbrsi* aside and spent the next few years looking for alternative approaches to both formalizing large-scale structure in score, and automating the typesetting of those structures into notation, going so far as to attempt scripting various vector graphics programs directly in order to establish a formalized model of music notation which I could use to complete the project.

A number of years later, at the beginning of my graduate studies, I was introduced to first LilyPond, and then Abjad. LilyPond, much like L<sup>A</sup>T<sub>E</sub>X, is a command-line tool which takes a text file comprising various commands which describe the content of a musical document, and converts those commands into graphic representations of music notation, generally as a PDF. This working modality is often referred to as “what you see is what you mean”: the user of the system edits with the intent to describe their desired structure, rather than manipulating a representation of the end-product itself. The discovery of a such a plain-text-based way of working was a revelation to me, precisely because plain text is eminently susceptible to manipulation. One would be hard-pressed to find *any* programming languages which cannot generate plain-text files. While hardly easy – and I simply couldn’t have known at the time how far off that impression was – a way forward finally seemed possible: I could develop a system to model notation, using LilyPond as its typesetting engine. Luckily, Víctor Adán and Trevor Bača had reached an identical conclusion sometime before, and had already begun the work of implementing a notation model in the Python programming language, using LilyPond as the primary output. And now, nearly six years later, this work continues, unabated.

Of course there were already a tremendous number of composition tools contemporary with my entry into the field, such as OpenMusic,<sup>2</sup> PWGL,<sup>10,9</sup> Common Lisp Music,<sup>17</sup> and somewhat later the BACH<sup>1</sup> library for use with modern versions of Max, as well as a whole host of more marginal experiments including LilyCollider, which

FIGURE 1.2: Page one of fifty-three from a spreadsheet containing fingering instructions for six of the twelve string performers in *mbsri* (2006). Each box consists of the instructions for one performer, over the course of one measure. The instructions in the upper-left box here correspond to the upper-most row of boxes in figure 1.1.

provides an interface between SuperCollider<sup>12</sup> – generally used for live electronics – and LilyPond. Why develop yet another? The answer is both pragmatic, and also simply preferential. I wanted a tool usable from the command-line, something transparently interoperative with the Unix ecosystem, not a graphical application. I wanted it to be open-source, so I could inspect it, break it, and come to understand all of its mechanisms. I wanted it to be free, so I could use it anywhere. And I wanted it to be written in an imperative language, so I could learn it more quickly. Abjad fit all of these desires.

## 1.2 OVERVIEW OF THE DISSERTATION

This dissertation consists of six chapters of prose – including this chapter –, followed by five chapters each presenting a score, and four extensive appendices comprising complete code listings for the implementations of the three most recent scores and of Consort. Please consult the source for Abjad version 2.16<sup>6</sup> directly, as it is far too large to include here.

The first six chapters discuss formalized score control, gradually laying out and building upon the concepts necessary to understand the process of modeling notation and composition computationally. Chapter 2 presents an overview of Abjad, detailing its structure and usage in the creation of scores. The objects comprising Abjad’s core notation model, *components*, *indicators* and *spanners*, are introduced, and the techniques by which those objects are aggregated into scores, inspected, selected, iterated over, mutated, persisted to disk, and visualized are demonstrated at length. Chapter 3 expands on chapter 2, discussing various models of musical time implemented in Abjad, and introducing many of the tools and techniques employed in Consort to create large-scale musical works. Timespans – objects which model a duration of time positioned along a timeline without regard for score hierarchy –, as well as massed collections of timespans, and operations applied against both single timespans and those collections, are presented as a means for modeling high-level phrasing and score structure. Highly-configurable factory classes for programmatically creating both timespan and rhythmic structures are introduced along with a hierarchical model of meter which affords coordination between the two models of time. Chapter 4 analyzes the mechanisms implemented in Consort’s model of composition to specify the structure of musical score at a high level and to interpret those specifications into completely-notated segments of music. The analysis proceeds step-by-step through the process of specification, introducing the reader to Consort’s segment-makers, music specifiers and music settings. The discussion then turns to interpretation, divided into rhythmic- and non-rhythmic stages. The mechanisms by

---

<sup>6</sup><https://github.com/Abjad/abjad/releases/tag/2.16>

which the music settings comprising a score specification are gradually resolved into a maquette consisting of annotated timespans, that maquette interpreted into a score, and grace notes, pitches and attachments applied against that interpreted score are presented, providing a link back to the iteration and selection techniques presented in chapter 2 and the timespan transformations demonstrated in chapter 3. Chapter 5 proposes standardized solutions to practical concerns surrounding the composition of scores in software, such as score package layout on the file-system, typesetting workflows, version control, and testing. The means by which an individual score segment, as described in chapter 4, is combined with many other segments to create a complete work is presented, and the document preparation techniques necessary for automating the extraction of parts in LilyPond and producing the various non-musical component documents of a score with L<sup>A</sup>T<sub>E</sub>X are also introduced. Chapter 6, the conclusion to the prose portion of the dissertation, summarizes the previously presented research, suggests some implications and areas for improvement, and proposes directions for future work.

The remaining chapters consist of five scores, all composed computationally with Abjad and LilyPond, both prior to and after the development of Consort. Chapter 7 presents *Aurora* (2011), for string orchestra, which represents the results of my first formal research into composition with timespan structures, as well as my first formalized work which allowed multiple distinct textures to overlap. This score implemented a very rudimentary version of the “specify & interpret” pattern enshrined in Consort. Chapter 8 presents *Plague Water* (2014), for baritone saxophone, electric guitar, piano and percussion, my first attempt at composing scores in segments through explicit specification and interpretation, and whose code-base provided Consort’s original foundation. Chapters 9, 10 and 11 present a set of pieces, *Invisible Cities (i): Zaira* (2014), for eight players, *Invisible Cities (ii): Armilla* (2015), for viola duet, and *Invisible Cities (iii): Ersilia* (2015), for chamber orchestra, all implemented via Consort. These three scores demonstrate Consort’s flexibility as a tool for structuring both large and small ensemble works of widely divergent textures, notated both conservatively – as in *Zaira* and *Ersilia* – and more unconventionally, with tablature – as in *Armilla*.

Finally, the appendices contain the source to all classes and functions implemented in Consort – as of the time of this writing –, as well as the source to all material and segment definitions, as well as any LilyPond stylesheets, for the three Invisible Cities scores. My sincere hope is that these appendices provide a concrete, “reverse-engineerable” even, link between the descriptions of my research into formalized score control and the scores written with these tools.

# 2

## *Abjad*: a model of notation



BJAD<sup>4,18</sup> IS AN OPEN-SOURCE PYTHON<sup>19</sup> PACKAGE extending the Python programming language with a computational model of music notation. Abjad allows its users to construct scores in an object-oriented fashion and to visualize their work as publication-quality notation at any point during the composition process. Importantly, Abjad is not a stand-alone application and has no GUI – no graphic user interface – unlike many other notation- or composition-modeling projects such as PWGL<sup>10,9</sup>, OpenMusic<sup>2</sup>, BACH<sup>1</sup>, and so forth. Instead, all work with Abjad is done in the command-line, either directly in an interactive Python console session, or by writing modules of code to be imported into one another and executed by Python’s interpreter. Nearly all of the code examples in the body of this document are presented as part of a single interactive Python console session simply because Python’s interactive console clearly distinguishes input from output. Lines preceded by `>>` will be passed to Python for interpretation. Those preceded by `...` indicate the continuation of a

incomplete language construct. All other lines are necessarily returned as output from Python’s interpreter.

We can import Abjad into Python with the following incantation:

```
| >>> import abjad
```

Abjad is implemented as a collection of over 900 public classes and functions spread across nearly 40 subpackages, totally almost 200,000 lines of source code. Many of these classes and functions are available immediately within Abjad’s root *namespace*.<sup>1</sup>

We can enumerate these “top-level” names by calling Python’s built-in `dir()` function on the imported Abjad module object. After stripping out all “private”<sup>2</sup> names – those starting with underscores – Abjad’s top-level names can be printed to Python’s interactive console:

```
>>> abjad_names = dir(abjad)
>>> abjad_names = [x for x in abjad_names if not x.startswith('_')]
>>> print(abjad_names)
['Accelerando', 'Articulation', 'Beam', 'Chord', 'Clef', 'Container', 'Context', 'Crescendo',
'Decrescendo', 'Duration', 'Dynamic', 'Fermata', 'Fraction', 'Glissando', 'Hairpin', 'KeySignature',
'Markup', 'Measure', 'MultimeasureRest', 'Multiplier', 'NamedPitch', 'Note', 'Offset', 'Rest',
'Ritardando', 'Score', 'Sequence', 'Skip', 'Slur', 'Staff', 'StaffGroup', 'Tempo', 'Tie',
'TimeSignature', 'Timespan', 'Tuplet', 'Voice', 'abctools', 'abjad_configuration', 'abjadbooktools',
'agenttools', 'attach', 'datastructuretools', 'detach', 'developerscripttools', 'documentationtools',
'durationtools', 'exceptiontools', 'f', 'graph', 'handlertools', 'indicationtools', 'inspect_',
'instrumenttools', 'ipythontools', 'iterate', 'labeltools', 'layouttools', 'lilypondfiletools',
'lilypondnametools', 'lilypondparsers', 'ly', 'markuptools', 'mathtools', 'metertools', 'mutate',
'new', 'override', 'parse', 'persist', 'pitchtools', 'play', 'quantizationtools', 'rhythmmakertools',
'rhythmtools', 'schemetools', 'scoretools', 'select', 'selectiontools', 'selectortools',
'sequencetools', 'set_', 'show', 'sievetools', 'spannertools', 'stringtools', 'systemtools',
'templatetools', 'timespantools', 'tonalanalysistools', 'topleveltools']
```

Names beginning with uppercase letters represent classes. These include many of the musical objects composers would likely create by hand. Most of these classes have direct notational analogs on the page – e.g. `Note`, `Rest` and `Chord` –, or map to common structural or typographic concepts in Abjad’s primary typesetting engine, LilyPond, such as `Voice`, `Markup` and `Skip`. Lowercase names ending in the string “tools” represent additional libraries or

---

<sup>1</sup>In programming, *namespaces* represents contexts in which a particular collection of *names*, or identifiers, are grouped together, with each name referencing some code object, such as a class, function or number. Python makes extensive use of namespaces, treating every module – any file containing code – as a namespace, as well as any class, the bodies of functions and so forth. Python’s namespaces are implemented as dictionaries, a type of data structure which maps a unique set of keys to values. Each name in a Python namespace therefore appears only, although multiple names may be bound to the same value.

<sup>2</sup>Unlike many other languages, such as C++ or Java, Python does not enforce the concept of “public” versus “private” objects at the language level. Any object in any namespace can be accessed by any actor at any time. Instead, Python programmers use naming conventions to indicate which objects represent components of public interfaces, and which objects should be considered private, although still accessible, implementation details.

*subpackages* within Abjad. These subpackages group together related functionality within a common namespace. For example, Abjad’s `pitchtools` subpackage contains dozens of classes and functions solely for modeling pitch objects, collections of those objects, and their transformations:

```
>>> pitchtools_names = dir(abjad.pitchtools)
>>> pitchtools_names = [x for x in pitchtools_names if not x.startswith('_')]
>>> print(pitchtools_names)
[''Accidental'', 'Interval', 'IntervalClass', 'IntervalClassSegment', 'IntervalClassSet',
'IntervalClassVector', 'IntervalSegment', 'IntervalSet', 'IntervalVector', 'Inversion', 'Multiplication',
'NamedInterval', 'NamedIntervalClass', 'NamedInversionEquivalentIntervalClass', 'NamedPitch',
'NamedPitchClass', 'NumberedInterval', 'NumberedIntervalClass',
'NumberedInversionEquivalentIntervalClass', 'NumberedPitch', 'NumberedPitchClass',
'NumberedPitchClassColorMap', 'Octave', 'Pitch', 'PitchArray', 'PitchArrayCell', 'PitchArrayColumn',
'PitchArrayInventory', 'PitchArrayRow', 'PitchClass', 'PitchClassSegment', 'PitchClassSet',
'PitchClassTree', 'PitchClassVector', 'PitchOperation', 'PitchRange', 'PitchRangeInventory',
'PitchSegment', 'PitchSet', 'PitchVector', 'Registration', 'RegistrationComponent',
'RegistrationInventory', 'Retrogression', 'Rotation', 'Segment', 'Set', 'StaffPosition', 'Transposition',
'TwelveToneRow', 'Vector', 'instantiate_pitch_and_interval_test_collection',
'inventory_inversion_equivalent_named_interval_classes',
'inventory_named_inversion_equivalent_interval_classes', 'iterate_named_pitch_pairs_in_expr',
'list_named_pitches_in_expr', 'list_numbered_interval_numbers_pairwise',
'list_numbered_inversion_equivalent_interval_classes_pairwise',
'list_ordered_named_pitch_pairs_from_expr_1_to_expr_2', 'list_pitch_numbers_in_expr',
'list_unordered_named_pitch_pairs_in_expr', 'set_written_pitch_of_pitched_components_in_expr',
'sort_named_pitch_carriers_in_expr', 'transpose_named_pitch_by_numbered_interval_and_respell',
'transpose_pitch_carrier_by_interval', 'transpose_pitch_class_number_to_pitch_number_neighbor',
'transpose_pitch_expr_into_pitch_range', 'transpose_pitch_number_by_octave_transposition_mapping',
'yield_all_pitch_class_sets']
```

Many of these subpackages implement opinionated collections of compositional devices, certainly useful for some composers but by no means considered core to Abjad’s notational model. For example, Abjad’s `quantizationtools` provides a rhythmic quantizer inspired by Paul Nauert’s concept of “Q-grids”<sup>13</sup>, `rhythmtree` tools parses a Lisp-like “RTM-syntax” – which should be familiar to anyone working with OpenMusic<sup>2</sup>, PWGL<sup>10,9</sup> or Bach<sup>1</sup> – into object-oriented rhythm-tree data structures, and the `sievetools` subpackage models Iannis Xenakis’ pitch sieves<sup>21</sup>. Still other subpackages implement more “objective” functionality, such as `mathtools` or `durationtools`. Many of the remaining subpackages collect together related musical and typographic classes and functions, e.g. `scoretools`, `spannertools`, `markuptools`, and `indicatortools`. Others simply exist for “internal” use by Abjad’s developers, assisting in the development and maintenance of the system, such as `abctools`, `developer`, `scripttools`, `documentation` tools and `system` tools.

The remaining names in Abjad’s namespace represent its “top-level” functions, including `attach()`, `detach()`, `inspect_()`, `iterate()`, `mutate()`, `new()`, `override()`, `parse()`, `persist()`, `play()`, `select()`, and – most impor-

tantly – `show()`. These functions provide a powerful set of tools for interacting with Abjad’s notational objects and factories, and will be explained in detail as we encounter each of them.

Finally, simply for the sake of brevity, this chapter – and each subsequent chapter – will behave as though the contents of Abjad’s namespace has been imported into Python’s global namespace:

```
| >>> from abjad import *
```

## 2.1 REPRESENTING OBJECTS

In Python, everything is an object, even integers, boolean values and functions. Because Python is also interpreted and can be used interactively, those objects must be representable in an interpreter session, which is necessarily textual. An object’s textual, or *string*, representation can take a variety of forms, and that same object may even specify explicitly how it should be represented textually, just as it might specify how it should behave in relation to such operations as addition, multiplication, or iteration.

Python classes often specify their behavior in terms of *protocols*, sets of methods which explain the kinds of behaviors a class is able to engage in. This explanation takes place when a class implements or overrides “dunder” methods – double-underscore methods – such as `__repr__()` which provides an object’s *interpreter representation* in cooperation with Python’s built-in `repr()` function, and `__str__()` which provides an object’s string representation in cooperation with Python’s built-in `str()` function. By overriding these dunder methods, a class may specify its own version of string or interpreter representation behavior. Like public and private names, Python protocols are also established by convention rather than enforced at the language level, in contrast to similar mechanisms in other languages such as Java’s interfaces.

For example, a simple `Foo` class, which overrides neither `__str__()` nor `__repr__()` may be defined and instantiated:

```
| >>> class Foo(object):
| ...     pass
| ...
| >>> foo = Foo()
```

When printed to the terminal, cast as a string, and represented, Python’s default object representation is used, giving the module and name of the class as well as its memory location. Note that when calling Python’s `print()` function on an object, that object is converted to a string and then printed to the console, resulting in text without quotation marks:

```

>>> print(foo)
<__main__.Foo object at 0x106775d90>

>>> repr(foo)
'<__main__.Foo object at 0x106775d90>'

>>> str(foo)
'<__main__.Foo object at 0x106775d90>'

```

Defining a new class, `Foo2`, which overrides both string and interpreter representation provides customized output.

Note that printing the `Foo2` instance relies on its string representation, rather than its interpreter representation, while simply referencing the instantiated `Foo2` object uses its interpreter representation – its “`repr`”:

```

>>> class Foo2(object):
...     def __repr__(self):
...         return '<I am Foo2>'
...     def __str__(self):
...         return 'foofoo'
...
>>> foo2 = Foo2()
>>> foo2
<I am Foo2>

>>> print(foo2)
foofoo

>>> repr(foo2)
'<I am Foo2>'

>>> str(foo2)
'foofoo'

```

Defining a third version of `Foo` with an overridden `__format__()` method allows for creating alternate string representations. By calling Python’s built-in `format()` function on a `Foo3` instance, either the normal string representation or a “special” representation can be created. This formatting behavior can also be extended to support an arbitrary number of format specifications:

```

>>> class Foo3(object):
...     def __format__(self, format_specification=''):
...         if format_specification == 'special':
...             return '~~~ special foo format ~~~~'
...         return str(self)
...     def __repr__(self):
...         return '<I am Foo3>'
...     def __str__(self):
...         return 'fooooofoo'
...
>>> foo3 = Foo3()
>>> foo3
<I am Foo3>

```

```

>>> print(foo3)
foofoofoo

>>> repr(foo3)
'<I am Foo3>'

>>> str(foo3)
'foofoofoo'

>>> format(foo3)
'foofoofoo'

>>> format(foo3, 'special')
'~~~ special foo format ~~~~',

```

Many Abjad objects have multiple possible string representations. For example, a `Note` object can be represented by its normal interpreter representation, as a LilyPond string, or by its *storage format* – a more verbose, potentially multi-line string representation generally used when persisting complex objects to disk.

```

>>> note = Note(NamedPitch("g'"), Duration(3, 4))
>>> note
Note("g'2.")

>>> print(repr(note))
Note("g'2.")

>>> print(note)
g'2.

>>> print(str(note))
g'2.

>>> print(format(note))
g'2.

>>> print(format(note, 'lilypond'))
g'2.

>>> print(format(note, 'storage'))
scoretools.Note("g'2.")

```

For some Abjad classes – especially those representing glyphs in music notation – their default string format is their LilyPond string representation. Other classes – especially Abjad’s various highly-configurable notation factory classes – default their string format to their storage format, allowing for complex but still human-readable interpreter output. The idiom `print(format(...))` is used extensively throughout this document – especially in later chapters – to quickly display these multi-line storage formats:

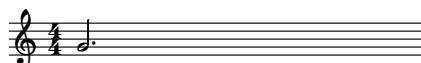
```

>>> rhythm_maker = rythmmakertools.TaleaRhythmMaker(
...     talea=rythmmakertools.Talea([1, 2, 3], 16),
...     extra_counts_per_division=(1, 0, 2, 1, 0),
...     output_masks=[
...         rythmmakertools.SustainMask([1], 3),
...         rythmmakertools.SilenceMask([-1]),
...         rythmmakertools.NullMask([0]),
...     ],
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tie_across_divisions=True,
...     ),
... )
>>> print(format(rhythm_maker))
rythmmakertools.TaleaRhythmMaker(
    talea=rythmmakertools.Talea(
        counts=(1, 2, 3),
        denominator=16,
    ),
    extra_counts_per_division=(1, 0, 2, 1, 0),
    output_masks=rhythmmakertools.BooleanPatternInventory(
        (
            rhythmmakertools.SustainMask(
                indices=(1,),
                invert=True,
            ),
            rhythmmakertools.SilenceMask(
                indices=(-1,),
            ),
            rhythmmakertools.NullMask(
                indices=(0,),
            ),
        )
    ),
    tieSpecifier=rhythmmakertools.TieSpecifier(
        tie_across_divisions=True,
    ),
)

```

Finally, in the spirit of Python’s protocols and dunder conventions, Abjad provides its own *illustration protocol* and corresponding `__illustrate__()` dunder method. This method works with Abjad’s top-level `show()` function to generate graphic realizations of illustrable Python objects by automatically formatting those objects into LilyPond syntax and passing the resulting input to LilyPond for typesetting. Abjad’s `show()` function will be used throughout this document to create and display music notation and other graphic representations of the objects discussed here:

```
>>> show(note)
```



## 2.2 COMPONENTS

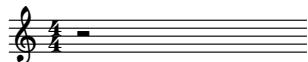
Abjad models notated musical scores as a *tree* consisting of *components*, *indicators* and *spanners*. All objects appearing in a score fall into one of those three categories. The most important and most complex of the three groups, components, make up the *nodes* of the score tree, treating the score as an acyclic directed rooted graph, or *arborescence*, rooted on a single component. All components inherit from Abjad's Component class, which encapsulates logic crucial to maintaining the correctness of the score tree, and affords behaviors common to all component subclasses including illustration, formatting and naming. Abjad divides score components into *containers* – those components which may contain other other components, such as staves, voices, tuplets, measures and so forth – and *leaves* – those components which may contain no other components, such as notes, chords and rests, as well as LilyPond's multi-measure rests and non-printing typographic *skips*. Additionally, components can each be instantiated from a variety of input, from parsable LilyPond syntax strings to parametric arguments – both numeric and explicitly object-modeled via pitch and duration objects. For example, we can instantiate and illustrate a middle-C quarter note:

```
>>> note = Note("c' 4")
>>> show(note)
```



a half-note rest:

```
>>> rest = Rest((1, 2))
>>> show(rest)
```



and a D-minor chord:

```
>>> chord = Chord(
...     [NamedPitch("d'"), NamedPitch("f'"), NamedPitch("a'")],
...     Duration(1, 4),
... )
>>> show(chord)
```



Each of the above three leaf instantiations demonstrates a different means of creating score objects, none of which is unique to the particular class they have been demonstrated with here. The Note instantiation demonstrates creating a score object via LilyPond syntax, the rest instantiation demonstrates creating a score object from sequences of integers – in this case the pair (1, 2), representing the duration of a half-note – and the chord instantiation demonstrates using explicit NamedPitch and Duration objects. The latter two instantiation techniques afford parametric approaches, while the LilyPond syntax allows for a great degree of expressivity as score components can be created with articulations, dynamics and other indicators already attached:

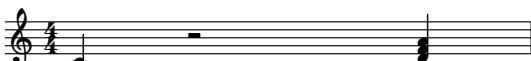
```
>>> fancy_chord = Chord(r"<g' bf'? d'>4. -\accent -\staccato ^\markup{ \italic gently }")
>>> show(fancy_chord)
```



## 2.2.1 CONTAINERS

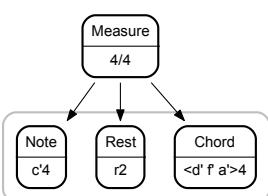
Abjad's container classes – measures, tuples, voice, staves, staff groups, scores and so forth – implement Python's *mutable sequence protocol*, allowing them to be used transparently as though they were lists. Components can be appended, extended, or inserted into containers. Containers can also be instantiated with lists of components to be inserted as one of their instantiation arguments. Note in the previous notation examples that the LilyPond output always shows a  $\frac{4}{4}$  time signature even though none of the created leaves takes up a whole note's duration. LilyPond simply treats all music as  $\frac{4}{4}$  unless explicitly instructed otherwise. We can make the  $\frac{4}{4}$  time signature explicit by instantiating a  $\frac{4}{4}$  measure object to contain the three previously-created leaf instances, with the three leaves passed in a list as an instantiation argument to the new measure:

```
>>> measure = Measure((4, 4), [note, rest, chord])
>>> show(measure)
```



The measure can also be represented as a graph, clarifying the containment:

```
>>> graph(measure)
```



Like lists, containers can be indexed into, measured for length and iterated over. For example, the second component contained by the measure can be selected by subscripting the measure with the index 1:<sup>3</sup>

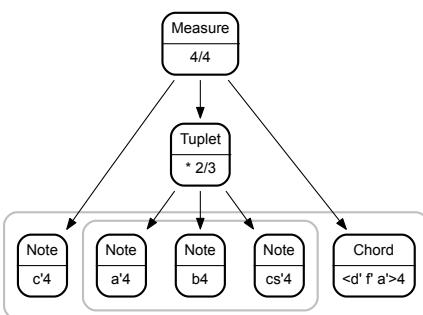
```
>>> measure[1]
Rest('r2')
```

That second component – the half-note rest – can be replaced by another container, a triplet:

```
>>> outer_tuplet = Tuplet((2, 3), "a'4 b4 cs'4")
>>> measure[1] = outer_tuplet
>>> show(measure)
```



```
>>> graph(measure)
```



Furthermore, the last two leaves in the new triplet can be replaced by a quintuplet using Python's *slice assignment* syntax. In the following code, the text `outer_tuplet[1:]` references the selection of components in the original triplet starting from the second component and going “to the end” – effectively the second and third item, as there are only three. That selection is then replaced by a list containing a quintuplet, substituting the contents of one sequence of components for the contents of another.

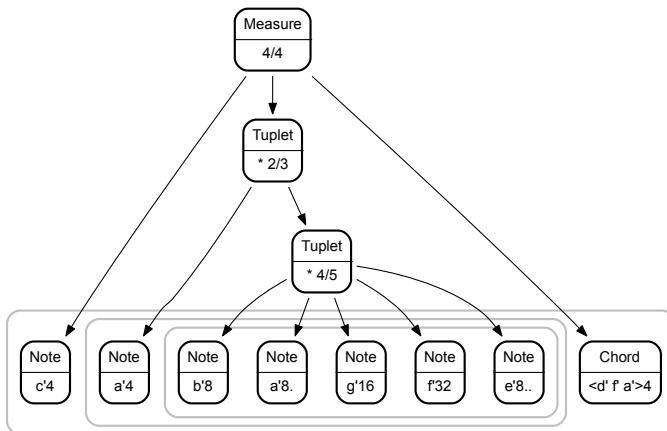
```
>>> inner_tuplet = Tuplet((4, 5), "b'8 a'8. g'16 f'32 e'8..")
>>> outer_tuplet[1:] = [inner_tuplet]
>>> show(measure)
```




---

<sup>3</sup>Like most programming languages, Python counts from zero. Therefore the “first” item in an ordered collection is indexed by the number zero, not one.

```
>>> graph(measure)
```



The measure can be inspected for its length by using Python's built-in `len()` function. Note that this returns the number of components *immediately* contained by the measure – 3 –, but not the total number of components, total number of leaves or total duration. The latter queries can be satisfied by other means.

```
>>> len(measure)
3
```

Iterating over the contents of the  $\frac{4}{4}$  measure yields only the top-most components in that container – its immediate children:

```
>>> for component in measure:
...     component
...
Note("c'4")
Tuplet(Multiplier(2, 3), "a'4 { 4/5 b'8 a'8. g'16 f'32 e'8... }")
Chord("<d' f' a'>4")
```

To select all of the leaves of a container, recursion must be used, as any container may contain other containers, and those in turn still more containers. To mitigate this complexity, every container provides a `select_leaves()` method, which returns a selection of the bottom-most leaf instances in the subtree rooted at that container:

```
>>> for leaf in measure.select_leaves():
...     leaf
...
Note("c'4")
Note("a'4")
Note("b'8")
Note("a'8..")
Note("g'16")
Note("f'32")
Note("e'8...")
Chord("<d' f' a'>4")
```

Note that Abjad's Chord class aggregates multiple *note heads* rather than notes. Likewise Abjad's Note class aggregates a single note head. While chords implement containment in terms of note heads – delegating to a dedicated note head inventory – they are not themselves "containers" in the same sense that a voice, measure or tuplet are containers. It is important here to differentiate between the concept of a "chord" as a single duration paired with a collection of pitches and a "chord" as the sounding sonority at some given point in a piece spread over some number of voices. Abjad always makes use of the former rather than the latter.

```
>>> note_head_inventory = chord.note_heads
>>> for note_head in note_head_inventory:
...     note_head
...
NoteHead("d")
NoteHead("f")
NoteHead("a")

>>> chord.note_heads.append(NamedPitch("c"))
>>> show(chord)
```



## 2.2.2 PARENTAGE

The container that contains a given component is called its *parent*, and the components contained in a container are known as that container's *children*. Every component may have one and only one parent container, although a component may also have no parent – a null parent reference. Any component whose parent is null is necessarily the root of its own score tree. Furthermore, no component may appear in its own *proper parentage* – the sequence of components comprising the unique path from a given component to the root of its tree, excepting the component itself – as this would induce reference cycles within the score tree.

Abjad object-models the concept of parentage explicitly as a Parentage *selection class*, accessible via the *component inspector* which exposes a given component's *inspection interface*: a collection of methods for accessing information about that component, many of which depend on that component's position within the score hierarchy, including the component's parentage or duration. An inspector can be instantiated by calling Abjad's top-level inspect\_() function on a component.<sup>4</sup> Consider the parentage for the first eighth note of the inner quintuplet in the previously

---

<sup>4</sup> Abjad uses inspect\_() rather than inspect() as Python already provides an inspect module for object introspection, and it is considered bad practice to overwrite names of functions, classes or modules found in the standard library such as set or object. When a name conflict would occur, projects traditionally append an underscore to the name they wish to use.

created measure. This eighth-note's parentage – *improper* by default – includes itself, its immediate quintuplet parent, its triplet grandparent and the  $\frac{4}{4}$  measure as its great-grandparent:<sup>5</sup>

```
>>> inner_b_eighth = measure[1][1][0]
>>> inspector = inspect_(inner_b_eighth)
>>> parentage = inspector.get_parentage()
>>> parentage.parent
Tuplet(Multiplier(4, 5), "b'8 a'8. g'16 f'32 e'8..")

>>> for component in parentage:
...     component
...
Note("b'8")
Tuplet(Multiplier(4, 5), "b'8 a'8. g'16 f'32 e'8..")
Tuplet(Multiplier(2, 3), "a'4 { 4/5 b'8 a'8. g'16 f'32 e'8.. }")
Measure((4, 4), "c'4 { 2/3 a'4 { 4/5 b'8 a'8. g'16 f'32 e'8.. } } <d' f' a' c'>4")
```

By default, every component's parent is null, represented in Python by the `None` object:

```
>>> whole_note = Note("c'1")
>>> inspect_(whole_note).get_parentage().parent is None
True
```

Inserting the above whole note into a container sets the whole note's parent to that container. Additionally, the container is now aware that it contains the whole note, demonstrating the bi-directional references inherent to Abjad's score tree model:

```
>>> container_one = Container()
>>> container_one.append(whole_note)
>>> inspect_(whole_note).get_parentage().parent is container_one
True

>>> whole_note in container_one
True

>>> print(format(container_one))
{
    c'1
}
```

Inserting the same whole note into a different container removes it from the first, demonstrating that components can only exist in one container at a time:

---

<sup>5</sup>The component summary strings in the interpreter representations of the components in the parentage are *not* valid LilyPond syntax, but simply a concise LilyPond-like syntax.

```

>>> container_two = Container()
>>> container_two.append(whole_note)
>>> inspect_(whole_note).get_parentage().parent is container_two
True

>>> whole_note in container_two
True

>>> whole_note in container_one
False

>>> print(format(container_two))
{
    c'1
}

>>> print(format(container_one))
{
}

```

Likewise, while the first container can be inserted into the second, the second container cannot be inserted back into the first. Such an arrangement would cause the second container to become its own grandparent, and therefore generates an error:

```

>>> container_two.append(container_one)
>>> container_one.append(container_two)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/josiah/Development/abjad/abjad/tools/scoretools/Container.py", line 1062, in append
    self._setitem__(slice(len(self), len(self)), [component])
  File "/Users/josiah/Development/abjad/abjad/tools/scoretools/Container.py", line 218, in __setitem__
    return self._set_item(i, expr)
  File "/Users/josiah/Development/abjad/abjad/tools/scoretools/Container.py", line 561, in _set_item
    raise ParentageError('Attempted to induce cycles.')
ParentageError: Attempted to induce cycles.

```

Why should we be concerned with reference cycles? When we create score by hand, notes – no matter how we may have arrived at them or conceived of them creatively – are ultimately simply black marks on a page. However, when working with computers, we can choose how to model musical objects. Again, a “note” could simply exist as an amalgam of dots and lines in a vector-graphics program, in which case it may not be possible to know where or even what that note or any other note *is* except in terms of dots and lines – the affordances provided by that vector-graphics editor. In that case, there is no semantic model of music, only a typographic model devoid of explicit musical relationships. Alternatively, we can model explicitly, in which case notes exist as “objects” in a rich network of references. In this latter case, certain conditions must be maintained to protect the integrity of the reference

network. Depending on the implementation, a note likely cannot be in two places at once. That is the case in Abjad’s model of musical notation: notes can only have one parent, and cannot exist in more than one parent. Note that these restrictions mainly derive from the fact that notes, like other scores components, are both *mutable* – that is, changeable after they have been instantiated, and therefore *stateful* – and possess many references to objects “outside” of themselves. In contrast, Abjad’s pitch and duration classes have no state – they are *immutable* – and reference no other objects. They can therefore appear in many places within the same score. Arbitrarily many score components can reference the same duration in memory, much as arbitrarily many objects can reference the same integer. Such an arrangement is often referred to as a *flyweight*<sup>8</sup>.

### 2.2.3 DURATIONS

Every Abjad score component may be expressed in terms of its duration, as well as its start offset relative to the score origin and, by extension, its timespan – the span of time bounded by its start offset and stop offset. Abjad models all such duration objects as *rationals*: ratios of real numbers, e.g.  $\frac{1}{2}$ ,  $\frac{7}{23}$  or  $\frac{16}{1}$ . This concern arises from the realization that all durations and offsets expressible in Western common practice notation are rational, rather than floating-point or any other representation. Necessarily, Abjad’s Duration, Offset and Multiplier classes all derive from Python’s fractions.Fraction class. Both Offset and Multiplier are little more than aliases to Abjad’s Duration class. However, their use throughout Abjad’s code-base, and those projects heavily dependent upon Abjad, such as Consort, greatly clarify compositional intent and increase the source’s legibility.

Abjad’s Duration class extends Python’s Fraction class with a number of new initialization patterns and a variety of notation-specific properties. For example, a  $\frac{7}{16}$  duration can be instantiated from a numerator/denominator pair, or from a LilyPond syntax duration string:

```
>>> Duration(7, 16)
Duration(7, 16)

>>> Duration.from_lilypond_duration_string('4..')
Duration(7, 16)
```

Duration objects can provide information about how they should be interpreted notationally, including how many dots or flags they would have if used to instantiate a leaf. A  $\frac{7}{16}$  duration, without resorting to tuplets or ties, might be represented in notation by a double-dotted quarter note, thus giving a dot count of 2 and a flag count of 0:

```
>>> Duration(7, 16).dot_count
2
```

```

>>> Duration(7, 16).flag_count
0

>>> Duration(7, 16).lilypond_duration_string
'4..'

```

As mentioned in subsection 2.2.2, duration are immutable. Like other immutable objects in Python – e.g. integers and string –, they raise errors when anything attempts to alter them in anyway:

```

>>> string = 'abcdefghijkl'
>>> string[1:-1] = 'xyz'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment

>>> Duration(3, 4).numerator = 7
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute

>>> Duration(1, 2).nonexistant_property = 'This will never work'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Duration' object has no attribute 'nonexistant_property'

```

Additionally, some mathematical operations between offsets and durations provide results typed in a musically-sensible fashion. For example, the difference between two offsets returns a duration, while a duration added to an offset returns another offset:

```

>>> offset_one = Offset(1, 2)
>>> offset_two = Offset(7, 4)
>>> offset_two - offset_one
Duration(5, 4)

>>> offset_two + Duration(3, 8)
Offset(17, 8)

```

The duration of each leaf derives from the product of its *written duration* – a duration representing the actual glyphs used in the score as represented by some combination of note heads, stems, beams and dots – and their *prolation* – the cumulative product of all of the duration multipliers of the containers in a component’s proper parentage.

Consider again the measure created earlier in subsection 2.2.1:

```

>>> show(measure)

```



The B eighth-note starting the quintuplet is nested within two different tuplets with different multipliers. While the eighth-note's parentage object can certainly calculate its prolation automatically, we will calculate the prolation here "by hand" in order to demonstrate the technique:

```
>>> parentage = inspect_(inner_b_eighth).get_parentage()
>>> parentage.prolation
Multiplier(8, 15)

>>> by_hand_prolation = 1
>>> for parent in parentage[1:]:
...     if isinstance(parent, Tuplet):
...         by_hand_prolation = by_hand_prolation * parent.multiplier
...
>>> by_hand_prolation
Multiplier(8, 15)
```

By multiplying each leaf's written duration by its prolation, we can determine that leaf's actual, or *prolated*, duration.

Note too that this observation conforms to the results of the inspector's `get_duration()` method:

```
>>> for leaf in measure.select_leaves():
...     inspector = inspect_(leaf)
...     written_duration = leaf.written_duration
...     prolation = inspector.get_parentage().prolation
...     actual_duration = inspector.get_duration()
...     string = '{!r}: {!s} * {!s} = {!s}'
...     string = string.format(leaf, written_duration, prolation, actual_duration)
...     print(string)
...
Note("c'4"): 1/4 * 1 = 1/4
Note("a'4"): 1/4 * 2/3 = 1/6
Note("b'8"): 1/8 * 8/15 = 1/15
Note("a'8."): 3/16 * 8/15 = 1/10
Note("g'16"): 1/16 * 8/15 = 1/30
Note("f'32"): 1/32 * 8/15 = 1/60
Note("e'8.."): 7/32 * 8/15 = 7/60
Chord("<d' f' a' c' >4"): 1/4 * 1 = 1/4
```

Written durations must be *assignable*. Assignability describes the set of durations describable in Western common practice notation solely through combining a single note head, its flags and dots, without recourse to ties or tuplets. Any rational duration  $n/d$  is considered assignable when and only when it adheres to the form

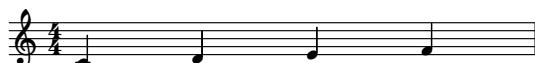
$$\frac{2^k * (2^u - j)}{2^v} \quad (2.1)$$

where  $u$ ,  $v$  and  $k$  are nonnegative integers,  $j \leq u$  and  $j$  is either 1 or 0. Assignability guarantees that a duration's denominator is always a positive power-of-two integer, such as 1, 2, 4, 8, 16 and so forth, and therefore precludes durations such as  $\frac{1}{3}$  or  $\frac{2}{5}$ . Likewise, assignability permits numerators such as 1, 2, 3, 4, 6, 7, 8, 12, 14 and 15 but forbids 5, 9, 10, 13 and 17, as they imply ties. More elegantly, any integer is considered assignable if its binary representation – disregarding any leading zeros – does not contain the substring “01”:

```
>>> for i in range(17):
...     binary_string = mathtools.integer_to_binary_string(i)
...     is_assignable = mathtools.is_assignable_integer(i)
...     string = '{}: {} [{}], {}'.format(i, binary_string, is_assignable)
...     print(string)
...
0: 0 [False]
1: 1 [True]
2: 10 [True]
3: 11 [True]
4: 100 [True]
5: 101 [False]
6: 110 [True]
7: 111 [True]
8: 1000 [True]
9: 1001 [False]
10: 1010 [False]
11: 1011 [False]
12: 1100 [True]
13: 1101 [False]
14: 1110 [True]
15: 1111 [True]
16: 10000 [True]
```

The duration of each container then derives from the product of its prolation and its *contents duration* – the sum of the durations of its children. Ultimately, all scores derive their durations from the durations of their leaves, prolated as necessary by any tuplets. Recall that containers are mutable. As components are added to and removed from a container, the duration of that container and the offsets of components following the inserted or deleted component adjust dynamically to reflect the altered structure:

```
>>> staff = Staff("c'4 d'4 e'4 f'4")
>>> show(staff)
```



```
>>> inspect_(staff).get_duration()
Duration(1, 1)
```

```

>>> for leaf in staff:
...     offset = inspect_(leaf).get_timespan().start_offset
...     print(offset, leaf)
...
(Offset(0, 1), Note("c'4"))
(Offset(1, 4), Note("d'4"))
(Offset(1, 2), Note("e'4"))
(Offset(3, 4), Note("f'4"))

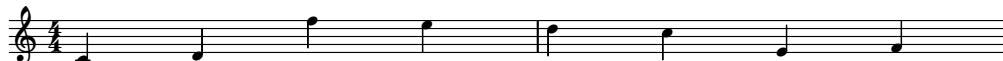
```

After inserting an additional four quarter-notes into the staff, the staff reports its duration as doubled, and all of the leaves – both new and old – report their expected start offsets:

```

>>> staff[2:2] = "f''4 e''4 d''4 c''4"
>>> show(staff)

```



```

>>> inspect_(staff).get_duration()
Duration(2, 1)

>>> for leaf in staff:
...     offset = inspect_(leaf).get_timespan().start_offset
...     print(offset, leaf)
...
(Offset(0, 1), Note("c'4"))
(Offset(1, 4), Note("d'4"))
(Offset(1, 2), Note("f''4"))
(Offset(3, 4), Note("e''4"))
(Offset(1, 1), Note("d''4"))
(Offset(5, 4), Note("c''4"))
(Offset(3, 2), Note("e'4"))
(Offset(7, 4), Note("f'4"))

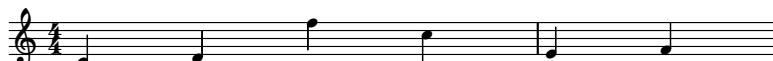
```

Likewise, removing the middle two of the previously inserted leaves results in the staff reporting a decreased duration, and all leaves updating their offsets to reflect the deletion:

```

>>> staff[3:5] = []
>>> show(staff)

```



```

>>> inspect_(staff).get_duration()
Duration(3, 2)

>>> for leaf in staff:
...     offset = inspect_(leaf).get_timespan().start_offset
...     print(offset, leaf)
(Offset(0, 1), Note("c'4"))

```

```
(Offset(1, 4), Note("d'4"))
(Offset(1, 2), Note("f'4"))
(Offset(3, 4), Note("c'4"))
(Offset(1, 1), Note("e'4"))
(Offset(5, 4), Note("f'4"))
```

Finally, multiplier objects may be attached to leaves to multiply their duration. When attached to multi-measure rests, not only does the overall duration of the leaf change, but LilyPond is able to generate typography representing multiple bars of tacet music compressed together:

```
>>> multimeasure_rest = scoretools.MultimeasureRest(1)
>>> inspect_(multimeasure_rest).get_duration()
Duration(1, 1)
```

```
>>> show(multimeasure_rest)
```



```
>>> attach(Multiplier(4), multimeasure_rest)
>>> inspect_(multimeasure_rest).get_duration()
Duration(4, 1)
```

```
>>> show(multimeasure_rest)
```



## 2.2.4 NAMED COMPONENTS

Abjad's scores components may be given unique names via their `name` property. Any named component found in the subtree of a given container may be accessed by subscripting the container with its name, regardless of its depth in that container.

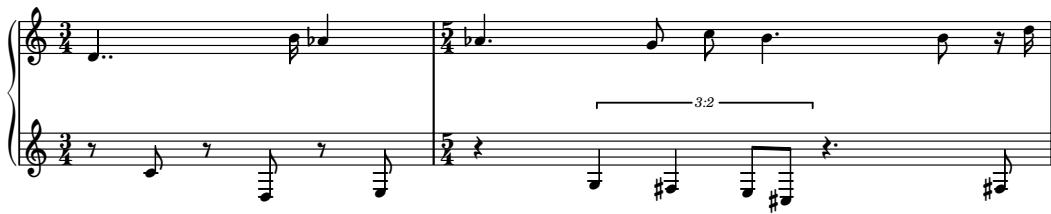
Consider the following score, containing a piano staff grouping two staves, each with one voice. Disregard the ugliness of the notation for the time being; it will be cleaned up in section 2.3 and section 2.4:

```
>>> voice_1 = Voice(name='Voice 1')
>>> voice_1.append(Measure((3, 4), "d'4.. b'16 af'4"))
>>> voice_1.append(Measure((5, 4), "af'4. g'8 c'8 b'4. b'8 r16 d'16"))
>>> upper_staff = Staff(
...     [voice_1],
...     name='Upper Staff',
... )
>>> voice_2 = Voice(name='Voice 2')
>>> voice_2.append(Measure((3, 4), "r8 c'8 r8 d8 r8 e8"))
>>> voice_2.append(Measure((5, 4), "r"r4 \times 2/3 { g4 fs4 e8 cs8 } r4. fs8"))
```

```

>>> lower_staff = Staff(
...     [voice_2],
...     name='Lower Staff',
... )
>>> staff_group = StaffGroup(
...     [upper_staff, lower_staff],
...     context_name='PianoStaff',
...     name='Both Staves',
... )
>>> score = Score([staff_group])
>>> show(score)

```



Printing the score as a LilyPond syntax string clearly shows the nested quality of the score hierarchy. Additionally, the previously provided names of some of the containers – e.g. “Voice 1” and “Lower Staff” – also appear in the LilyPond output:

```

>>> print(format(score))
\new Score <<
  \context PianoStaff = "Both Staves" <<
    \context Staff = "Upper Staff" {
      \context Voice = "Voice 1" {
        {
          \time 3/4
          d'4..
          b'16
          af'4
        }
        {
          \time 5/4
          af'4.
          g'8
          c'8
          b'4.
          b'8
          r16
          d''16
        }
      }
    }
  \context Staff = "Lower Staff" {
    \context Voice = "Voice 2" {
      {
        \time 3/4
        r8
      }
    }
  }

```

```

        c'8
        r8
        d8
        r8
        e8
    }
{
    \time 5/4
    r4
    \times 2/3 {
        g4
        fs4
        e8
        cs8
    }
    r4.
    fs8
}
}
>>
>>

```

Any of the named components within the score hierarchy can be selected by subscripting any container in their proper parentage with their name. For example, the voice container named “Voice 1” can be selected by subscripting the score object with its name, even though it is not immediately contained by the score but is in fact a “great-grandchild” of the score:

```

>>> score['Voice 1']
<Voice="Voice 1">{2}>

>>> score['Voice 1'] is voice_1
True

```

Note that the staff group definition above received both a `name` and a `context_name` keyword argument. While the various voices and staves in the above score all appear in the LilyPond syntax output as `\context Staff = "Upper Staff"` or `\context Voice = "Voice 2"`, the staff group appears instead as `\context PianoStaff = "Both Staves"`. Its “context name” has been substituted for where `StaffGroup` would normally appear, allowing Abjad to specify a different LilyPond *context* for the music it contains.

Some Abjad container classes correspond to LilyPond’s notion of *contexts*. These include voices, staves, staff groups and scores, but not measures, tuplets or “bare” containers. LilyPond uses contexts during typesetting to maintain various kinds of musical information hierarchically. For example, LilyPond’s `Staff` context maintains information about the accidentals that have appeared so far in any voice contained by that staff as well as the staff’s

current clef – all of which is necessarily local to a single staff –, while the Score context maintains more global information, such as the current tempo and measure number. LilyPond allow composers to define new contexts, and provides a number of specially-defined contexts, e.g. `ChoirStaff`, `TabVoice` and `FiguredBass`. While LilyPond’s contexts may be either named or anonymous, named contexts allow LilyPond to stitch together different sections of music into a single continuous score, allowing different segments of a work to be defined in different files and then concatenated together.

## 2.3 INDICATORS

Abjad’s indicators include any object attached to a single score component, such as articulations, textual markup, clefs, tempo or time signature indications. Unlike score components and spanners, indicators do not share a common base class. Instead, they are unified by the means by which they have been attached to components: Abjad’s top-level `attach()` function. Indicators are generally immutable, like integers and durations. Regardless, in being attached to a score component they do not receive a reference to that component, allowing the same indicator to be attached to many components. Abjad binds the indicator to the component via an `IndicatorExpression` object, which holds the necessary references to both the indicator and the component along with other important information about the behavior of the attachment.

Consider a simple four note staff. A single accent articulation can be attached to each note in the staff via `attach()`:

```
>>> staff = Staff("c'4 d'4 e'4 f'4")
>>> accent = Articulation('accent')
>>> for note in staff:
...     attach(accent, note)
...
>>> show(staff)
```



Once attached, indicators can be removed via the top-level `detach()` function. For example, the attached accents can be detached from the last three leaves of the above staff via the `detach()` function:

```
>>> for note in staff[1:]:
...     detach(accent, note)
...
(Articulation('accent'),)
(Articulation('accent'),)
(Articulation('accent'),)
```

```
>>> show(staff)
```



At this point, only the first note in the staff has anything attached. As with durations and parentage, we can use the component inspector to verify that this is true by testing each note for the existence of an indicator of the class `Articulation`:

```
>>> for note in staff:  
...     has_articulation = inspect_(note).has_indicator(Articulation)  
...     print(note, has_articulation)  
...  
(Note("c'4"), True)  
(Note("d'4"), False)  
(Note("e'4"), False)  
(Note("f'4"), False)
```

Likewise, we can use the component inspector to retrieve the attached articulation. The inspector provides two methods for retrieving indicators attached to a single component: `get_indicator()` and `get_indicators()`. The latter returns a tuple of zero or more indicators matching any supplied class prototype, while the former returns only one and raises an error if more or less than one indicator matching the supplied class prototype is attached. In the case of retrieving any attached articulations from the staff's first note – given that there is only one articulation attached –, both methods work perfectly.

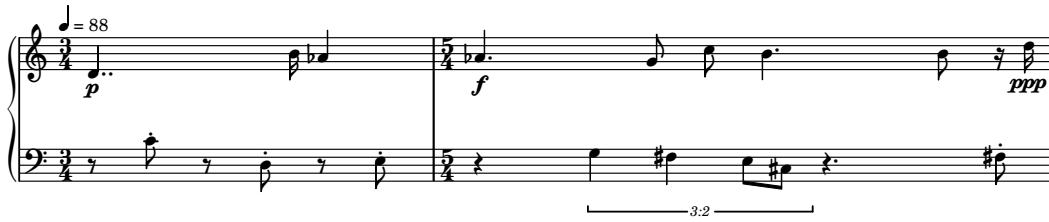
```
>>> inspect_(staff[0]).get_indicators(Articulation)  
(Articulation('accent'),)  
  
>>> inspect_(staff[0]).get_indicator(Articulation)  
Articulation('accent')
```

Consider again the two-staff score created earlier:

```
>>> show(score)
```

A variety of indicators can be attached to the leaves in this score to present a more convincing musical result, including clefs, dynamics, tempi and articulations. Note that indicators can be attached to both leaf and container components:

```
>>> attach(Clef('treble'), score['Upper Staff'])
>>> attach(Clef('bass'), score['Lower Staff'])
>>> lower_leaves = score['Lower Staff'].select_leaves()
>>> for i in [1, 3, 5, 12]:
...     attach(Articulation('staccato'), lower_leaves[i])
...
>>> attach(Tempo((1, 4), 88), score['Voice 1'][0][0])
>>> attach(Dynamic('p'), score['Voice 1'][0][0])
>>> attach(Dynamic('f'), score['Voice 1'][1][0])
>>> attach(Dynamic('ppp'), score['Voice 1'][1][-1])
>>> show(score)
```



### 2.3.1 INDICATOR SCOPE

Abjad models how the influence of certain types of indicators persists across components subsequent to the component they attach to via the concept of *indicator scope*. Indicator scope describes how, for example, all components in a score are governed by one tempo from the moment that tempo appears until the moment a different appears. Likewise, scoping describes how all leaves in a staff are understood to be governed by the staff's clef up until the point that that clef changes. Different indicators govern different scopes by default. As just described, tempo indications govern the score context, while clef, key signature and dynamics govern the staff context. An indicator governing some component is known as that component's *effective* indicator. As with non-scoped indicators, the component inspector can be used to examine if and what indicator of a given type is effective for a given component.

Consider the score above, to which a variety of indicators have just been attached. By inspecting the leaves in both staves with the component inspector's `get_effective()` method, we can determine what indicators are effective for each leaf. For example, all of the leaves in the upper staff will report that their clef is a treble clef, while all of the leaves in the lower staff will report that theirs is a bass clef:

```
>>> for leaf in score['Upper Staff'].select_leaves():
...     clef = inspect_(leaf).get_effective(Clef)
```

```

...     print(leaf, clef)
...
(Note("d'4.."), Clef(name='treble'))
(Note("b'16"), Clef(name='treble'))
(Note("af'4"), Clef(name='treble'))
(Note("af'4."), Clef(name='treble'))
(Note("g'8"), Clef(name='treble'))
(Note("c'8"), Clef(name='treble'))
(Note("b'4."), Clef(name='treble'))
(Note("b'8"), Clef(name='treble'))
(Rest('r16'), Clef(name='treble'))
(Note("d'16"), Clef(name='treble'))

>>> for leaf in score['Lower Staff'].select_leaves():
...     clef = inspect_(leaf).get_effective(Clef)
...     print(leaf, clef)
(Rest('r8'), Clef(name='bass'))
(Note("c'8"), Clef(name='bass'))
(Rest('r8'), Clef(name='bass'))
(Note('d8'), Clef(name='bass'))
(Rest('r8'), Clef(name='bass'))
(Note('e8'), Clef(name='bass'))
(Rest('r4'), Clef(name='bass'))
(Note('g4'), Clef(name='bass'))
(Note('fs4'), Clef(name='bass'))
(Note('e8'), Clef(name='bass'))
(Note('cs8'), Clef(name='bass'))
(Rest('r4.'), Clef(name='bass'))
(Note('fs8'), Clef(name='bass'))

```

While the tempo indication is only attached to the first leaf in the upper staff, all leaves in the entire score report that same tempo as their effective tempo:

```

>>> for leaf in score['Upper Staff'].select_leaves():
...     tempo = inspect_(leaf).get_effective(Tempo)
...     print(leaf, tempo)
...
(Note("d'4.."), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("b'16"), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("af'4"), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("af'4."), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("g'8"), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("c'8"), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("b'4."), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("b'8"), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Rest('r16'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("d'16"), Tempo(duration=Duration(1, 4), units_per_minute=88))

>>> for leaf in score['Lower Staff'].select_leaves():
...     tempo = inspect_(leaf).get_effective(Tempo)
...     print(leaf, tempo)
...

```

```
(Rest('r8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note("c'8"), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Rest('r8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('d8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Rest('r8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('e8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Rest('r4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('g4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('fs4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('e8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('cs8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Rest('r4.'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('fs8'), Tempo(duration=Duration(1, 4), units_per_minute=88))
```

However, the dynamic indications attached to the leaves in the upper staff are effective only for those leaves, and not for the leaves in the lower staff:

```
>>> for leaf in score['Upper Staff'].select_leaves():
...     dynamic = inspect_(leaf).get_effective(Dynamic)
...     print(leaf, dynamic)
...
(Note("d'4.."), Dynamic(name='p'))
(Note("b'16"), Dynamic(name='p'))
(Note("af'4"), Dynamic(name='p'))
(Note("af'4."), Dynamic(name='f'))
(Note("g'8"), Dynamic(name='f'))
(Note("c'8"), Dynamic(name='f'))
(Note("b'4."), Dynamic(name='f'))
(Note("b'8"), Dynamic(name='f'))
(Rest('r16'), Dynamic(name='f'))
(Note("d'16"), Dynamic(name='ppp'))

>>> for leaf in score['Lower Staff'].select_leaves():
...     dynamic = inspect_(leaf).get_effective(Dynamic)
...     print(leaf, dynamic)
...
(Rest('r8'), None)
(Note("c'8"), None)
(Rest('r8'), None)
(Note('d8'), None)
(Rest('r8'), None)
(Note('e8'), None)
(Rest('r4'), None)
(Note('g4'), None)
(Note('fs4'), None)
(Note('e8'), None)
(Note('cs8'), None)
(Rest('r4.'), None)
(Note('fs8'), None)
```

Both the clef, tempo and dynamics inspected above made use of their *default scope* when being attached, specifying implicitly that they be effective at either the staff or score context-level. Indicators can also be attached with an

explicitly-specified scope, overriding any default the indicator’s class might provide. By detaching the implicitly-staff-scoped dynamics, those same dynamic indications can be reattached, explicitly scoped for the “PianoStaff” context, allowing all leaves contained by that piano staff – which includes both the upper and lower staves – to detect their appropriate dynamic level:

```
>>> piano_dynamic = detach(Dynamic, score['Voice 1'][0][0])[0]
>>> forte_dynamic = detach(Dynamic, score['Voice 1'][1][0])[0]
>>> pianississimo_dynamic = detach(Dynamic, score['Voice 1'][1][-1])[0]
>>> attach(piano_dynamic, score['Voice 1'][0][0], scope='PianoStaff')
>>> attach(forte_dynamic, score['Voice 1'][1][0], scope='PianoStaff')
>>> attach(pianississimo_dynamic, score['Voice 1'][1][-1], scope='PianoStaff')
>>> for leaf in score['Upper Staff'].select_leaves():
...     dynamic = inspect_(leaf).get_effective(Dynamic)
...     print(leaf, dynamic)
...
(Note("d'4.."), Dynamic(name='p'))
(Note("b'16"), Dynamic(name='p'))
(Note("af'4"), Dynamic(name='p'))
(Note("af'4."), Dynamic(name='f'))
(Note("g'8"), Dynamic(name='f'))
(Note("c'8"), Dynamic(name='f'))
(Note("b'4."), Dynamic(name='f'))
(Note("b'8"), Dynamic(name='f'))
(Rest('r16'), Dynamic(name='f'))
(Note("d'16"), Dynamic(name='ppp'))

>>> for leaf in score['Lower Staff'].select_leaves():
...     dynamic = inspect_(leaf).get_effective(Dynamic)
...     print(leaf, dynamic)
...
(Rest('r8'), Dynamic(name='p'))
(Note("c'8"), Dynamic(name='p'))
(Rest('r8'), Dynamic(name='p'))
(Note('d8'), Dynamic(name='p'))
(Rest('r8'), Dynamic(name='p'))
(Note('e8'), Dynamic(name='p'))
(Rest('r4'), Dynamic(name='f'))
(Note('g4'), Dynamic(name='f'))
(Note('fs4'), Dynamic(name='f'))
(Note('e8'), Dynamic(name='f'))
(Note('cs8'), Dynamic(name='f'))
(Rest('r4.'), Dynamic(name='f'))
(Note('fs8'), Dynamic(name='f'))
```

Recall the voices in the score were populated by Abjad Measure objects. While LilyPond’s musical model does not make use of explicit measures, Abjad provides measure-like containers as a convenience. When instantiated, Abjad measures attach the appropriate time signature indication to themselves. These indicators are also scoped by default, allowing all of the leaves contained in that measure to detect their appropriate time signature:

```

>>> measure_1 = score['Voice 1'][0]
>>> measure_2 = score['Voice 1'][1]
>>> inspect_(measure_1).get_indicator(TimeSignature)
TimeSignature((3, 4))

>>> for leaf in measure_1.select_leaves():
...     time_signature = inspect_(leaf).get_effective(TimeSignature)
...     print(leaf, time_signature)
...
(Note("d'4.."), TimeSignature((3, 4)))
(Note("b'16"), TimeSignature((3, 4)))
(Note("af'4"), TimeSignature((3, 4)))

>>> inspect_(measure_2).get_indicator(TimeSignature)
TimeSignature((5, 4))

>>> for leaf in measure_2.select_leaves():
...     time_signature = inspect_(leaf).get_effective(TimeSignature)
...     print(leaf, time_signature)
...
(Note("af'4."), TimeSignature((5, 4)))
(Note("g'8"), TimeSignature((5, 4)))
(Note("c'8"), TimeSignature((5, 4)))
(Note("b'4."), TimeSignature((5, 4)))
(Note("b'8"), TimeSignature((5, 4)))
(Rest('r16'), TimeSignature((5, 4)))
(Note("d'16"), TimeSignature((5, 4)))

```

### 2.3.2 ANNOTATIONS

Indicators may be attached to score components as *annotations*, “visible” to inspection and potentially scoped, but contributing no formatting to a score hierarchy’s LilyPond output. Additionally, any attached indication which cannot contribute formatting is considered an implicit annotation. Consider some of the indicators used above, such as clefs, time signatures, dynamics and tempo indications. All of these objects can be formatted as LilyPond syntax, and when attached to components in a score will appear in the output if and when the score itself is formatted as LilyPond syntax:

```

>>> clef = Clef('bass')
>>> print(format(clef, 'lilypond'))
\clef "bass"

>>> time_signature = TimeSignature((5, 4))
>>> print(format(time_signature, 'lilypond'))
\time 5/4

>>> dynamic = Dynamic('p')
>>> print(format(dynamic, 'lilypond'))
\p

```

```

>>> tempo = Tempo((1, 4), 88)
>>> print(format(tempo, 'lilypond'))
\tempo 4=88

```

For example, the above four indicators can be attached to the contents of the following staff which, when formatted, shows the expected format contributions of each indicator. Note that the staff must be wrapped in a score container so that its tempo indication can find the appropriate context:

```

>>> staff = Staff("g f e d c")
>>> attach(clef, staff)
>>> attach(dynamic, staff)
>>> attach(tempo, staff)
>>> attach(time_signature, staff)
>>> Score([staff])
<Score<<1>>

>>> print(format(staff))
\new Staff {
  \clef "bass"
  \tempo 4=88
  \time 5/4
  g4
  f4
  e4
  d4
  c4
}

>>> show(staff)

```



The above staff can be recreated with the indicators attached as annotations, still effective for each leaf, but providing no format contributions in the output. This results in rather poor notation, falling back on LilyPond's default  $\frac{4}{4}$  time signature and treble clef:

```

>>> staff = Staff("g f e d c")
>>> attach(clef, staff, is_annotation=True)
>>> attach(dynamic, staff, is_annotation=True)
>>> attach(tempo, staff, is_annotation=True)
>>> attach(time_signature, staff, is_annotation=True)
>>> Score([staff])
<Score<<1>>

>>> print(format(staff))
\new Staff {
  g4

```

```

f4
e4
d4
c4
}

>>> show(staff)

```



Still, the leaves in this second staff can be inspected and will all report that various indicators attached to the staff, although annotative, are effective for them:

```

>>> for note in staff:
...     tempo = inspect_(note).get_effective(Tempo)
...     print(note, tempo)
...
(Note('g4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('f4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('e4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('d4'), Tempo(duration=Duration(1, 4), units_per_minute=88))
(Note('c4'), Tempo(duration=Duration(1, 4), units_per_minute=88))

```

Annotation allows arbitrary objects to be attached to any score component as a kind of metadata without directly affecting LilyPond's typesetting, allowing other later compositional processes to inspect or react to those annotations. Abjad provides a variety of indicator classes for this purpose, none of which contribute any formatting, but all of which can be attached and scoped, allowing composers to better model instrumental technique or compositional intent. Some of these annotative indicators include `BowContactPoint`, `IsAtSoundingPitch`, `IsUnpitched`, `StringContactPoint` and `StringTuning`.

## 2.4 SPANNERS

A final collection of classes, *spanners*, model musical and typographic constructs such as slurs, beams, hairpins, and glissandi which *span* across different levels of hierarchy in the score tree. Spanners attach to not just one but potentially many components in a score tree, provided those components are contiguous in time and occupy the same *logical voice*.<sup>6</sup> Like indicators, multiple spanners can attach to the same component. Unlike indicators, spanners are

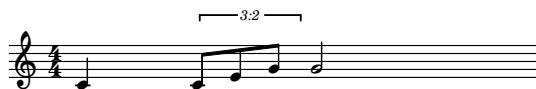
---

<sup>6</sup>A logical voice is a structural relationship between components in a score representing a single musical voice. For example, notes may be contained directly by a staff, without any explicit voice container between those notes and the staff. While those notes do not share an explicit voice, they do share an implicit logical voice. Likewise, notes contained by two explicit voice contexts where the explicit voices share the same name not only inhabit explicit voices but share the same logical voice.

aware of the components, including the leaves, they *cover* as well as the order in which those components appear. This awareness allows spanners to contribute formatting to a score tree's LilyPond syntax output conditionally. The first and last leaves covered by a spanner can be formatted specially, and the types of leaves covered by a spanner, their durations or even any indicators attached to those leaves can be considered during formatting, allowing spanners to make use of annotative indicators attached to components.

Consider the following staff, which contains leaves as well as a triplet, itself containing more leaves:

```
>>> staff = Staff(r"c'4 \times 2/3 { c'8 e'8 g'8 } g'2")
>>> show(staff)
```



A variety of spanners can be attached to both the leaves of the staff as well as to the staff itself using Abjad's `attach()` function. For example, a slur can be attached to all of the leaves of the staff – including those in the triplet –, a crescendo attached to the staff itself – but not to its leaves –, and two ties attached to the pairs of C- and G-notes which border the bounds of the triplet:

```
>>> slur = Slur()
>>> crescendo = Crescendo()
>>> tie_one = Tie()
>>> tie_two = Tie()
>>> leaves = staff.select_leaves()
>>> attach(slur, leaves)
>>> attach(crescendo, staff)
>>> attach(tie_one, leaves[:2])
>>> attach(tie_two, leaves[-2:])
>>> show(staff)
```



As with indicators, the component inspector provides methods for inspecting any spanners attached to a given component:

```
>>> inspect_(staff[0]).get_spanners()
set([Tie("c'4, c'8"), Slur("c'4, c'8, e'8, g'8, g'2")])

>>> inspect_(staff).get_spanners()
set([Crescendo('<Staff{3}>')])
```

Note above that the results of each inspection for spanner attachments returns a set comprising a unique collection of spanners. Analogous to component parents and children, each component can appear only once in a single spanner's component selection, and a spanner can only be attached once to the same component. Spanners can also be inspected – much like containers – for any components they cover via indexing, as well as for their duration:

```
>>> len(slur)
5

>>> slur[1]
Note("c'8")

>>> slur[:]
Selection(Note("c'4"), Note("c'8"), Note("e'8"), Note("g'8"), Note("g'2"))

>>> inspect_(slur).get_duration()
Duration(1, 1)
```

Note that subscripting the spanner with Python's slice notation `[:]` returns a *selection* object containing all of the components to which the spanner attaches. The role of selections is discussed in more detail in section 2.6.

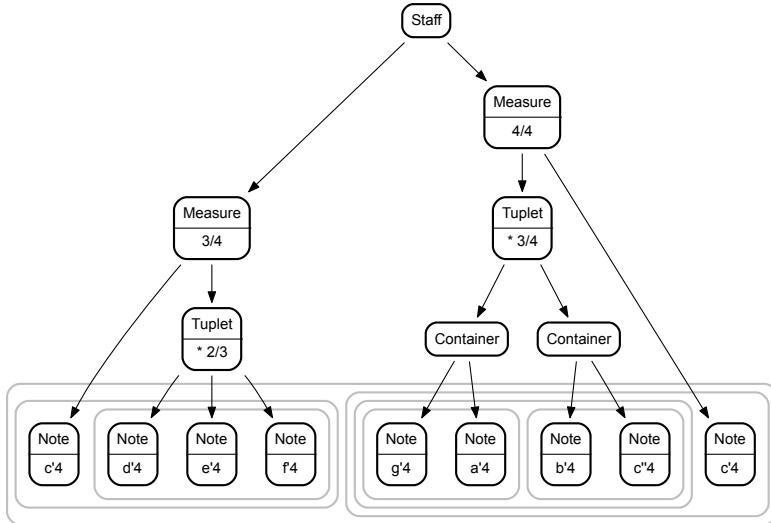
Returning to the graph visualizations of score trees demonstrated earlier will clarify how spanners attach to components. Consider the following new staff, containing two measures of differing durations, each containing a mixture of tuplets and notes. The second measure's tuplet contains two bare containers, each containing two notes in turn. The leaves of the staff then vary in depth from 2 – contained by a measure, then by the staff – to 4 – contained by a bare container, then a tuplet, a measure and finally the staff:

```
>>> staff = Staff()
>>> measure_one = Measure((3, 4), r"c'4 \times 2/3 { d'4 e'4 f'4 }")
>>> measure_two = Measure((4, 4), r"\times 3/4 { { g'4 a'4 } { b'4 c'4 } } c'4")
>>> staff.extend([measure_one, measure_two])
>>> show(staff)
```



The staff's score hierarchy can be visualized as a graph with the leaves arranged at the bottom and grouped together according to their depth:

```
>>> graph(staff)
```



A Spanner can be attached to the leaves of the staff. Note that while all spanners' structural behavior is identical to that of their parent Spanner class, Spanner itself provides no formatting beyond the optional typographic overrides discussed in section 2.5, so there is no need to illustrate the newly-altered staff as notation. However, the staff can be graphed again, this time highlighting and connecting together the nodes in the graph representing the components to which the spanner attaches by passing that spanner as a keyword to `graph()`:<sup>7</sup>

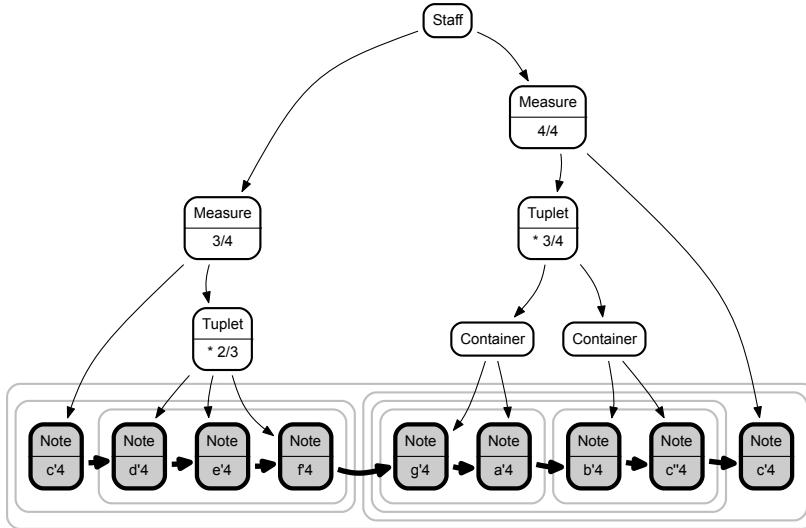
```

>>> spanner = spannertools.Spanner()
>>> attach(spanner, staff.select_leaves())
>>> graph(staff, spanner=spanner, graph_attributes={
...     'splines': 'curved',
...     'concentrate': True,
... })

```

---

<sup>7</sup>Abjad relies on the open-source Graphviz toolkit to visualize graph structures. Abjad also provides an object-oriented model of Graphviz's input file format as part of its documentationtools subpackage. There are many layout options available when working with Graphviz, and they can be passed as keywords to Abjad's top-level `graph()` function, hence the appearance of the `graph_attributes` keyword. These overrides are necessary to circumvent deficiencies in Graphviz's handling of this particular graph's layout.

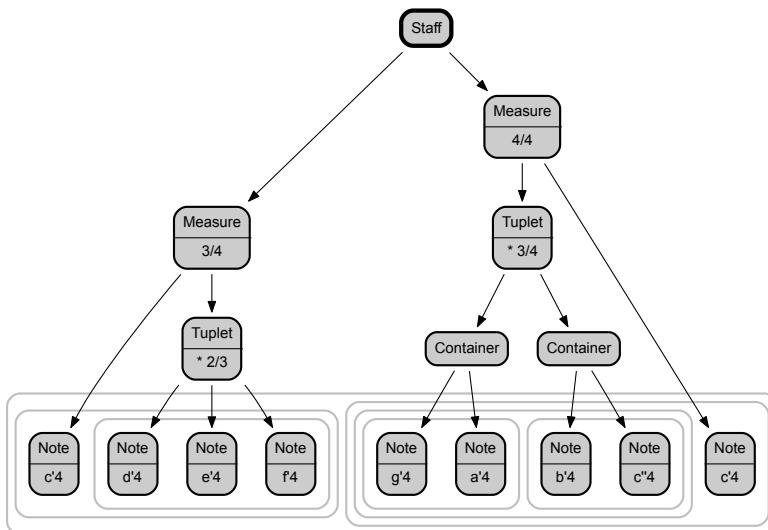


The spanner can be detached from the staff's leaves and reattached to the staff itself. Recall that spanners can attach to one or more of any combination of containers and leaves, provided all are contiguous in time and occupy the same logical voice. Note in the following graph that the node representing the staff is both outlined in bold and shaded grey while all of the components underneath it in the score hierarchy are simply shaded grey. In these score hierarchy graphs, only those nodes to which the spanner directly attaches are outlined in bold, while all components that the spanner covers are simply shaded:

```

>>> detach(spanner)
>>> attach(spanner, staff)
>>> graph(staff, spanner=spanner)

```

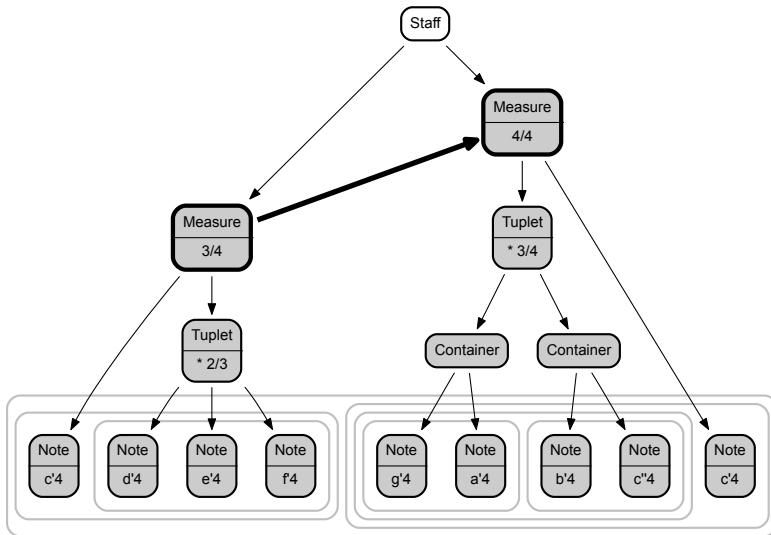


The spanner can then be detached from the staff and attached to the two measures it contains:

```

>>> detach(spanner)
>>> attach(spanner, staff[:])
>>> graph(staff, spanner=spanner)

```

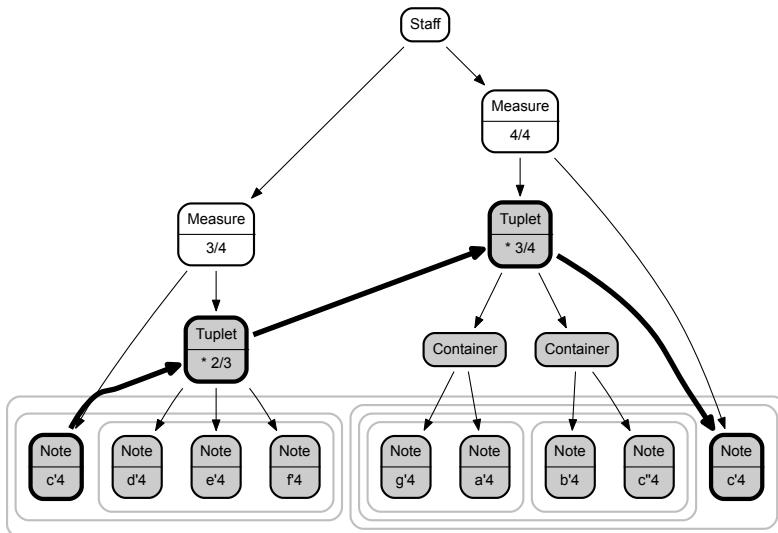


The spanner can also be detached again and reattached to a selection created by selecting the contents of both measures and concatenating those selections together. Note that the spanner's components – those nodes outlined in bold and connected by bold arrows – here comprise both leaves and containers, but outline the same collection of leaves as the previous graphs:

```

>>> detach(spanner)
>>> attach(spanner, measure_one[:] + measure_two[:])
>>> graph(staff, spanner=spanner)

```

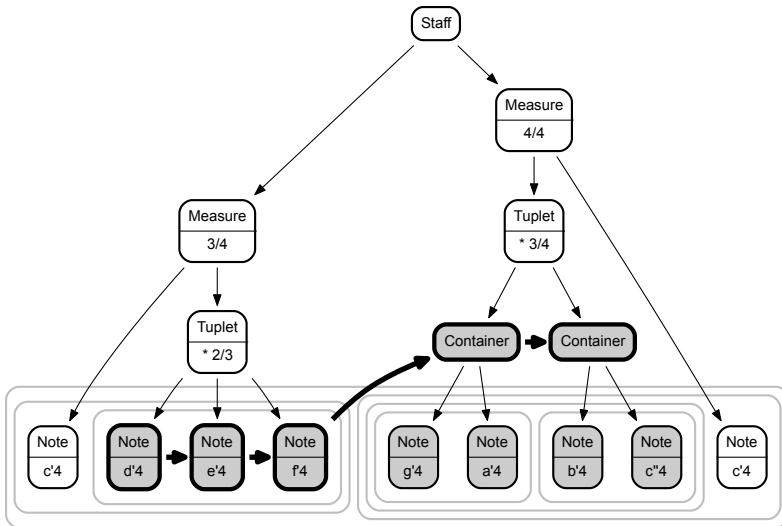


Finally, the spanner can be detached from the previous selection and reattached to a new selection comprising the concatenated contents of each measures' tuplet child:

```

>>> detach(spanner)
>>> attach(spanner, measure_one[1][:] + measure_two[0][:])
>>> graph(staff, spanner=spanner)

```

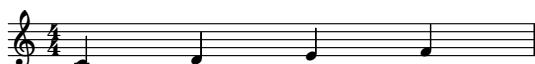


Yet despite being attachable to both containers and leaves, spanners only contribute formatting to the leaves they cover, not to any container. Why attach spanners to containers at all? Any spanner attached to a container will contribute formatting to the leaves of that container, regardless of what those leaves are, and even if they change during the course of composition. Consider the following small staff example, containing four notes, with the inner two notes wrapped in a bare container:

```

>>> staff = Staff("c'4 { d'4 e'4 } f'4")
>>> show(staff)

```



After attaching a slur to the inner container, the slur appears in the notational output, starting on the first leaf of the inner container and stopping on the last leaf:

```

>>> attach(Slur(), staff[1])
>>> show(staff)

```

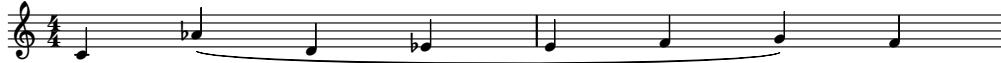


More leaves can be inserted into the inner container, both before, between and after the original two notes. The slur still appears starting on the first leaf of the inner container and continuing until its last, although both have changed from the original two notes:

```

>>> staff[1].insert(1, "ef'4")
>>> staff[1].insert(0, "af'4")
>>> staff[1].extend("f'4 g'4")
>>> show(staff)

```



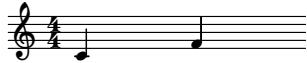
The inner container can be emptied completely, at which point the slur no longer appears in any notational output.

However, inspecting the inner container's inventory of spanners shows that the slur is still attached:

```

>>> staff[1][:] = []
>>> print(format(staff))
\new Staff {
    c'4
    {
        f'4
    }
}
>>> show(staff)

```



```

>>> inspect_(staff[1]).get_spanners()
set([Slur('Container')])

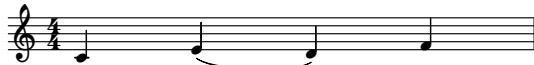
```

Finally, the inner container can be repopulated, allowing the slur to appear again:

```

>>> staff[1].extend("e'4 d'4")
>>> show(staff)

```



Attaching spanners to containers rather than leaves lets spanners act as a kind of *decorator*<sup>8</sup> for those containers in the sense of computer programming design patterns: an object which wraps another object, extending it with new functionality while preserving all of the old. In this case, spanners can extend containers with additional typographic logic, but remain entirely ignorant of the container's contents.

## 2.5 TYPOGRAPHIC OVERRIDES

LilyPond's graphic objects, or *grobs* in LilyPond parlance, represent all of the typographic glyphs which may appear on the page as the result of the typesetting process. These include note-heads, dots, stems, flags, beams, slurs, accidentals, articulations, clefs, bar-numbers and bar-lines, the lines of the staff itself, as well as all manner of *invisible*

grobs used to describe the spacing and positioning of other glyphs on the page. Every such grob in LilyPond can be configured extensively. An accidental's color can be changed, its size increased or decreased, its position relative to its parent grob – the note-head it modifies – adjusted, or its symbol replaced entirely. LilyPond calls the act of modifying graphic objects *overriding*, and the act of returning them to their previous settings *reverting*. Abjad provides a top-level `override()` function for constructing such LilyPond `\override` and `\revert` commands, allowing composers to alter the typography of individual leaves or the contents of entire containers. Similarly to `inspect_()`, calling `override()` on a score component returns an agentive object with that component as its client: a *LilyPond grob name manager*. This manager object exposes an interface for constructing override statements in a dot-chained syntax which mimics the way one would write LilyPond override commands by hand. For example, the style of a single note's note-head can be changed from the default oval shape to a cross:

```
>>> note = Note("c'4")
>>> override(note).note_head.style = 'cross'
>>> print(format(note))
\once \override NoteHead #'style = #'cross
c'4

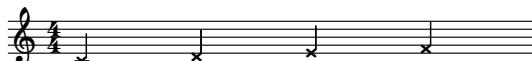
>>> show(note)
```



In the resulting LilyPond code above, Abjad prepends a `\once` modifier to the `override` command, obviating the need to explicitly revert the typography override afterward. The same override can be applied against an entire container full of notes, modifying each note in turn:

```
>>> container = Container("c'4 d'4 e'4 f'4")
>>> override(container).note_head.style = 'cross'
>>> print(format(container))
{
  \override NoteHead #'style = #'cross
  c'4
  d'4
  e'4
  f'4
  \revert NoteHead #'style
}

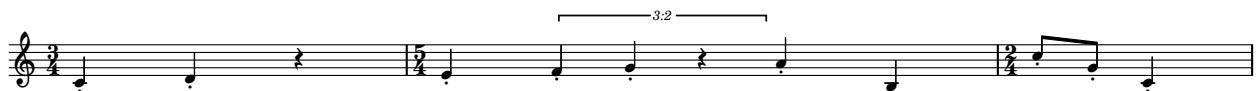
>>> show(container)
```



Note above in the LilyPond syntax output that the override command appears just inside the opening brace of the container, and the accompanying revert command appears just before the closing brace. Additionally, much like the earlier spanner examples where a slur was attached directly to a container, overrides applied against a container persist regardless of the contents of that container.

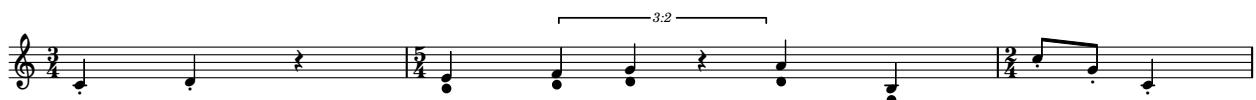
As with score components, composers can also apply typographic overrides against spanners. The initial override commands appears before the first leaf of the spanner – if the spanner covers any leaves at all – and the accompanying revert commands appear following the last leaf of the spanner. Consider the following staff example, containing three measures, with staccato articulations attached to each leaf:

```
>>> staff = Staff()
>>> staff.append(Measure((3, 4), "c'4 d' r"))
>>> staff.append(Measure((5, 4), r"e'4 \times 2/3 { f' g' r4 } a' b"))
>>> staff.append(Measure((2, 4), "c'8 g' c'4"))
>>> articulation = Articulation('staccato')
>>> for leaf in staff.select_leaves():
...     if isinstance(leaf, Note):
...         attach(articulation, leaf)
...
>>> show(staff)
```



We can override the middle measure to increase the size of the staccato articulations – what LilyPond terms *scripts* – by increasing their font size:

```
>>> override(staff[1]).script.font_size = 10
>>> show(staff)
```



Next we attach a slur to all leaves starting from the second and going until the next-to-last. We also override that slur to change the style of the note head of each note and chord it covers from the default oval to a cross. Recall that spanners can cross different levels of hierarchy in the score tree. Likewise, typographic overrides applied against spanners also cross container boundaries such as the three measures contained within this staff:

```

>>> slur = spanertools.Slur(direction=Down)
>>> attach(slur, staff.select_leaves()[1:-1])
>>> override(slur).note_head.style = 'cross'
>>> show(staff)

```



Note the locations of the \override and \revert commands for both the spanner and the middle measure in the LilyPond syntax output. While the middle measure's \override and \revert commands appear just within its enclosing braces, the slur's \override command appears just before its first leaf – the D quarter-note – and its \revert command just after its last leaf – the G eighth-note:

```

>>> print(format(staff))
\new Staff {
{
    \time 3/4
    c'4 -\staccato
    \override NoteHead #'style = #'cross
    d'4 -\staccato _ (
    r4
}
{
    \override Script #'font-size = #10
    \time 5/4
    e'4 -\staccato
    \times 2/3 {
        f'4 -\staccato
        g'4 -\staccato
        r4
    }
    a'4 -\staccato
    b4 -\staccato
    \revert Script #'font-size
}
{
    \time 2/4
    c'8 -\staccato
    g'8 -\staccato )
    \revert NoteHead #'style
    c'4 -\staccato
}
}

```

The typographic overrides of both the slur and the middle measure persist even when changing the contents of that middle measure, such as replacing the triplet with four eighth-notes:

```
>>> staff[1][1:2] = "f'8 -. g'8 -. fs'8 -. g'8 -."
>>> show(staff)
```



Importantly, typographic overrides *cascade*, with local settings taking precedence over more global settings. Although the previously attached slur overrides the note-head style of all leaves it covers, the note-head styles of leaves within that collection can be overridden themselves. For example, the first, last and next-to-last leaf of the middle measure can be overridden to display a slash symbol rather than cross for their note-heads, effectively overruling the more global typographic commands applied against their covering spanner:

```
>>> for index in (0, -2, -1):
...     override(staff[1][index]).note_head.style = 'slash'
...
>>> show(staff)
```



By providing an object-oriented interface to the low-level typographic options of its typesetter, LilyPond, Abjad affords composers powerful programmatic control over not only the content but the appearance of the scores they create.

## 2.6 SELECTING COMPONENTS

Abjad provides a variety of methods for selecting components from within a score tree. A number of these techniques have already been demonstrated. For example, a score can be subscripted with integers or integer *slices* – e.g. [1:-1] – representing the indices of some component or components in order to select those component, as shown in subsection 2.2.1 and later. Similarly, named components can be selected from a score by subscripting that score with their names, as shown in subsection 2.2.4. The leaves at the bottom of a score hierarchy can also be selected en masse via a component’s `select_leaves()` method regardless of their depth within the tree, also shown in subsection 2.2.1. All of these techniques represent *component selection*.

Abjad object-models the concept of component selection explicitly with its `Selection` class. Selections allow arbitrary collection of components to be grouped together and inspected without altering the score tree in the

act of selecting. Unlike containers or spanner, components grouped by a selection have no reference back to that selection. Such unidirectional references prevent considerable complexity in the system and allow selections to be made and discarded with ease. However, like containers, spanners and the component inspector, selections expose a variety of methods for examining the components they group. Consider the following staff, which contains a variety of spanners attached to its leaves:

```
>>> staff = Staff("c'8 ( [ d' e' f' ] g' [ a' b' c'' ) ]")
>>> show(staff)
```



A selection can be made simply by subscripting the staff to select from the third leaf up to, but not including, the seventh:

```
>>> selection = staff[2:6]
>>> print(format(selection))
selectiontools.SliceSelection(
(
    scoretools.Note("e'8"),
    scoretools.Note("f'8 ["),
    scoretools.Note("g'8 ["),
    scoretools.Note("a'8"),
)
)
```

That selection can be iterated over like a list, and inspected for its total duration and timespan – its start and stop offsets, discussed at length in chapter 3:

```
>>> for component in selection:
...     component
...
Note("e'8")
Note("f'8")
Note("g'8")
Note("a'8")

>>> selection.get_duration()
Duration(1, 2)

>>> selection.get_timespan()
Timespan(start_offset=Offset(1, 4), stop_offset=Offset(3, 4))
```

Furthermore, the selection can be passed as the target to which a spanner attaches. Recall the use of subscripts in calls to `attach()` in section 2.4. All of those container subscripting expressions produced selection objects, although they were not displayed on the console at any point. In this case, we can attach a glissando spanner to the leaf selection:

```
>>> attach(Glissando(), selection)
>>> show(staff)
```



Selections can also be queried for the spanners covering the components they contain:

```
>>> for spanner in selection.get_spanners():
...     spanner
...
Glissando("e'8, f'8, g'8, a'8")
Beam("g'8, a'8, b'8, c''8")
Beam("c'8, d'8, e'8, f'8")
Slur("c'8, d'8, e'8, f'8, g'8, a'8, b'8, c''8")
```

### 2.6.1 LOGICAL TIES

Abjad provides a number of selection subclasses, some of which have already been described. For example, the `Parentage` class, discussed in subsection 2.2.2 is actually a type of selection, and affords the same methods described above in addition to many others. One of the most fundamental selection subclasses in Abjad is the `LogicalTie`. Logical ties – which are distinct from tie *spanners* – model the concept of one or more leaves tied together to create a single attack point with an aggregate duration. Logical ties containing a single note or chord are termed *trivial*. Logical ties comprising rests are not well-defined, but can certainly be instantiated like any other selection if a composer would find them useful.

Consider the following staff, which contains a mixture of notes and rests. The “~” symbol in LilyPond syntax indicates that two notes should be joined by a tie. Notably, this staff contains notes which are untied, notes which are tied to a single other note, a chain of three notes tied together, and ties which cross the bounds of a triplet:

```
>>> staff = Staff(context_name='RhythmicStaff')
>>> staff.extend(r"c'4 r4.. c'16 ~ c'4 ~ \times 2/3 { c'8 r4 c'8 ~ } c'4 r4 c'4")
>>> show(staff)
```



The most direct way to select a leaf’s logical tie is via the component inspector’s `get_logical_tie()` method. Once selected, the logical tie can be examined or printed much like any other selection:

```
>>> logical_tie = inspect_(staff[2]).get_logical_tie()
>>> print(format(logical_tie))
selectiontools.LogicalTie(
```

```

        (
            scoretools.Note("c'16 ~"),
            scoretools.Note("c'4 ~"),
            scoretools.Note("c'8"),
        )
    )

>>> logical_tie.get_duration()
Duration(19, 48)

>>> logical_tie.get_timespan()
Timespan(start_offset=Offset(11, 16), stop_offset=Offset(13, 12))

```

Like `Parentage`, logical tie selections expose properties which are specific to their domain. Two of the most important such properties are a logical tie's *head* and *tail*: the first and last leaves of the logical tie respectively. We can label the previously selected logical tie's head and tail with textual *markup*.<sup>8</sup>

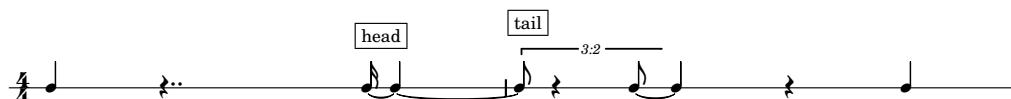
```

>>> logical_tie.head
Note("c'16")

>>> logical_tie.tail
Note("c'8")

>>> head_markup = Markup('head', direction=Up)
>>> head_markup = head_markup.pad_around(0.5).box().pad_around(0.5)
>>> tail_markup = Markup('tail', direction=Up)
>>> tail_markup = tail_markup.pad_around(0.5).box().pad_around(0.5)
>>> attach(head_markup, logical_tie.head)
>>> attach(tail_markup, logical_tie.tail)
>>> show(staff)

```



Logical ties expose other pertinent properties. For example, a logical tie can be queried for whether all of its leaves occupy the same parent, whether none of its leaves are rests – therefore the logical tie is pitched –, or whether the logical tie is trivial:

```

>>> logical_tie.all_leaves_are_in_same_parent
False

>>> logical_tie.is_pitched
True

```

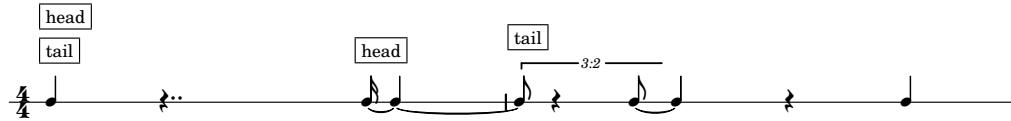
---

<sup>8</sup>LilyPond provides a *markup* environment for formatting text and graphics, which can then be attached to leaves in the score, used as titles, headers, footers and so forth, or set as the *stencil* for various grobs, overriding that grob's original appearance entirely. Abjad models LilyPond's markup environment with its `Markup` class, which exposes many dot-chainable methods for configuring markup objects, mirroring the markup functions available in LilyPond.

```
>>> logical_tie.is_trivial  
False
```

Trivial logical ties can also be selected. For example, the first leaf in the above staff is covered by no tie spanner at all. Still, that leaf can be modeled as a logical tie containing only a single note. A trivial logical tie's head and tail are then necessarily the same component:

```
>>> trivial_logical_tie = inspect_(staff[0]).get_logical_tie()  
>>> trivial_logical_tie.is_trivial  
True  
  
>>> print(format(trivial_logical_tie))  
selectiontools.LogicalTie(  
(  
    scoretools.Note("c' 4"),  
)  
)  
  
>>> attach(head_markup, trivial_logical_tie.head)  
>>> attach(tail_markup, trivial_logical_tie.tail)  
>>> show(staff)
```



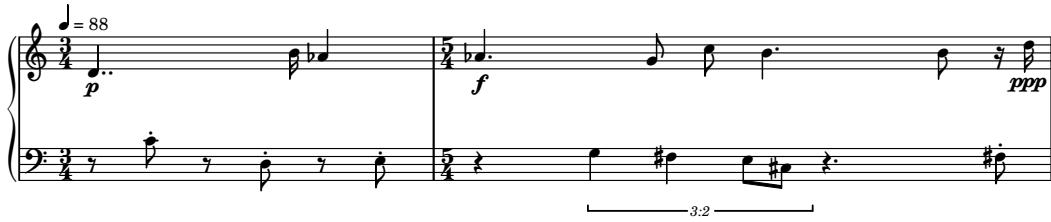
## 2.6.2 ITERATION

Abjad provides a variety of techniques for *iterating* over the components in a score tree. The “simplest” iteration technique employed in Abjad relies on Python’s *iteration protocol*, which allows objects to be iterated over in “for” loops – as well as many other constructs – yielding up items from their contents one at a time:

```
>>> for x in ['foo', 10, 3.14159]:  
...     x  
...  
'foo'  
10  
3.14159
```

Recall the two-staff score created earlier. Using this basic iteration technique – already demonstrated in subsection 2.2.1 –, the components immediately contained by any Abjad container can be iterated over as though that container were a list. For example, the components immediately contained in the score, the score’s staff group and the voice contained in the upper staff can be iterated over:

```
>>> show(score)
```



```
>>> for component in score:  
...     component  
...  
<PianoStaff-"Both Staves"<<2>>>
```

```
>>> for component in score['Both Staves']:  
...     component  
...  
<Staff-"Upper Staff">{1}>  
<Staff-"Lower Staff">{1}>  
  
>>> for component in score['Voice 1']:  
...     component  
...  
Measure((3, 4), "d'4.. b'16 af'4")  
Measure((5, 4), "af'4. g'8 c''8 b'4. b'8 r16 d''16")
```

Abjad also provides an *iteration agent* – much like Abjad’s component inspector – which exposes a variety of iteration methods. Calling the top-level `iterate()` function against a score component returns an iteration agent configured to iterate over the contents of that component:

```
>>> iteration_agent = iterate(score)
```

Perhaps the most fundamental method of score tree traversal is *depth-first iteration*<sup>7</sup> whereby not only is a given container iterated over but each container it contains as well and each container those contain, recursively down to the leaves of the score tree. The iteration agent exposes depth first traversal via a method call. We can demonstrate this traversal algorithm by inspecting the parentage of each score component yielded during each step of the traversal process and printing them to the terminal indented by their “depth” relative to the root of the score tree:

```
>>> for component in iteration_agent.depth_first():  
...     parentage = inspect_(component).get_parentage()  
...     component_depth = parentage.depth  
...     indent = '    ' * component_depth  
...     string = '{}{}'.format(indent, repr(component))  
...     print(string)  
...  
<Score<<1>>>
```

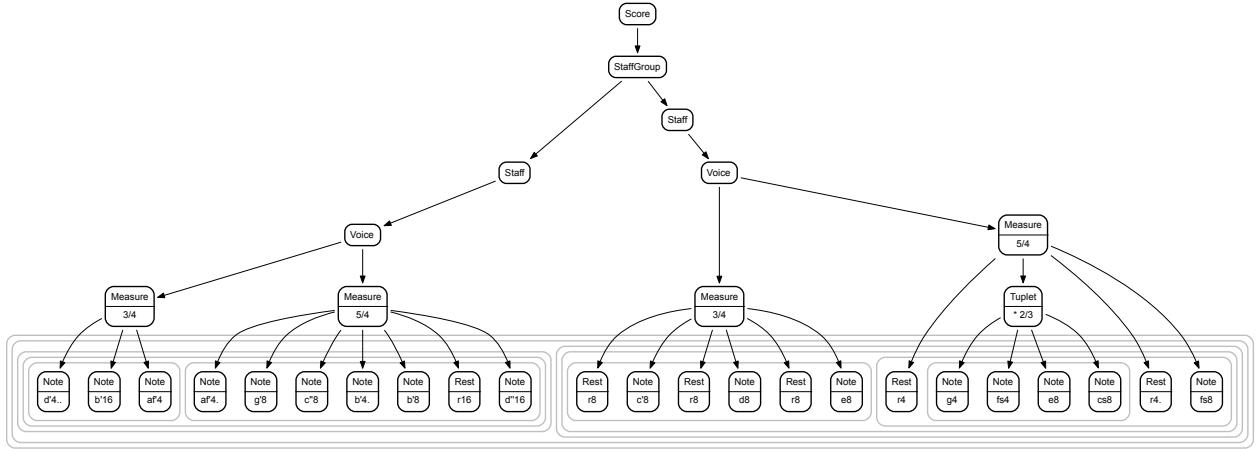
```

<PianoStaff-”Both Staves”<<2>>>
    <Staff-”Upper Staff”{1}>
        <Voice-”Voice 1”{2}>
            Measure((3, 4), ”d’4.. b’16 af’4”)
                Note(”d’4..”)
                Note(”b’16”)
                Note(”af’4”)
            Measure((5, 4), ”af’4. g’8 c’8 b’4. b’8 r16 d’16”)
                Note(”af’4.”)
                Note(”g’8”)
                Note(”c’8”)
                Note(”b’4.”)
                Note(”b’8”)
                Rest(’r16’)
                Note(”d’16”)
        <Staff-”Lower Staff”{1}>
            <Voice-”Voice 2”{2}>
                Measure((3, 4), ”r8 c’8 r8 d8 r8 e8”)
                    Rest(’r8’)
                    Note(”c’8”)
                    Rest(’r8’)
                    Note(’d8’)
                    Rest(’r8’)
                    Note(’e8’)
                Measure((5, 4), ’r4 { 2/3 g4 fs4 e8 cs8 } r4. fs8’)
                    Rest(’r4’)
                    Tuplet(Multiplier(2, 3), ’g4 fs4 e8 cs8’)
                        Note(’g4’)
                        Note(’fs4’)
                        Note(’e8’)
                        Note(’cs8’)
                    Rest(’r4.’)
                    Note(’fs8’)

```

Note how the above indented output mirrors the graph visualization of the same score tree, with the indent of each components representation corresponding to the number of edges between that component and the score tree’s root score container:

```
>>> graph(score)
```

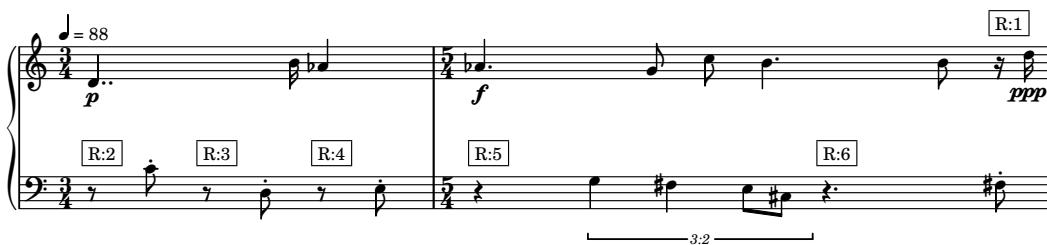


In fact, Abjad's score tree graph visualizations, like many of the other more-specialized score tree iteration techniques – including the `select_leaves()` method shown throughout this document –, rely on this depth-first traversal. One of the most common iteration techniques based closely on depth-first traversal is iteration by class, whereby a score tree is iterated over recursively and only those items matching a component class prototype – e.g. only rests, only tuplets, only notes and chords, or any type of leaf – are yielded. We can demonstrate this technique by iterating over the same score by rests and attaching some markup to each yielded rest:<sup>9</sup>

```

>>> iterator = iteration_agent.by_class(Rest)
>>> for count, rest in enumerate(iterator, start=1):
...     string = 'R:{}'.format(count)
...     markup = Markup(string, direction=Up)
...     markup = markup.pad_around(0.5).box().pad_around(0.25)
...     attach(markup, rest)
...
>>> show(score)

```



Note how the attached markup orders rests from left-to-right, starting in the upper staff and then proceeding to the lower staff. Such a traversal mirrors the order in which those rests appear during depth-first traversal. Now, before

---

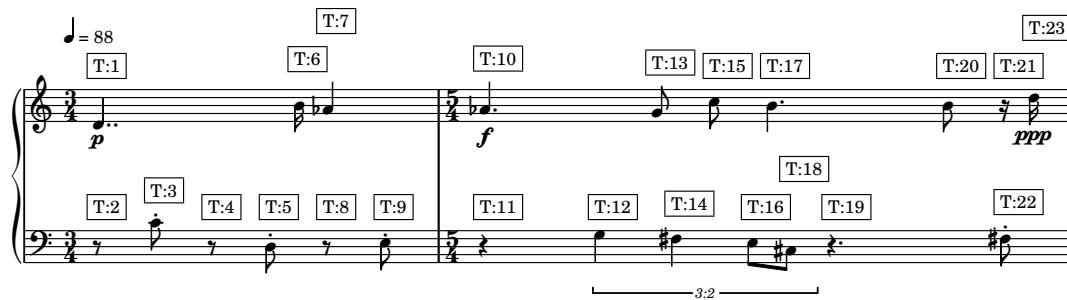
<sup>9</sup>The built-in Python `enumerate` function returns an iterator which yields pairs of indices and items from its iterable argument, effectively giving the index of each item in that original iterable. In the case of the following code example, the original iterable may itself be an iterator object rather than a list or tuple. This is an example of Python's iterator protocol in action, as `enumerate` accepts any argument as long as it can be iterated over.

proceeding to a demonstration of the next technique, we first iterate over all of the components in the score tree depth-first and detach any markup previously attached, such as the rest-numbering markup attached above:

```
>>> for component in iteration_agent.depth_first():
...     detached = detach(Markup, component)
```

Iteration by *timeline* is another important score traversal technique, yielding each component – optionally filtered by a class prototype – in the order in which they in time relative to the start of the score. Components with identical start times are yielded according to their score index – a tuple of indices describing the index of each of the components in a component’s parentage in *their* parents. We can demonstrate this iteration technique by iterating over all of the leaves in time-wise order, attaching markup to display their index in the timeline:

```
>>> iterator = iteration_agent.by_timeline(scoretools.Leaf)
>>> for count, leaf in enumerate(iterator, 1):
...     string = 'T:{}'.format(count)
...     markup = Markup(string, direction=Up)
...     markup = markup.pad_around(0.5).box().pad_around(0.25)
...     attach(markup, leaf)
...
>>> show(score)
```



Composers are often concerned with contiguous *runs* of notes and chords rather than any other collection of components. The iteration agent’s `by_run()` method iterates over such contiguous groups by class:

```
>>> lower_leaves = score['Voice 2'].select_leaves()
>>> for run in iterate(lower_leaves).by_run(Note):
...     run
...
(Note("c'8"),)
(Note('d8'),)
(Note('e8'),)
(Note('g4'), Note('fs4'), Note('e8'), Note('cs8'))
(Note('fs8'),)
```

Finally, score trees can be traversed by logical ties – selections representing one or more component tied together by a tie spanner, as described in section 2.6. Before demonstrating this technique we must both attach ties to some of the leaves in the upper staff – the A-flats and the Bs – and strip out the previously-attached markup:

```
>>> attach(Tie(), score['Upper Staff'].select_leaves()[2:4])
>>> attach(Tie(), score['Upper Staff'].select_leaves()[6:8])
>>> for component in iterate(score).depth_first():
...     detached = detach(Markup, component)
...
...
```

With the score prepared, all pitched logical ties in the upper staff can be iterated over and markup attached to the head of each:

```
>>> upper_leaves = score['Voice 1'].select_leaves()
>>> for logical_tie in iterate(upper_leaves).by_logical_tie(pitched=True):
...     markup = Markup('H', direction=Up)
...     markup = markup.pad_around(0.5).box()
...     attach(markup, logical_tie.head)
...
>>> show(score)
```

### 2.6.3 SELECTORS

Component selectors are highly-configuration objects which object-model the act of selecting components – as demonstrated above – by aggregating together a series of small *callback* classes into a *pipeline*. Each callback describes one step in the act of performing a complex selection, such as selecting leaves, selecting logical ties, selecting the first item from a selection, or selecting items whose length is longer than some count. Selectors allow composers to model the procedure by which they, for example, select rest-delimited runs of notes and chords, then select all logical ties within those runs and finally select the last leaf of each logical tie, even if that logical tie contains only a single leaf. Such a procedure, once codified in a fully-configured selector, can be applied against any leaf, container or selection of components.

When called against a component, an unconfigured selector simply selects that component, returning the selection wrapped within a sequence so that the result can be iterated over:

```

>>> selector = selectortools.Selector()
>>> for x in selector(score['Upper Staff']):
...     x
...
<Staff "Upper Staff">{1}>

```

The selector can be further configured by calling one of its various generating methods. These methods duplicate the selector's current sequence of callbacks and append a new one, returning a new selector containing the modified callback sequence.<sup>10</sup> For example, the previously-created selector can be configured to select the leaves in each selection it processes:

```

>>> selector = selector.by_leaves(flatten=True)
>>> for x in selector(score['Upper Staff']):
...     x
...
Note("d'4..")
Note("b'16")
Note("af'4")
Note("af'4.")
Note("g'8")
Note("c'8")
Note("b'4.")
Note("b'8")
Rest('r16')
Note("d'16")

```

The selector's storage format shows its callback sequence. The leaf selection callback appears here as a `PrototypeSelectorCallback`, itself configured to select components of type `Leaf`:

```

>>> print(format(selector))
selectortools.Selector(
    callbacks=(
        selectortools.PrototypeSelectorCallback(
            prototype=scoretools.Leaf,
            flatten=True,
        ),
    ),
)

```

The selector can be configured to select all pitched logical-ties from within the previous selections.

```

>>> selector = selector.by_logical_tie(pitched=True)
>>> for x in selector(score['Upper Staff']):

```

---

<sup>10</sup> Selector configuration is conceptually similar to the textual markup configuration used elsewhere in this chapter. In both cases, the object can make itself produce new, modified versions of itself through the use of generative methods.

```

...
x
...
LogicalTie(Note("d'4.."),)
LogicalTie(Note("b'16"),)
LogicalTie(Note("f'4"), Note("af'4."))
LogicalTie(Note("g'8"),)
LogicalTie(Note("c'8"),)
LogicalTie(Note("b'4."), Note("b'8"))
LogicalTie(Note("d'16"),)

```

Again, note the addition of a new callback to the selector's callback pipeline:

```

>>> print(format(selector))
selectortools.Selector(
    callbacks=(
        selectortools.PrototypeSelectorCallback(
            prototype=scoretools.Leaf,
            flatten=True,
        ),
        selectortools.LogicalTieSelectorCallback(
            flatten=True,
            pitched=True,
            trivial=True,
        ),
    ),
)

```

The selector can be configured to filter out logical ties shorter than a duration of  $\frac{1}{4}$ :

```

>>> selector = selector.by_duration('>', Duration(1, 4))
>>> for x in selector(score['Upper Staff']):
...     x
...
LogicalTie(Note("d'4.."),)
LogicalTie(Note("af'4"), Note("af'4."))
LogicalTie(Note("b'4."), Note("b'8"))

```

The first item from each selected logical-tie can then be selected by using Python's index subscripting syntax.

```

>>> selector = selector[0]
>>> for x in selector(score['Upper Staff']):
...     print(repr(x))
...
ContiguousSelection(Note("d'4.."),)
ContiguousSelection(Note("af'4"),)
ContiguousSelection(Note("b'4."),)

```

Note that the returned selections each contain a single note. This is not necessarily useful yet. By appending a single additional flattening callback to the selector's callback stack, the selector can be configured to return a flat selection of components which can be iterated over, yielding only a single note on each iteration:

```

>>> selector = selector.flatten()
>>> for x in selector(score['Upper Staff']):
...     print(repr(x))
...
Note("d'4..")
Note("af'4")
Note("b'4.")

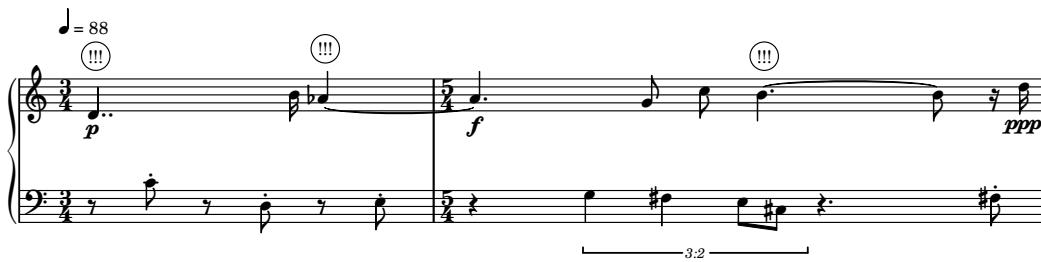
```

Unlike component iterators – which eventually terminate –, selectors can be re-used. The same selector which was called against the upper staff in the above example can be called again, returning the same result. As with the examples in subsection 2.6.2, we can label each component returned by the selector with some markup. Because the score was already labeled – marking the head of each logical tie in the upper staff – we first remove all markup by iterating depth-first:

```

>>> for component in iterate(score).depth_first():
...     detached = detach(Markup, component)
...
>>> markup = Markup('!!!', Up).pad_around(0.5).circle()
>>> for x in selector(score['Upper Staff']):
...     attach(markup, x)
...
>>> show(score)

```



Printing the storage format of the current selector shows its deeply-nested configuration:

```

>>> print(format(selector))
selectortools.Selector(
    callbacks=(
        selectortools.PrototypeSelectorCallback(
            prototype=scoretools.Leaf,
            flatten=True,
        ),
        selectortools.LogicalTieSelectorCallback(
            flatten=True,
            pitched=True,
            trivial=True,
        ),
        selectortools.DurationSelectorCallback(
            duration=selectortools.DurationInequality(
                operator_string='>',

```

```

        duration=durationtools.Duration(1, 4),
    ),
),
selectortools.ItemSelectorCallback(
    item=0,
    apply_to_each=True,
),
selectortools.FlattenSelectorCallback(
    depth=-1,
),
),
)
)

```

Selectors are incredibly flexible. They allow composers to describe in a great degree of detail the process they wish to use to select components from a score for further transformation or ornamentation. As will be elaborated on in the following chapters, they are also useful because they are fully object-modeled as classes. Such object-modeling allows them to not only be used to configure still-larger aggregate compositional objects, but also be persisted to disk due to their well-formed storage format.

## 2.7 TEMPLATING & PERSISTENCE

One of the last, but most important aspects of working with Abjad does not concern modeling notation at all. The act of *templating* takes one object and replaces the values assigned to one or more of its properties with new values, returning a new object based on – but differing from – the old, while the old object remains unchanged. Templating’s combination of copying and re-configuration is a standard concepts in computer science, but needs to be introduced here due to its pervasive use throughout the rest of this document.

Recall the earlier “rhythm-maker” code example from section 2.1:

```

>>> rhythm_maker = rhythmmakertools.TaleaRhythmMaker(
...     talea=rhythmmakertools.Talea([1, 2, 3], 16),
...     extra_counts_per_division=(1, 0, 2, 1, 0),
...     output_masks=[
...         rhythmmakertools.SustainMask([1], 3),
...         rhythmmakertools.SilenceMask([-1]),
...         rhythmmakertools.NullMask([0]),
...     ],
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tie_across_divisions=True,
...     ),
... )
>>> print(format(rhythm_maker))
rhythmmakertools.TaleaRhythmMaker(
    talea=rhythmmakertools.Talea(
        counts=(1, 2, 3),

```

```

denominator=16,
),
extra_counts_per_division=(1, 0, 2, 1, 0),
output_masks=rhythmmakertools.BooleanPatternInventory(
(
    rhythmmakertools.SustainMask(
        indices=(1,),
        invert=True,
    ),
    rhythmmakertools.SilenceMask(
        indices=(-1,),
    ),
    rhythmmakertools.NullMask(
        indices=(0,),
    ),
)
),
tieSpecifier=rhythmmakertools.TieSpecifier(
    tie_across_divisions=True,
),
)
)

```

Without yet discussing the structure or purpose of this particular object – all of which is covered at length in chapter 3 and chapter 4, especially in section 3.5 – we can demonstrate Abjad’s templating functionality via its top-level `new()` function. For example, `new()` can be used to template a new rhythm-maker from the old one shown above, replacing the value referenced by one of its properties with a new value. Here we replace its `extra_counts_per_division` with the sequence [1, 2, 3]. Note the change in the new rhythm-maker’s storage format:

```

>>> new_rhythm_maker = new(rhythm_maker,
...     extra_counts_per_division=[1, 2, 3],
... )
>>> print(format(new_rhythm_maker))
rhythmmakertools.TaleaRhythmMaker(
    talea=rhythmmakertools.Talea(
        counts=(1, 2, 3),
        denominator=16,
    ),
    extra_counts_per_division=(1, 2, 3),
    output_masks=rhythmmakertools.BooleanPatternInventory(
(
    rhythmmakertools.SustainMask(
        indices=(1,),
        invert=True,
    ),
    rhythmmakertools.SilenceMask(
        indices=(-1,),
    ),
    rhythmmakertools.NullMask(
        indices=(0,),
    ),
)
),
tieSpecifier=rhythmmakertools.TieSpecifier(
    tie_across_divisions=True,
),
)
)

```

```

        ),
    )
),
tieSpecifier=rhythmmakertools.TieSpecifier(
    tie_across_divisions=True,
),
)

```

Multiple values can be replaced, and previously unspecified values specified. Here we replace both the new rhythm-makers's `talea` and `extra_counts_per_division` values and specify a previously unspecified `beamSpecifier`:

```

>>> new_rhythm_maker = new_rhythm_maker,
...     talea=rhythmmakertools.Talea([2, 1], 8),
...     extra_counts_per_division=[1, 0, 1, 2, 0, 1],
...     beamSpecifier=rhythmmakertools.BeamSpecifier(
...         beam_divisions_together=True,
...         ),
...     )
>>> print(format(new_rhythm_maker))
rhythmmakertools.TaleaRhythmMaker(
    talea=rhythmmakertools.Talea(
        counts=(2, 1),
        denominator=8,
    ),
    extra_counts_per_division=(1, 0, 1, 2, 0, 1),
    beamSpecifier=rhythmmakertools.BeamSpecifier(
        beam_each_division=True,
        beam_divisions_together=True,
        use_feather_beams=False,
    ),
    output_masks=rhythmmakertools.BooleanPatternInventory(
        (
            rhythmmakertools.SustainMask(
                indices=(1,),
                invert=True,
            ),
            rhythmmakertools.SilenceMask(
                indices=(-1,),
            ),
            rhythmmakertools.NullMask(
                indices=(0,),
            ),
        )
    ),
    tieSpecifier=rhythmmakertools.TieSpecifier(
        tie_across_divisions=True,
    ),
)

```

Arbitrarily-nested values can also be replaced by specifying names delimited by double underscores. Here the denominator of the talea can be reconfigured directly, changing it from the previous value of 8 to 4:

```

>>> new_rhythm_maker = new(new_rhythm_maker,
...     talea__denominator=4,
...     )
>>> print(format(new_rhythm_maker))
rhythmmakertools.TaleaRhythmMaker(
    talea=rhythmmakertools.Talea(
        counts=(2, 1),
        denominator=4,
    ),
    extra_counts_per_division=(1, 0, 1, 2, 0, 1),
    beamSpecifier=rhythmmakertools.BeamSpecifier(
        beam_each_division=True,
        beam_divisions_together=True,
        use_feather_beams=False,
    ),
    output_masks=rhythmmakertools.BooleanPatternInventory(
        (
            rhythmmakertools.SustainMask(
                indices=(1,),
                invert=True,
            ),
            rhythmmakertools.SilenceMask(
                indices=(-1,),
            ),
            rhythmmakertools.NullMask(
                indices=(0,),
            ),
        )
    ),
    tieSpecifier=rhythmmakertools.TieSpecifier(
        tie_across_divisions=True,
    ),
)

```

Storage formatting is used throughout Abjad and tools extending Abjad for a number of reasons. For one, it provides a full description of a potentially complex object’s configuration in as human-readable a format as possible. Storage formats are deterministic; any storage-formattable object with a given configuration is guaranteed to format in one and only one way. Non-trivial storage formats also take up multiple lines, making them well-suited to line-based version control systems, such as Subversion, Git and Mercurial – among many others –, discussed in subsection 5.3.2. Most importantly, the storage format for any storage-formattable object is guaranteed to be evaluable as code in Abjad’s namespace. Any object returned by evaluating another object’s storage format as a string will be configured identically to the original and compare equally. This is a kind of human-readable *serialization*. For example, the selector created previously in subsection 2.6.3 can be persisted via its storage format and recreated by evaluating that storage format in Abjad’s namespace. The new selector compares equally to the old:

```

>>> print(format(selector))
selectortools.Selector(
    callbacks=(
        selectortools.PrototypeSelectorCallback(
            prototype=scoretools.Leaf,
            flatten=True,
        ),
        selectortools.LogicalTieSelectorCallback(
            flatten=True,
            pitched=True,
            trivial=True,
        ),
        selectortools.DurationSelectorCallback(
            duration=selectortools.DurationInequality(
                operator_string='>',
                duration=durationtools.Duration(1, 4),
            ),
        ),
        selectortools.ItemSelectorCallback(
            item=0,
            apply_to_each=True,
        ),
        selectortools.FlattenSelectorCallback(
            depth=-1,
        ),
    ),
)
>>> selector_format = format(selector)
>>> evaluated_selector = eval(selector_format)
>>> evaluated_selector == selector
True

```

Importantly, storage formatting affords persistence to disk. Abjad's *persistence agent* – conceptually similar to the inspection agent, iteration agent and others – can write any storage-formattable object to disk as an importable Python module, automatically including the appropriate library import statements at the top of the file.

```

>>> persistence_agent = persist(selector)
>>> persistence_agent.as_module(
...     module_file_path='a_selector_module.py',
...     object_name='persisted_selector',
... )

```

The persisted module file can then be read. Note the various import statements at the top – durationtools, scoretools, selectortools – which have been determined by introspecting the selector:

```

>>> with open('a_selector_module.py', 'r') as module_file_pointer:
...     module_file_contents = module_file_pointer.read()
...

```

```

>>> print(module_file_contents)
# -*- encoding: utf-8 -*-
from abjad.tools import durationtools
from abjad.tools import scoretools
from abjad.tools import selectortools


persisted_selector = selectortools.Selector(
    callbacks=(
        selectortools.PrototypeSelectorCallback(
            prototype=scoretools.Leaf,
            flatten=True,
        ),
        selectortools.LogicalTieSelectorCallback(
            flatten=True,
            pitched=True,
            trivial=True,
        ),
        selectortools.DurationSelectorCallback(
            duration=selectortools.DurationInequality(
                operator_string='>',
                duration=durationtools.Duration(1, 4),
            ),
        ),
        selectortools.ItemSelectorCallback(
            item=0,
            apply_to_each=True,
        ),
        selectortools.FlattenSelectorCallback(
            depth=-1,
        ),
    ),
)

```

The persisted selector can also be imported from the persisted module, like any other importable Python code. It too compares equally to the original selector:

```

>>> from a_selector_module import persisted_selector
>>> persisted_selector == selector
True

```

Storage-formatting, templating and persistence are crucial for working with Abjad at a high-level. Many of the compositionally-interesting devices afforded by Abjad – objects which describe compositional processes, transformations, factories and the like – involve a considerable amount of configuration. Templating affords variation in these situations, allowing composers to create new objects from old, varying as few or as many aspects of the object under consideration as they wish, but saving them from the work of having to define a new object from scratch. Likewise, disk persistence allows composers to easily save their work.

## 2.8 OTHER TOOLS

Abjad provides many other tools for working with notation and modeling musical concepts, too many to discuss in full here. A wide variety of Abjad's classes for modeling time, rhythm and meter are explored in chapter 3, along with various extensions implemented in the Consort composition-modeling library, but others bear mentioning now.

### 2.8.1 MUTATION

Abjad's *mutation agent* – similar again its inspection, iteration and persistence agents – affords techniques for *destructively* transforming already-instantiated score objects. Accessible via the top-level `mutate()` function, the mutation agent can split, transpose and replace components within a score. For example, the leaves in a score can be split cyclically, every other split shard transposed by a minor-second, and slurs and articulations attached to each shard:

```
>>> staff = Staff("c'4 d'4 e'4 f'4 g'4 a'4 b'4 c''4")
>>> agent = mutate(staff[:])
>>> for i, shard in enumerate(agent.split(durations=[(3, 16)], cyclic=True)):
...     if i % 2:
...         mutate(shard).transpose('m2')
...     if 1 < len(shard):
...         attach(Slur(), shard)
...         attach(Articulation('accent'), shard[0])
...
>>> show(staff)
```



### 2.8.2 PITCH MODELING

Abjad provides a rich model of pitch, making explicit distinctions between named pitches – C4, Db5 – and numbered pitches – 0, 13 – as well as providing concrete models of named and numbered pitch-classes, intervals, interval-classes, pitch-ranges, octaves, accidentals, and all manner of specialized collection classes for these pitch objects, including pitch sets and interval-class vectors, each with transformation methods pertinent to their domain.

```
>>> pitch_segment = pitchtools.PitchSegment("c' d' e' fs' gf'")
>>> pitch_segment
PitchSegment(["c'", "d'", "e'", "fs'", "gf'"])

>>> pitch_segment = pitch_segment.rotate(1, transpose=True)
>>> pitch_segment
PitchSegment(["c'", 'fs', 'gs', 'as', 'bs'])
```

```

>>> pitch_segment[2].accidental
Accidental('s')

>>> pitch_segment[2].octave
Octave(3)

>>> pitch_segment[2].named_pitch_class
NamedPitchClass('gs')

>>> pitchtools.IntervalSegment(pitch_segment)
IntervalSegment(['-dim5', '+M2', '+M2', '+M2'])

```

Notably, Abjad's note-head objects – aggregated into its notes and chords – always store their pitch information as named pitches, guaranteeing an internal representation as close to the notational output as possible.

### 2.8.3 PARSERS

Abjad comes equipped with a number of parsers. Most notably, Abjad implements a parser for LilyPond's syntax, and can understand much – although not all – of LilyPond's grammar. Any time a note, rest, chord, container or even textual markup is instantiated with a string, the LilyPond parser is at work. The top-level `parse()` function provides access to the this parser:

```

>>> string = r"\new Staff { c'4 ( \p \l d'4 e'4 f'4 ) \! }"
>>> result = parse(string)
>>> show(result)

```



The LilyPond parser can also be instantiated by hand and called:

```

>>> string = r'''\\new Staff { c' _ \\markup { X \\bold Y \\italic Z "a b c" } }'''
>>> parser = lilypondparser.tools.LilyPondParser()
>>> result = parser(string)
>>> show(result)

```



Abjad also implements a simple parser for RTM-like *rhythm-tree* syntax which parses strings into rhythm-tree objects which can then be converted into proper tuplets:

```

>>> rtm_parser = rhythmtree.tools.RhythmTreeParser()
>>> rtm_syntax = '(1 (1 (2 (1 -1 1)) -2))'
>>> rtm_container = rtm_parser(rtm_syntax)[0]
>>> abjad_container = rtm_container(Duration(1, 2))[0]
>>> show(abjad_container)

```



## 2.8.4 NOTATION FACTORIES

Composers can certainly create every object in a score by hand, instantiating each note, chord or measure one-at-a-time. However, with experience, when one is already working with notation computationally one tends to want means of generating many score objects at once. Abjad provides a number of *factory* classes and functions for generating arbitrarily large amounts of notation. The simplest such factory is the `scoretools.make_notes()` function, which combines sequences of pitches and durations in a patterned way to create a selection of leaves:

```
>>> pitches = ["c'", "ef'", "g'", 'b']
>>> durations = [(3, 8), (5, 16), (1, 4)] * 3
>>> notes = scoretools.make_notes(pitches, durations)
>>> staff = Staff(notes)
>>> show(staff)
```



Abjad's *rhythm-maker* family of classes – discussed in depth in section 3.5 – provide a collection of highly-customizable factories for generating rhythms:

```
>>> rhythm_maker = rhythmmakertools.TaleaRhythmMaker(
...     talea=rhythmmakertools.Talea(
...         counts=[1, 2, 3, 4],
...         denominator=16,
...         ),
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tie_across_divisions=True,
...         ),
...     )
>>> divisions = [(3, 8), (5, 8), (2, 8), (3, 8), (3, 8)]
>>> rhythm = rhythm_maker(divisions)
>>> staff = Staff(rhythm)
>>> show(staff)
```



Abjad's *score template* classes act as factories for scores, fully populated with voices, staves, staff groups, and potentially clefs and instruments, but devoid of any “count-time” components – notes, rests, chords, tuplets or measures

– which comprise musical content. Score templates play a large role in the work described in later chapters, especially chapter 4, as they provide a way for multiple score-generating processes to synchronize on a common score structure. For example, `templatetools` provides a score template class for generating string quartet scores:

```
>>> score_template = templatetools.StringQuartetScoreTemplate()
>>> score = score_template()
>>> print(format(score))
\context Score = "String Quartet Score" <<
    \context StaffGroup = "String Quartet Staff Group" <<
        \context Staff = "First Violin Staff" {
            \clef "treble"
            \set Staff.instrumentName = \markup { Violin }
            \set Staff.shortInstrumentName = \markup { Vn. }
            \context Voice = "First Violin Voice" {
            }
        }
        \context Staff = "Second Violin Staff" {
            \clef "treble"
            \set Staff.instrumentName = \markup { Violin }
            \set Staff.shortInstrumentName = \markup { Vn. }
            \context Voice = "Second Violin Voice" {
            }
        }
        \context Staff = "Viola Staff" {
            \clef "alto"
            \set Staff.instrumentName = \markup { Viola }
            \set Staff.shortInstrumentName = \markup { Va. }
            \context Voice = "Viola Voice" {
            }
        }
        \context Staff = "Cello Staff" {
            \clef "bass"
            \set Staff.instrumentName = \markup { Cello }
            \set Staff.shortInstrumentName = \markup { Vc. }
            \context Voice = "Cello Voice" {
            }
        }
    >>
>>
```

## 2.9 CONCLUSION

This chapter discussed the core components and concepts for working with notation in Abjad. Composers aggregate score components, consisting of leaves and containers, into nested, hierarchical score structures. Indicators and spanners attach to the components in the tree, adding additional structuration, annotation or typographic embellishment. Iteration and selection then provide means for examining the constructed score tree, allowing composers to locate components within the growing composition for examination, mutation or decoration. The chapters

that follow build on these techniques, and describe increasingly high-level means of organizing time and generating notation en masse with Abjad.

This page intentionally left blank.

# 3

## Modeling time, rhythm and meter



ONSORT IS AN OPEN-SOURCE PYTHON PACKAGE extending Abjad and implementing a model of composition which relies on a number of interrelated but distinct approaches to working with musical time. For example, Abjad's *timespans* suggests a “coarse” approach to musical time. These objects represent arbitrary durated events on a timeline, without respect for score hierarchy or meter. They are well-suited for modeling large-scale phrasing and gestural density structures and can be transformed through splitting, scaling, stretching, and translation among various other manipulations. Because they have no notational reality, they can model temporal concepts unsuited to notated music – without introducing additional complexity–, such as arbitrarily overlapping events. Timespans may also be annotated, allowing composers to position metadata anywhere on a timeline, much like arranging audio regions in a DAW.<sup>15</sup> Moreover, every durated object in a score hierarchy can be described as a timespan, allowing score components to engage in abstract time relations. In contrast, no-

tated rhythms, composed of note, rest, tie and tuplet objects – among others–, provide the most “fine-grained” approach to musical time. While incredibly expressive, fully notated rhythms are potentially complex to create, and must ultimately be anchored in a score hierarchy. Highly-configurable *rhythm-maker* objects ameliorate the complexity of creating notated rhythms by providing a high-level interface to the process of rhythm generation. Abjad’s hierarchical model of *meter* coordinates time and rhythm vertically across different levels of depth in the score tree, and bridges Consort’s coarse and fine stages of rhythmic interpretation. Meter sequences can be generated from timespan-based phrase structures, and those meter sequences used to transform notated rhythms in turn. A thorough discussion of the implementation of these time models and their implications will clarify an analysis in chapter 4 of Consort’s *score interpretation stage*.

### 3.1 TIMESPANS

*Timespans* model left-closed / right-open intervals of time positioned absolutely along a timeline. Every timespan describes a range of offsets  $x$  starting with – and including – some start offset  $A$  and leading up to – but *not* including – some stop offset  $B$ :

$$A \leq x < B \quad (3.1)$$

Note that all leaves in a score describe such half-bounded intervals of time. Adjacent notes in a score do not overlap but rather abut one another because their timespans do not overlap. In fact, every durated object in a score – every note, chord, rest, measure, staff, and even the score itself – can be described in terms of timespans. Yet, while score objects can always be expressed as timespans, those timespans themselves do not – by definition – refer to any score objects. Abjad implements timespans as immutable constants, much like Abjad’s Pitch and Duration objects, and similarly to Python’s implementation of numbers and strings. Constancy allows timespans to avoid a variety of computational reference problems. For example, multiple objects can reference the same timespan without fear of that timespan changing state, much as multiple objects can reference the integer 11 without fear that that same integer will change into the integer 5.

Abjad implements timespans via the `Timespan` class in its `timespantools` library. The following code shows the definition of a timespan object beginning at the offset  $\frac{1}{4}$  and continues up until the offset  $\frac{3}{2}$ :

```
>>> timespan = timespantools.Timespan(
...     start_offset=Offset(1, 4),
```

```
| ...     stop_offset=Offset(3, 2),  
| ... )
```

Like many objects in Abjad, `timespan` can be formatted for human-readable textual inspection, or displayed as a graphic illustration:

```
| >>> print(format(timespan))  
timespantools.Timespan(  
    start_offset=durationtools.Offset(1, 4),  
    stop_offset=durationtools.Offset(3, 2),  
)
```

```
| >>> show(timespan)
```



The `Timespan` class provides a large number of methods and properties for inspecting timespans, comparing them to other timespans or offsets, and for operating on timespans to generate new timespans. Once instantiated, a `timespan` can be examined for its start offset, stop offset and duration. Because of the `Timespan` class' immutability, these properties are read-only and therefore can only be accessed, but not changed:

```
| >>> timespan.start_offset  
Offset(1, 4)
```

```
| >>> timespan.stop_offset  
Offset(3, 2)
```

```
| >>> timespan.duration  
Duration(5, 4)
```

A `timespan`'s start offset must be equal to or less than its stop offset. Timespans with identical start and stop offsets have a duration of 0 and effectively model a single time-point. Such timespans are considered not “well-formed”:

```
| >>> timepoint_timespan = timespantools.Timespan(1, 1)  
| >>> timepoint_timespan.duration  
Duration(0, 1)
```

```
| >>> timepoint_timespan.is_well_formed  
False
```

Both the `start_offset` and `stop_offset` keywords to the `Timespan` class' initializer are optional, and default to Abjad's built-in rational constants `NegativeInfinity` and `Infinity` respectively. A `timespan` created without specifying either a start or stop offset effectively describes the `timespan` which encompasses all possible offsets in time:

```

>>> infinite_timespan = timespantools.Timespan()
>>> infinite_timespan.start_offset
NegativeInfinity

>>> infinite_timespan.stop_offset
Infinity

>>> infinite_timespan.duration
Infinity

```

By specifying only a start or stop offset, timespans can also be created which encompass the infinite set of offsets up until some stop offset, or the infinite set of offsets starting at and following some start-offset:

```

>>> timespantools.Timespan(stop_offset=0)
Timespan(start_offset=NegativeInfinity, stop_offset=Offset(0, 1))

>>> timespantools.Timespan(start_offset=0)
Timespan(start_offset=Offset(0, 1), stop_offset=Infinity)

```

Timespan objects also partake in Abjad's templating regime. New timespans can be created from old ones through the use of the top-level new() function:

```

>>> new(timespan, stop_offset=(5, 16))
Timespan(start_offset=Offset(1, 4), stop_offset=Offset(5, 16))

```

Note that the above timespans have been configured from input in various formats, such as explicit Offset objects, integers, and even numerator/denominator pairs. Timespans always coerce arguments given for their start\_offset and stop\_offset properties into explicit Offset instances. Abjad's other classes and functions implement similar offset, duration, and even pitch-coercion behavior pervasively.

### 3.1.1 ANNOTATED TIMESPANS

While the Timespan class only has two configurable properties – its start offset and stop offset –, subclassing allows for the creation of new classes with the same core functionality as Timespan but extended to support new behaviors and configurations. As an example, Abjad's timespantools library provides an AnnotatedTimespan class which subclasses Timespan but adds a third read-only annotation property, which can be configured with any arbitrary object:

```

>>> annotated_timespan = timespantools.AnnotatedTimespan(
...     start_offset=(1, 8),
...     stop_offset=(7, 8),
...     annotation='Any arbitrary object can act as an annotation.')
...
>>> annotated_timespan.annotation
'Any arbitrary object can act as an annotation.'

```

Annotated timespans allow composers to position annotations or any other metadata along a timeline. If the annotation object is itself a mutable data structure such as a Python list or dictionary, that annotation can be used to store increasing amounts of information during the compositional process. Additionally, Abjad's top-level new() function can be used to template new annotated timespans from old ones, replacing one annotation with another while preserving temporal information. If that same annotation object supports templating, nested reconfiguration can be performed:

```
>>> metadata_timespan = new(annotated_timespan,
...     stop_offset=(3, 2),
...     annotation={
...         'durations': ((1, 8), (1, 8), (3, 16)),
...         'dynamic': indicatortools.Dynamic('ppp'),
...         'pitch_segment': pitchtools.PitchSegment([0, 1, 4, 7]),
...         },
...     )
>>> metadata_timespan.annotation['bow_contact_point'] = Multiplier(1, 3)
>>> print(format(metadata_timespan))
timespantools.AnnotatedTimespan(
    start_offset=durationtools.Offset(1, 8),
    stop_offset=durationtools.Offset(3, 2),
    annotation={
        'bow_contact_point': durationtools.Multiplier(1, 3),
        'durations': (
            (1, 8),
            (1, 8),
            (3, 16),
        ),
        'dynamic': indicatortools.Dynamic(
            name='ppp',
        ),
        'pitch_segment': pitchtools.PitchSegment(
            (
                pitchtools.NumberedPitch(0),
                pitchtools.NumberedPitch(1),
                pitchtools.NumberedPitch(4),
                pitchtools.NumberedPitch(7),
            ),
            item_class=pitchtools.NumberedPitch,
        ),
    },
)
```

Other Timespan subclasses are possible, allowing for even more configurable properties, as well as new methods. Two Timespan subclasses discussed later, `consort.PerformedTimespan` and `consort.SilentTimespan`, are core components in Consort's score interpretation stage.

### 3.1.2 TIME RELATIONS

*Time relations* model the disposition of one timespan relative to another timespan or offset. These relationships include intersection, containment, simultaneous start offsets or stop offsets, and many others. Abjad's `timespantools` library provides a `TimeRelation` class and a collection of factory methods for configuring `TimeRelation` instances which formalize all possible dispositions of a timespan relative another timespan or offset<sup>1</sup>. Time relations may be configured with or without reference to any timespans or offsets at all, allowing for the possibility of modeling a purely abstract time relationship:

```
>>> time_relation_1 = timespantools.timespan_2_intersects_timespan_1()
>>> print(format(time_relation_1))
timespantools.TimespanTimespanTimeRelation(
    inequality=timespantools.CompoundInequality(
        [
            timespantools.CompoundInequality(
                [
                    timespantools.SimpleInequality('timespan_1.start_offset <= timespan_2.start_offset'),
                    timespantools.SimpleInequality('timespan_2.start_offset < timespan_1.stop_offset'),
                ],
                logical_operator='and',
            ),
            timespantools.CompoundInequality(
                [
                    timespantools.SimpleInequality('timespan_2.start_offset <= timespan_1.start_offset'),
                    timespantools.SimpleInequality('timespan_1.start_offset < timespan_2.stop_offset'),
                ],
                logical_operator='and',
            ),
        ],
        logical_operator='or',
    ),
)
```

---

<sup>1</sup>The thirty-three time relation factory functions contained in `timespantools` are `offset_happens_after_timespan_starts()`, `offset_happens_after_timespan_stops()`, `offset_happens_before_timespan_starts()`, `offset_happens_before_timespan_stops()`, `offset_happens_during_timespan()`, `offset_happens_when_timespan_starts()`, `offset_happens_when_timespan_stops()`, `timespan_2_contains_timespan_1_improperly()`, `timespan_2_curtails_timespan_1()`, `timespan_2_delays_timespan_1()`, `timespan_2_happens_during_timespan_1()`, `timespan_2_intersects_timespan_1()`, `timespan_2_is_congruent_to_timespan_1()`, `timespan_2_overlaps_all_of_timespan_1()`, `timespan_2_overlaps_only_start_of_timespan_1()`, `timespan_2_overlaps_only_stop_of_timespan_1()`, `timespan_2_overlaps_start_of_timespan_1()`, `timespan_2_overlaps_stop_of_timespan_1()`, `timespan_2_starts_after_timespan_1_starts()`, `timespan_2_starts_after-timespan_1_stops()`, `timespan_2_starts_before_timespan_1_starts()`, `timespan_2_starts_before-timespan_1_stops()`, `timespan_2_starts_during_timespan_1()`, `timespan_2_starts_when_timespan_1_starts()`, `timespan_2_starts_when-timespan_1_stops()`, `timespan_2_stops_after_timespan_1_starts()`, `timespan_2_stops_after-timespan_1_stops()`, `timespan_2_stops_before_timespan_1_starts()`, `timespan_2_stops_before-timespan_1_stops()`, `timespan_2_stops_during_timespan_1()`, `timespan_2_stops_when_timespan_1_starts()`, `timespan_2_stops_when-timespan_1_stops()` and `timespan_2_trisects_timespan_1()`.

The above intersection relationship between the timespans  $[a, b)$  and  $[c, d)$  can be described more pithily by the following predicate:

$$(a \leq c \wedge c < b) \vee (c \leq a \wedge a < d) \quad (3.2)$$

A “half-configured” time relation is also possible. Such an object acts as a kind of “frozen” predicate. Calling the time relation as though it was a function<sup>2</sup> on another timespan or offset returns a truth value:

```
>>> time_relation_2 = timespantools.timespan_2_intersects_timespan_1(  
...     timespan_1=timespantools.Timespan(0, 10),  
...     )  
>>> time_relation_2(timespan_2=timespantools.Timespan(5, 15))  
True  
  
>>> time_relation_2(timespan_2=timespantools.Timespan(30, 45))  
False
```

Providing two timespans – or one timespan and an offset, as required – to one of the various time relation factory functions found in `timespantools` will configure and then immediately evaluate the generated time relation object, allowing the factory function itself to behave as a predicate:

```
>>> timespantools.timespan_2_intersects_timespan_1(  
...     timespan_1=timespantools.Timespan(1, 3),  
...     timespan_2=timespantools.Timespan(2, 4),  
...     )  
True
```

The time relation factory functions in `timespantools` are mirrored as methods on the `Timespan` class itself, allowing composers to determine the various relations of any timespan relative any other timespan or offset in an object-oriented fashion. The `Timespan` object automatically “fills in” the `timespan_1` argument to the `TimeRelation` with a reference to itself, and can pass the optional argument to its method call as the other object in the relation, allowing for the immediate evaluation of the relation as either true or false.

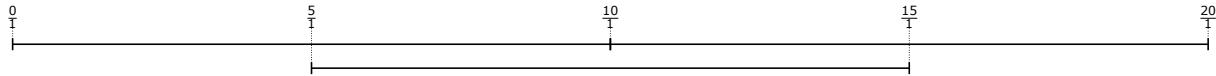
Consider the following three timespans:

```
>>> timespan_1 = timespantools.Timespan(0, 10)  
>>> timespan_2 = timespantools.Timespan(5, 15)  
>>> timespan_3 = timespantools.Timespan(10, 20)
```

---

<sup>2</sup>Any class in Python can be made treatable as a function by implementing a `__call__()` method.

After being collected into a *timespan inventory*, discussed further in section 3.2, the three timespans can be illustrated. Note that the beginning and end of each timespan is demarcated by a short vertical line, and that only non-intersecting timespans are shown in each “row” in the illustration:



We can test for intersection between these three timespans via the `intersects_timespan()` method. Two timespans are considered to intersect if any part of one timespan overlaps any part of another. *Intersection* is therefore commutative. Note that `timespan_1` and `timespan_3` do not overlap even though they share the offset 10. As discussed earlier, timespans are left-closed / right-open, meaning that while their start offset is contained in the infinite set of offsets they range over, their stop offset is not:

```
>>> timespan_1.intersects_timespan(timespan_2)
True

>>> timespan_1.intersects_timespan(timespan_3)
False

>>> timespan_2.intersects_timespan(timespan_1)
True

>>> timespan_2.intersects_timespan(timespan_3)
True

>>> timespan_3.intersects_timespan(timespan_1)
False

>>> timespan_3.intersects_timespan(timespan_2)
True
```

*Congruency* tests whether two timespans share the same start and stop offset. Every timespan is congruent with itself:

```
>>> timespan_1.is_congruent_to_timespan(timespan_2)
False

>>> timespan_1.is_congruent_to_timespan(timespan_1)
True
```

*Tangency* tests whether one timespan’s stop offset is the same as another timespan’s start offset, or vice versa. Tangency can be used to determine if a sorted collection of timespans is both entirely contiguous and non-overlapping:

```
>>> timespan_1.is_tangent_to_timespan(timespan_2)
False
```

```
>>> timespan_1.is_tangent_to_timespan(timespan_3)
True
```

A wide variety of other time relation predicates are also possible, such as testing if a timespan intersects with a specific offset, testing if a timespan overlaps only the beginning or end of another timespan, or testing if a timespan contains another timespan entirely. These predicates make possible the many generative operations carried out on timespans.

### 3.1.3 OPERATIONS ON TIMESPANS

Many Timespan methods provide transformations, such as translation, scaling or offset rounding. Because timespans are immutable, these methods create a new timespan based on the old one and then return the new, leaving the old exactly as it was:

```
>>> timespan = timespantools.Timespan(0, 15)
>>> timespan.translate(3)
Timespan(start_offset=Offset(3, 1), stop_offset=Offset(18, 1))

>>> timespan.scale(3)
Timespan(start_offset=Offset(0, 1), stop_offset=Offset(45, 1))

>>> timespan.round_offsets(2)
Timespan(start_offset=Offset(0, 1), stop_offset=Offset(16, 1))
```

These generative methods are implemented internally in terms of templating via `new()`, as described in section 3.1, allowing annotated timespan subclasses to partake in the same generative functionality – reconfiguring their start and / or stop offsets, but maintaining all other previously configured properties:

```
>>> annotated_timespan = timespantools.AnnotatedTimespan(0, 5, 'an annotation')
>>> scaled_annotated_timespan = annotated_timespan.translate((-1, 3))
>>> print(format(scaled_annotated_timespan))
timespantools.AnnotatedTimespan(
    start_offset=durationtools.Offset(-1, 3),
    stop_offset=durationtools.Offset(14, 3),
    annotation='an annotation',
)
```

Some generative operations may return zero or more timespans, aggregated in a data structure called a *timespan inventory*, discussed at length in section 3.2. Splitting a timespan by an offset is one such operation. If the timespan to

be split *properly contains*<sup>3</sup> the splitting offset, a timespan inventory containing two new timespans will be returned.

Otherwise, the split operation will return a timespan inventory containing a copy of the original input timespan:

```
>>> two_shards = timespan.split_at_offset(5)
>>> print(format(two_shards))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(5, 1),
        ),
        timespantools.Timespan(
            start_offset=durationtools.Offset(5, 1),
            stop_offset=durationtools.Offset(15, 1),
        ),
    ]
)

>>> one_shard = timespan.split_at_offset(10000)
>>> print(format(one_shard))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(15, 1),
        ),
    ]
)
```

More complex operations between timespans resulting in timespan inventories include subtraction and the logical operations AND, OR and XOR. These generative operations are conceptually *set* operations – union, difference, intersection, symmetric difference, etc. – performed on the two sets of offsets contained by the two timespan operands. Consider these same set operations – union, difference, intersection, symmetric difference – carried out on trivial sets in Python:

```
>>> set([1, 2, 3]) | set([2, 3, 4])  # union
set([1, 2, 3, 4])

>>> set([1, 2, 3]) - set([2, 3, 4])  # difference
set([1])

>>> set([1, 2, 3]) & set([2, 3, 4])  # intersection
set([2, 3])
```

<sup>3</sup>Proper containment of an offset means that the offset is greater than the timespan's start offset and less than the timespan's stop offset. Improper containment would indicate the offset is greater than or equal to the timespan's start offset and less than or equal to its stop offset.

```
>>> set([1, 2, 3]) ^ set([2, 3, 4]) # symmetric difference
set([1, 4])
```

Set operations performed on timespans are conceptually identical, but operate on infinite but bounded sets of offsets instead of discrete sets of objects such as integers. For example, subtracting one timespan from another computes the set difference of the offsets contained by both. This operation is not commutative – subtracting one timespan from another will not result in the same output as subtracting the latter from the former. Subtracting a timespan from itself always results in the empty set of offsets: no timespan at all:

```
>>> result = timespantools.Timespan(0, 10) - timespantools.Timespan(0, 10)
>>> print(format(result))
timespantools.TimespanInventory(
[])
)

>>> result = timespantools.Timespan(0, 10) - timespantools.Timespan(5, 15)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(5, 1),
    ),
]
)

>>> result = timespantools.Timespan(0, 10) - timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(10, 1),
    ),
]
)

>>> result = timespantools.Timespan(5, 15) - timespantools.Timespan(0, 10)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(10, 1),
        stop_offset=durationtools.Offset(15, 1),
    ),
]
)

>>> result = timespantools.Timespan(5, 15) - timespantools.Timespan(5, 15)
>>> print(format(result))
```

```

timespantools.TimespanInventory(
    []
)

>>> result = timespantools.Timespan(5, 15) - timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(5, 1),
            stop_offset=durationtools.Offset(10, 1),
            ),
    ]
)

>>> result = timespantools.Timespan(10, 20) - timespantools.Timespan(0, 10)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(10, 1),
            stop_offset=durationtools.Offset(20, 1),
            ),
    ]
)

>>> result = timespantools.Timespan(10, 20) - timespantools.Timespan(5, 15)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(15, 1),
            stop_offset=durationtools.Offset(20, 1),
            ),
    ]
)

>>> result = timespantools.Timespan(10, 20) - timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
    []
)

```

Computing the logical OR of two timespans results in an offset set union – a commutative operation, effectively fusing the timespans together if they overlap:

```

>>> result = timespantools.Timespan(0, 10) | timespantools.Timespan(5, 15)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(15, 1),
            ),
    ]
)

```

```

        stop_offset=durationtools.Offset(15, 1),
    ),
]
)

>>> result = timespantools.Timespan(0, 10) | timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(20, 1),
    ),
]
)

>>> result = timespantools.Timespan(5, 15) | timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(5, 1),
        stop_offset=durationtools.Offset(20, 1),
    ),
]
)

```

Unioning guarantees that all of the offsets contained in the two input timespans will appear in the output timespan or timespans, whether or not any overlap occurred:

```

>>> result = timespantools.Timespan(10, 20) | timespantools.Timespan(25, 50)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(10, 1),
        stop_offset=durationtools.Offset(20, 1),
    ),
    timespantools.Timespan(
        start_offset=durationtools.Offset(25, 1),
        stop_offset=durationtools.Offset(50, 1),
    ),
]
)

```

The logical AND – set intersection – results in the intersection of the two input timespans. Only those offsets which occur in both timespan operands will occur in the output timespan, if any:

```

>>> result = timespantools.Timespan(0, 10) & timespantools.Timespan(5, 15)
>>> print(format(result))

```

```

timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(5, 1),
            stop_offset=durationtools.Offset(10, 1),
            ),
        ],
    )
)

>>> result = timespantools.Timespan(0, 10) & timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
    []
)

>>> result = timespantools.Timespan(5, 15) & timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(10, 1),
            stop_offset=durationtools.Offset(15, 1),
            ),
        ],
    )
)

```

Logical XOR – also known as exclusive OR – results in the symmetric difference of the two input timespans. Only those offsets which are contained by only one of the two input timespans will occur in the output:

```

>>> result = timespantools.Timespan(0, 10) ^ timespantools.Timespan(5, 15)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(5, 1),
            ),
        timespantools.Timespan(
            start_offset=durationtools.Offset(10, 1),
            stop_offset=durationtools.Offset(15, 1),
            ),
        ],
    )
)

>>> result = timespantools.Timespan(0, 10) ^ timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(10, 1),
            ),
        ],
    )
)

```

```

timespantools.Timespan(
    start_offset=durationtools.Offset(10, 1),
    stop_offset=durationtools.Offset(20, 1),
),
],
)

>>> result = timespantools.Timespan(5, 15) ^ timespantools.Timespan(10, 20)
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(5, 1),
        stop_offset=durationtools.Offset(10, 1),
    ),
    timespantools.Timespan(
        start_offset=durationtools.Offset(15, 1),
        stop_offset=durationtools.Offset(20, 1),
    ),
]
)

```

These operations, while perhaps initially rather abstract, are incredibly powerful and artistically useful. They afford composers with the procedural building blocks to mask temporal objects with one another, fuse them together, or create lacunae. When extended to operate on many timespans at once, wholesale transformations on massed timespans becomes practical.

## 3.2 TIMESPAN INVENTORIES

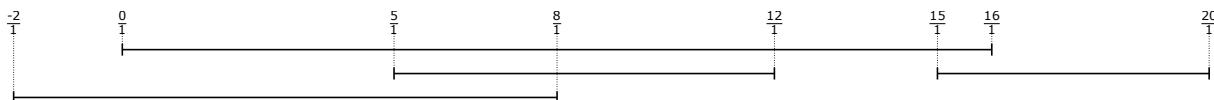
Abjad provides a `TimespanInventory` class specifically for aggregating together a collection of `Timespan` objects.<sup>4</sup> `Timespan` inventories implement Python’s *mutable sequence protocol* which allow them to behave exactly like lists, supporting appension, extension, insertion, indexing, sorting, iteration and other procedures pertinent to list-like objects. They also provide a wide variety of properties and methods for interacting with massed groups of timespans such as searching for timespans matching a time relation, splitting all timespans which intersect with a given offset, mapping a logical operation against all timespans in the inventory, or partitioning an inventory containing overlapping timespans into multiple separate inventories containing non-overlapping timespans.

---

<sup>4</sup>Consort provides its own timespan collection class – the `TimespanCollection`. This class stores timespans internally not in a list, but in a balanced *interval tree*.<sup>7</sup> An interval tree is an augmented, self-balancing binary tree which stores start offsets and stop offsets. Such a data structure guarantees its contents are always sorted, and allows for highly optimized lookups for timespan matching various search criteria. The `TimespanCollection` class is used at crucial points during Consort’s interpretation stage simply for purposes of speed, and should be considered an implementation detail. It provides only a few methods, specifically for affording rapid search and retrieval of timespans intersecting other timespans or offsets. With work, its internal data structure may eventually be merged into Abjad’s own `TimespanInventory` class.

Like a Python list, a timespan inventory can be created with an iterable of zero or more timespans as an instantiation argument, be appended to, or extended into:

```
>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 16),
... ])
>>> timespan_inventory.append(timespantools.Timespan(5, 12))
>>> timespan_inventory.extend([
...     timespantools.Timespan(-2, 8),
...     timespantools.Timespan(15, 20),
... ])
>>> print(format(timespan_inventory))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(16, 1),
    ),
    timespantools.Timespan(
        start_offset=durationtools.Offset(5, 1),
        stop_offset=durationtools.Offset(12, 1),
    ),
    timespantools.Timespan(
        start_offset=durationtools.Offset(-2, 1),
        stop_offset=durationtools.Offset(8, 1),
    ),
    timespantools.Timespan(
        start_offset=durationtools.Offset(15, 1),
        stop_offset=durationtools.Offset(20, 1),
    ),
]
)
>>> show(timespan_inventory)
```



Timespan inventories can also be queried for their length, be indexed into, or iterated over like any other sequence-like object in Python:

```
>>> len(timespan_inventory)
4

>>> timespan_inventory[1]
Timespan(start_offset=Offset(5, 1), stop_offset=Offset(12, 1))

>>> for timespan in timespan_inventory:
...     timespan
...
```

```
Timespan(start_offset=Offset(0, 1), stop_offset=Offset(16, 1))
Timespan(start_offset=Offset(5, 1), stop_offset=Offset(12, 1))
Timespan(start_offset=Offset(-2, 1), stop_offset=Offset(8, 1))
Timespan(start_offset=Offset(15, 1), stop_offset=Offset(20, 1))
```

One timespan inventory can be created from another by passing the first timespan inventory as an instantiation argument to the second, just like one can create a Python list simply by calling `list()` on another iterable object, including another list. Unlike timespans, timespan inventories are mutable. As many of the operations implemented on timespan inventories mutate the inventory *in-place*, this instantiation pattern provides a simple means of “copying”, allowing composers to duplicate a timespan structure before operating on it, thereby preserving the original:

```
>>> duplicate = timespantools.TimespanInventory(timespan_inventory)
>>> duplicate == timespan_inventory
True

>>> duplicate is timespan_inventory
False

>>> print(format(duplicate))
timespantools.TimespanInventory(
    [
        timespantools.Timespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(16, 1),
        ),
        timespantools.Timespan(
            start_offset=durationtools.Offset(5, 1),
            stop_offset=durationtools.Offset(12, 1),
        ),
        timespantools.Timespan(
            start_offset=durationtools.Offset(-2, 1),
            stop_offset=durationtools.Offset(8, 1),
        ),
        timespantools.Timespan(
            start_offset=durationtools.Offset(15, 1),
            stop_offset=durationtools.Offset(20, 1),
        ),
    ]
)
```

Timespan inventories can be treated as timespans themselves, having a start offset equal to the minimum start offset of any of their contained timespans, and a stop offset equal to the maximum stop offset of any of their contained timespans. Their start and stop offset properties allow them to express a duration, as well as provide a concrete timespan representation. Because timespan inventories can be modeled as timespans, they can even be inserted into other timespan inventories, effectively masquerading as timespans:

```

>>> timespan_inventory.start_offset
Offset(-2, 1)

>>> timespan_inventory.stop_offset
Offset(20, 1)

>>> timespan_inventory.duration
Duration(22, 1)

>>> timespan_inventory.timespan
Timespan(start_offset=Offset(-2, 1), stop_offset=Offset(20, 1))

```

Because timespan inventories aggregate multiple timespans together, they also provide properties for describing collective qualities of those timespans. *Contiguity* tests if every timespan in the inventory is tangent to another timespan and also does not overlap any other timespan. *Overlap* tests if any timespan intersects any other timespan. *Well-formedness* tests that all timespans' durations are greater than 0. A timespan inventory whose timespans are contiguous is necessarily also non-overlapping:

```

>>> timespan_inventory.all_are_contiguous
False

>>> timespan_inventory.all_are_nonoverlapping
False

>>> timespan_inventory.all_are_well_formed
True

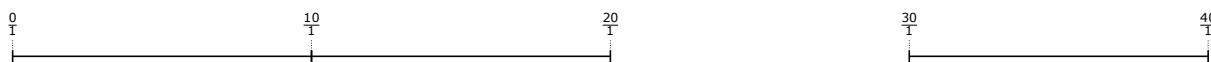
```

The following timespan inventory's timespans are non-overlapping but also non-contiguous:

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 10),
...     timespantools.Timespan(10, 20),
...     timespantools.Timespan(30, 40),
... ])
>>> show(timespan_inventory)

```



```

>>> timespan_inventory.all_are_contiguous
False

>>> timespan_inventory.all_are_nonoverlapping
True

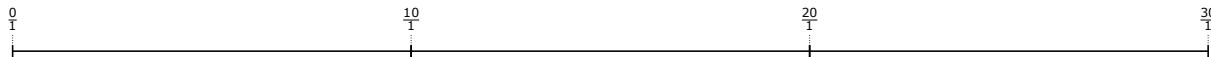
```

In contrast, this timespan inventory's timespans are both non-overlapping and contiguous:

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 10),
...     timespantools.Timespan(10, 20),
...     timespantools.Timespan(20, 30),
... ])
>>> show(timespan_inventory)

```



```

>>> timespan_inventory.all_are_contiguous
True

>>> timespan_inventory.all_are_nonoverlapping
True

```

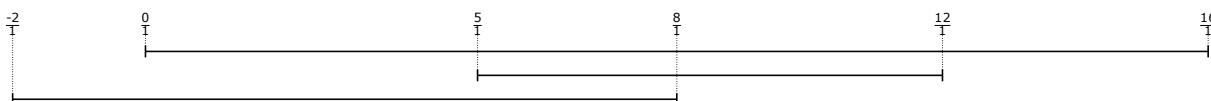
### 3.2.1 OPERATIONS ON TIMESPAN INVENTORIES

Timespan inventories implement unioning, differencing and splitting methods which parallel those implemented on timespans themselves. These methods map the desired operation onto the contents of the inventory by, for example, splitting every timespan contained in a given inventory by some offset. All of these operations act in place. The intersection of all of the timespans in a timespan inventory relative another timespan can be computed with the Python & operator, the same syntax used when operating on individual timespans, as demonstrated in subsection 3.1.3.<sup>5</sup>

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 16),
...     timespantools.Timespan(5, 12),
...     timespantools.Timespan(-2, 8),
... ])
>>> show(timespan_inventory)

```



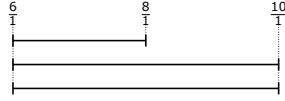
```

>>> timespan_operand = timespantools.Timespan(6, 10)
>>> result = timespan_inventory & timespan_operand
>>> show(result, range_=(-2, 16))

```

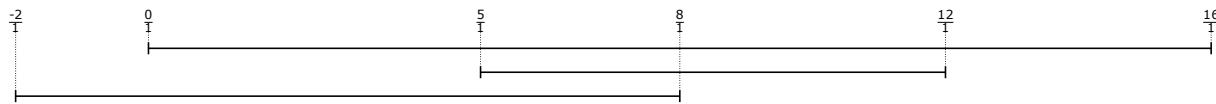
---

<sup>5</sup>The call to `show()` in the code example here contains the keyword argument `range_`. This argument controls the horizontal scaling and spacing of the timespan illustration, allowing different illustrations to be aligned against one another in a document even if the timespan inventories they illustrate have different start and stop offsets. The name `range_` – with a trailing underscore – is used instead of `range`, a commonly-used built-in Python function. It is common practice in the Python community to append underscores to names when they would otherwise conflict with reserved words in Python’s grammar or global built-in namespace. For example, one would use the names `break_`, `object_`, `set_` and `type_` instead of the reserved word `break`, or the built-in names `object`, `set` and `type`.



Likewise, the offsets bound by a given timespan can be subtracted from all of the timespans in a timespan inventory, effectively cutting a hole in that inventory's timeline:

```
>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 16),
...     timespantools.Timespan(5, 12),
...     timespantools.Timespan(-2, 8),
... ])
>>> show(timespan_inventory)
```



```
>>> timespan_operand = timespantools.Timespan(6, 10)
>>> result = timespan_inventory - timespan_operand
>>> show(result)
```



As with a single timespan and an offset, an entire timespan inventory can be split into two separate inventories via the `split_at_offset()` method:

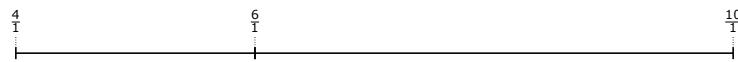
```
>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 3),
...     timespantools.Timespan(3, 6),
...     timespantools.Timespan(6, 10),
... ])
>>> show(timespan_inventory)
```



```
>>> left, right = timespan_inventory.split_at_offset(4)
>>> show(left, range_=(0, 10))
```



```
>>> show(right, range_=(0, 10))
```



The `TimespanInventory` class also provides the convenience method `split_at_offsets()` for splitting an inventory by an arbitrary number of offsets at once:

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 3),
...     timespantools.Timespan(3, 6),
...     timespantools.Timespan(6, 10),
... ])
>>> show(timespan_inventory)

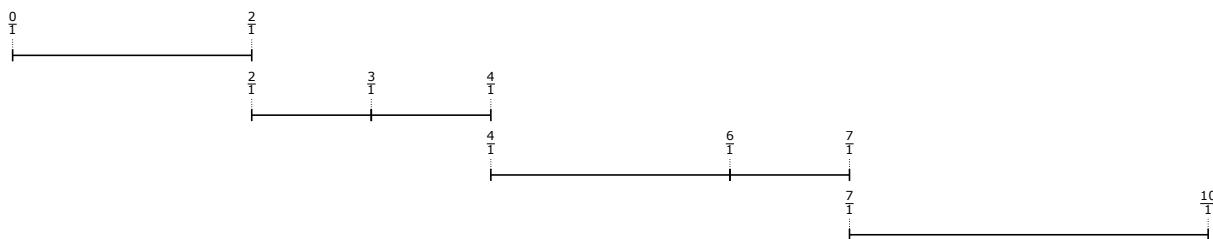
```



```

>>> shards = timespan_inventory.split_at_offsets((2, 4, 7))
>>> for shard in shards:
...     show(shard, range_=(0, 10))
...

```

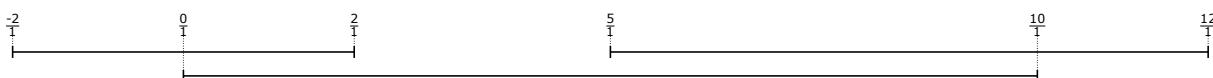


The mutating methods described above modify a timespan inventory by mapping some procedure against its contents and some outside timespan or offset. However, timespan inventories may also be modified by applying a procedure solely against the contents of the inventory itself, mapping each timespan in the collection against each other timespan in that collection. For example, a timespan inventory can be modified by computing the logical OR – the set union – of every timespan in the inventory relative every other timespan, effectively fusing all overlapping timespans together:

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(-2, 2),
...     timespantools.Timespan(0, 10),
...     timespantools.Timespan(5, 12),
... ])
>>> show(timespan_inventory)

```

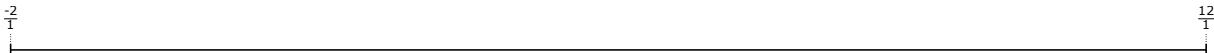


```

>>> result = timespan_inventory.compute_logical_or()
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(-2, 1),
        stop_offset=durationtools.Offset(12, 1),
    ),
]
)

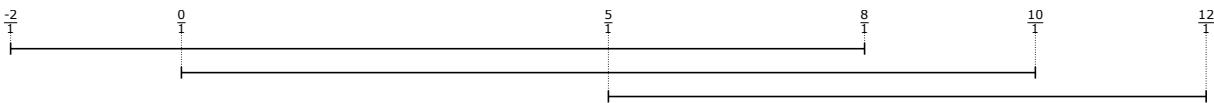
```

```
| >>> show(result)
```



A timespan inventory can also be modified by computing the logical `AND` of every timespan in the inventory relative every other timespan. This procedure leaves only those offsets where every single timespan overlaps:

```
>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(-2, 8),
...     timespantools.Timespan(0, 10),
...     timespantools.Timespan(5, 12),
... ])
>>> show(timespan_inventory)
```



```
>>> result = timespan_inventory.compute_logical_and()
>>> print(format(result))
timespantools.TimespanInventory(
[
    timespantools.Timespan(
        start_offset=durationtools.Offset(5, 1),
        stop_offset=durationtools.Offset(8, 1),
    ),
]
)
```

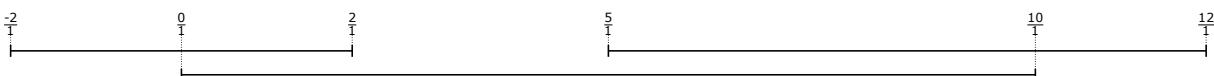
  

```
>>> show(result, range_=(-2, 12))
```



Lastly, computing the in-place logical `XOR` removes all overlap from the timespan inventory, leaving only those offsets occupied by only one timespan:

```
>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(-2, 2),
...     timespantools.Timespan(0, 10),
...     timespantools.Timespan(5, 12),
... ])
>>> show(timespan_inventory)
```



```
>>> result = timespan_inventory.compute_logical_xor()
>>> print(format(result))
timespantools.TimespanInventory(
[
```

```

timespantools.Timespan(
    start_offset=durationtools.Offset(-2, 1),
    stop_offset=durationtools.Offset(0, 1),
),
timespantools.Timespan(
    start_offset=durationtools.Offset(2, 1),
    stop_offset=durationtools.Offset(5, 1),
),
timespantools.Timespan(
    start_offset=durationtools.Offset(10, 1),
    stop_offset=durationtools.Offset(12, 1),
),
]
)

```

>>> show(result, range\_=(-2, 12))

Timespan *partitioning* separates a timespan inventory into groups of overlapping and optionally tangent timespans, aggregated into new timespan inventories. This procedure allows composers to isolate contiguous blocks of activity:

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(0, 10),
...     timespantools.Timespan(5, 15),
...     timespantools.Timespan(15, 20),
...     timespantools.Timespan(25, 30),
... ])
>>> show(timespan_inventory)


```

```

>>> for shard in timespan_inventory.partition():
...     show(shard, range_=(0, 30))
...

```

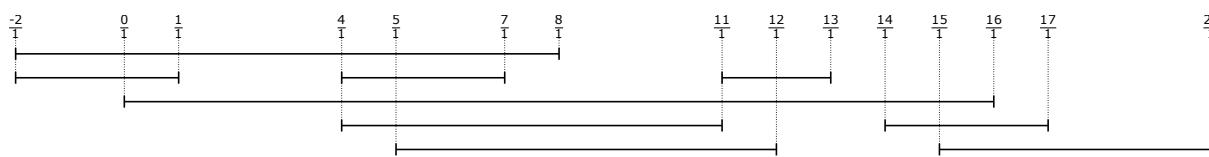
```

>>> for shard in timespan_inventory.partition(include_tangent_timespans=True):
...     show(shard, range_=(0, 30))
...

```

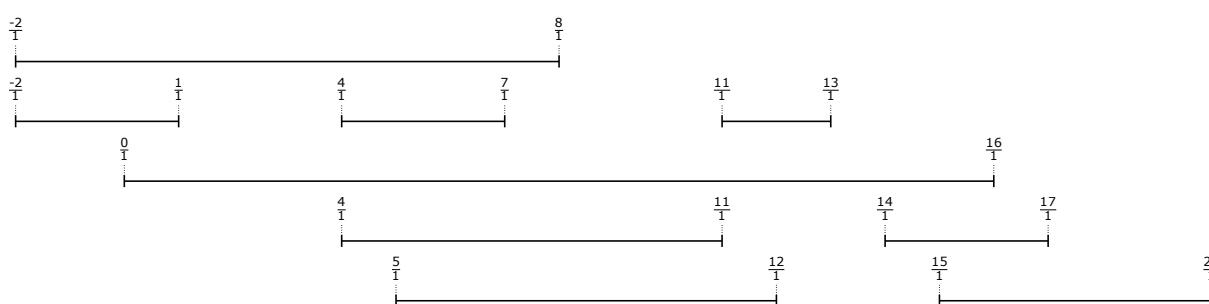
Conversely, *explosion* separates a timespan inventory into one or more new inventories in an attempt to limit the amount of overlap in each resulting inventory. The number of output inventories can be left unspecified, in which case explosion will generate as many inventories as necessary to prevent overlap entirely in every resulting inventory:<sup>6</sup>

```
>>> timespan_inventory = timespantools.TimespanInventory([
...     timespantools.Timespan(-2, 8),
...     timespantools.Timespan(-2, 1),
...     timespantools.Timespan(0, 16),
...     timespantools.Timespan(4, 7),
...     timespantools.Timespan(4, 11),
...     timespantools.Timespan(5, 12),
...     timespantools.Timespan(11, 13),
...     timespantools.Timespan(14, 17),
...     timespantools.Timespan(15, 20),
... ])
>>> show(timespan_inventory)
```



```
>>> for shard in timespan_inventory.explode():
...     show(shard, range_=(-2, 20))
...

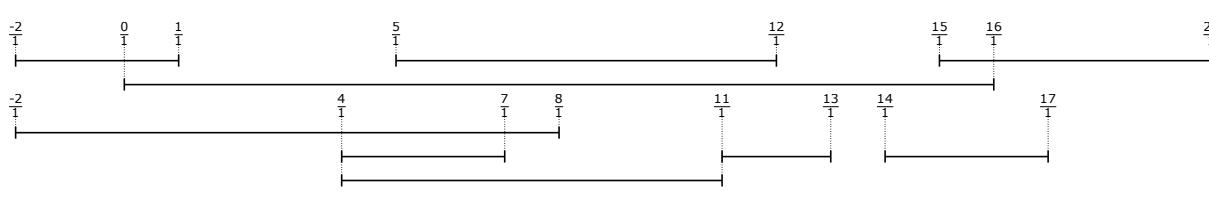
```



The number of output inventories can also be set explicitly, in which case explosion will attempt to limit the amount of overlap as much as possible, while maintaining a similar level of density across each resulting inventory:

```
>>> for shard in timespan_inventory.explode(inventory_count=2):
...     show(shard, range_=(-2, 20))
...

```



<sup>6</sup>Explosion is one of the techniques used to create timespan inventory illustrations, providing the mechanism by which arbitrary collections of timespans are automatically separated into non-overlapping inventories, which can then be rendered graphically as “rows” of line segments.

The procedures outlined above provide high-level tools for interacting with large numbers of timespans at once. All of the techniques described in chapter 4 with regards to Consort’s score interpretation stage – timespan consolidation, cascading overlap resolution, multiplexing multiple inventories into one, demultiplexing one inventory into many, and so forth – build on and extend these techniques.

### 3.3 ANNOTATED TIMESPANS IN CONSORT

While timespans and timespan inventories provide a very general model for modeling the disposition of durated events in time, a larger question remains: how can composers create enough timespans, and in various patterns, to be musically interesting? Consort approaches this problem by providing a collection of factory classes – *timespan-makers* – which can be configured and called to create arbitrarily large amounts of timespans. However, before turning to a detailed discussion of timespan-makers, we must first discuss the products of the timespan themselves.

Consort provides two separate timespan subclasses which are integral, if transient, components of its score interpretation stage: the `PerformedTimespan` and `SilentTimespan` classes. These classes are never created “by hand” during Consort’s specification stage – this is, explicitly instantiated by a composer while specifying a score segment –, but are instead generated as transitory objects during interpretation. Consort uses `PerformedTimespan` objects to indicate locations in the score timeline where some active musical material should appear, while `SilentTimespan` objects indicate tacet passages.

As will be described in more detail in chapter 4, Consort requires composers to specify musical materials in *layers*, and to specify specifically in which voice contexts in the score – vertically – that material should occur. During the course of interpretation, Consort organizes generated timespans by voice name and layer into separate timespan inventories, and then progressively masks out timespans in timespan inventories with lower layer numbers by those timespans with higher layer numbers. One can imagine this process as analogous to the use of opaque overlapping layers in image editing software. Both the `PerformedTimespan` and `SilentTimespan` classes provide configurable properties for layer and voice name, in addition to the start offset and stop offset properties provided by their parent `Timespan` class. These properties allow the processes that generate them to record *when* they were created, as well as *where* they should appear in the score, should they survive the masking process:

```
>>> performed_timespan = consort.PerformedTimespan(  
...     layer=1,  
...     start_offset=(1, 2),  
...     stop_offset=(7, 8),
```

```
| ...     voice_name='Clarinet Voice',
| ...     )
```

Performed timespans possess a number of other configurable properties used during score interpretation. These include `minimum_duration` – used to force erasure of timespans considered too short, often because they have been partially masked, `music_specifier` – used to attach information about how to generate notation from a timespan, `divisions` – a memento of the original duration structure of a contiguous sequence of performed timespans which have been consolidated into a single performed timespan after the masking process ends, and `music` – used to attach any generated notation prior to maquetting:

```
>>> performed_timespan = consort.PerformedTimespan(
...     layer=1,
...     minimum_duration=Duration(1, 8),
...     music_specifier=consort.MusicSpecifier(),
...     start_offset=Offset(1, 4),
...     stop_offset=Offset(2, 1),
...     voice_name='Violin 1 LH Voice',
...     )
```

Unlike performed timespans, silent timespans have no explicit reality in score, but can still be created for a particular voice context and layer. They act to simply erase any timespan in a lower layer through masking. This allows timespan-making processes to demand not only when a musical event should happen, but also when musical events should *not*. For example, silent timespans make it possible to demand silence before music events involving instrument or mallet changes:

```
>>> silent_timespan = consort.SilentTimespan(
...     layer=2,
...     start_offset=Offset(0, 1),
...     stop_offset=Offset(1, 4),
...     voice_name='Oboe Voice',
...     )
```

All of the above properties are explained in more depth in chapter 4, in context of Consort’s interpretation process. For the sake of demonstrating timespan-creation principles pithily, the `music_specifier` property of any performed timespan can be set to an arbitrary object such as Python’s `None` rather than a `MusicSpecifier` instance as actually done in practice.

## 3.4 TIMESPAN-MAKERS

Consort provides a family of factory classes for producing timespans, each implementing a different strategy for populating timespan inventories, but all unified via the same callable interface. Timespan-makers take as input a mapping of voice-names to *music specifiers* – arbitrary objects specifying how a given timespan might be rendered as notation –, a *target* timespan indicating the minimum and maximum permitted start offsets of any timespan created by the timespan-maker, an optional timespan inventory to modify in-place, and an optional *layer* identifier indicating in which pass a particular timespan was created.<sup>7</sup> All timespan-makers produce timespan inventories as output.

Timespan-makers fall into two broad categories: independent and dependent. Independent timespan-makers create timespans without regard for any pre-existing timespans, and therefore do not require being called with a timespan inventory instance. Dependent timespan-makers create their timespans based on the contents of a pre-existing timespan inventory, basing their output on various aspects of the structure of their input.

Additionally, all timespan-makers can be configured with padding and `timespan_specifier` keywords. Padding allows the timespan-maker to force “silence” – in the form of `SilentTimespan` objects – around the beginnings and ends of contiguous groups of timespans it creates. Timespan specifiers provide templates for some of the configurable properties specific to performed timespans, such as `minimum_duration` and `forbid_splitting`, allowing the timespan-maker to apply the configuration defined in the timespan specifier to each of the timespans it creates.

### 3.4.1 FLOODED TIMESPAN-MAKERS

*Flooded* timespan-makers implement the most trivial timespan-creation strategy in the timespan-maker class family. Their name derives from the ubiquitous graphic-design “flood fill” tool, which fills an entire connected area with the same color or texture. Flooded timespan-makers create one timespan for each voice in the input voice-name-to-music-specifier mapping, filling the entirety of the provided target timespan from beginning to end.

For example, the following flooded timespan-maker will create a timespan inventory populated by a single performed timespan associated with a “Violin Voice” context, completely filling the span of its target timespan, from  $\frac{1}{4}$  to  $1\frac{1}{8}$ :

---

<sup>7</sup>Layer identifiers allow timespans to be sorted not only by their position in time or by their associated voice name, but also by at which point – which *layer* – during some compositional process they were created. Layer ordering allows masking processes to mask “earlier” timespans with “later” ones.

```

>>> music_specifiers = {'Violin Voice': 'violin music'}
>>> target_timespan = timespantools.Timespan((1, 4), (11, 8))
>>> flooded_timespan_maker = consort.FloodedTimespanMaker()
>>> timespan_inventory = flooded_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
... )
>>> print(format(timespan_inventory))
timespantools.TimespanInventory(
    [
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 4),
            stop_offset=durationtools.Offset(11, 8),
            musicSpecifier='violin music',
            voiceName='Violin Voice',
        ),
    ]
)

```

Adding a second entry to the music specifier mapping results in two timespans in the output. Likewise, calling the timespan-maker with a layer keyword configures the output timespans with that layer number. The layer is indicated in the illustration just above and to the right of the beginning of each timespan:

```

>>> music_specifiers = {
...     'Violin Voice': 'violin music',
...     'Cello Voice': 'cello music',
... }
>>> timespan_inventory = flooded_timespan_maker(
...     layer=3,
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
... )
>>> print(format(timespan_inventory))
timespantools.TimespanInventory(
    [
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 4),
            stop_offset=durationtools.Offset(11, 8),
            layer=3,
            musicSpecifier='cello music',
            voiceName='Cello Voice',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 4),
            stop_offset=durationtools.Offset(11, 8),
            layer=3,
            musicSpecifier='violin music',
            voiceName='Violin Voice',
        ),
    ]
)

```

```
>>> show(timespan_inventory, key='voice_name')
```



Configuring the timespan-maker with padding creates silent timespans around the beginning and end of each group of output timespans on a per-voice basis. The timespan-maker will also configure these silent timespans with any specified layer number, allowing them to be sorted along with performed timespans from the same timespan-creation pass. Silent timespans are illustrated with dashed line to distinguish them from performed timespans:

```
>>> flooded_timespan_maker = consort.FloodedTimespanMaker(
...     padding=Duration(1, 4),
...     timespanSpecifier=consort.TimespanSpecifier(
...         minimumDuration=Duration(1, 8),
...         ),
...     )
>>> timespan_inventory = flooded_timespan_maker(
...     layer=5,
...     musicSpecifiers=music_specifiers,
...     targetTimespan=target_timespan,
...     )
>>> print(format(timespan_inventory))
timespantools.TimespanInventory(
    [
        consort.tools.SilentTimespan(
            startOffset=durationtools.Offset(0, 1),
            stopOffset=durationtools.Offset(1, 4),
            layer=5,
            voiceName='Violin Voice',
            ),
        consort.tools.SilentTimespan(
            startOffset=durationtools.Offset(0, 1),
            stopOffset=durationtools.Offset(1, 4),
            layer=5,
            voiceName='Cello Voice',
            ),
        consort.tools.PerformedTimespan(
            startOffset=durationtools.Offset(1, 4),
            stopOffset=durationtools.Offset(11, 8),
            layer=5,
            minimumDuration=durationtools.Duration(1, 8),
            musicSpecifier='cello music',
            voiceName='Cello Voice',
            ),
        consort.tools.PerformedTimespan(
            startOffset=durationtools.Offset(1, 4),
            stopOffset=durationtools.Offset(11, 8),
            layer=5,
```

```

        minimum_duration=durationtools.Duration(1, 8),
        musicSpecifier='violin music',
        voice_name='Violin Voice',
    ),
    consort.tools.SilentTimespan(
        start_offset=durationtools.Offset(11, 8),
        stop_offset=durationtools.Offset(13, 8),
        layer=5,
        voice_name='Violin Voice',
    ),
    consort.tools.SilentTimespan(
        start_offset=durationtools.Offset(11, 8),
        stop_offset=durationtools.Offset(13, 8),
        layer=5,
        voice_name='Cello Voice',
    ),
),
]
)

```

>>> show(timespan\_inventory, key='voice\_name')

Cello Voice:

Violin Voice:

### 3.4.2 TALEA TIMESPAN-MAKERS

Consort's `TaleaTimespanMaker` class creates rich timespan textures through the use of *talea* – infinitely cyclic duration patterns. Abjad implements the concept of talea via the `Talea` class in its `rhythmmakertools` library, which is discussed at length in section 3.5. `Talea` objects define their infinitely cyclic sequence of durations in terms of a finite sequence of numerators paired with a single denominator. Once defined, they can be iterated over and indexed like an infinite sequence, simplifying the process of generating looped patterns of durations of arbitrary lengths:

```

>>> talea = rhythmmakertools.Talea(
...     counts=[1, 2, 3, 4],
...     denominator=16,
... )
>>> for index in range(10):
...     talea[index]
...
NonreducedFraction(1, 16)
NonreducedFraction(2, 16)
NonreducedFraction(3, 16)
NonreducedFraction(4, 16)
NonreducedFraction(1, 16)
NonreducedFraction(2, 16)
NonreducedFraction(3, 16)

```

```
NonreducedFraction(4, 16)
NonreducedFraction(1, 16)
NonreducedFraction(2, 16)
```

```
>>> talea[99999]
NonreducedFraction(4, 16)
```

*Talea* timespan-makers make use of talea to control patterns of performed durations and silences. These timespan-makers can create timespans in *synchronized* or *unsynchronized* fashions. Unsynchronized timespan generation proceeds voice-by-voice through the input voice-name-to-music-specifier mapping, creating timespans from the start of the target timespan until its stop, then wrapping around to the next voice. Such unsynchronized timespan creation is ideal for creating massed textures of seemingly-unrelated timespans. Careful management of silence patterns and contiguous timespan group lengths during unsynchronized creation can create dense, overlapping textures or sparse, pointillistic ones. Contrastingly, synchronized creation provides a mechanism for creating moments of shared attack across voices, followed by shared silences:

```
>>> talea_timespan_maker = consort.TaleaTimespanMaker()
>>> print(format(talea_timespan_maker))
consort.tools.TaleaTimespanMaker(
    playing_talea=rhythmmakertools.Talea(
        counts=(4,),
        denominator=16,
    ),
    playing_groupings=(1,),
    repeat=True,
    silence_talea=rhythmmakertools.Talea(
        counts=(4,),
        denominator=16,
    ),
    step_anchor=Right,
    synchronize_groupings=False,
    synchronize_step=False,
)
```

All of the timespan inventories created in the following talea timespan-maker examples make use of the same music specifier mapping and target timespan. An ordered dictionary from Python’s `collections` module guarantees that the timespan-makers process the music specifier entries in the same order, from “Voice 1” through “Voice 4”:<sup>8</sup>

---

<sup>8</sup>The hash implementation used by Python’s dictionary class does not guarantee any particular ordering of keys, and may differ from one version of Python to the next, or even from one machine architecture to the next. While the timespan inventory illustrations shown with voice-name labels appear ordered, that is because of explicit lexical sorting of the voice-names in the

```

>>> import collections
>>> music_specifiers = collections.OrderedDict([
...     ('Voice 1', None),
...     ('Voice 2', None),
...     ('Voice 3', None),
...     ('Voice 4', None),
... ])
>>> target_timespan = timespan.tools.Timespan(0, (19, 4))

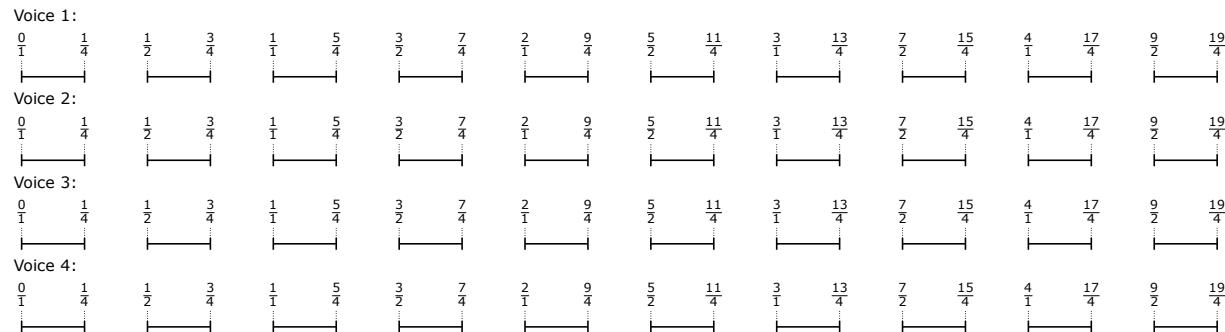
```

Without any manual configuration, talea timespan-makers generate unsynchronized 1-length groups of  $\frac{1}{4}$ -duration timespans, separated by  $\frac{1}{4}$ -duration silences, creating the appearance of synchronization across voices:

```

>>> result = talea_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
... )
>>> show(result, key='voice_name')

```



Changing the talea timespan-maker's `playing_talea` property from a series of  $\frac{1}{4}$ -durations to  $\frac{1}{4}, \frac{2}{4}, \frac{3}{4}, \frac{4}{4}$  reveals the timespan-maker's voice-wrapping behavior. Note how "Voice 1"'s timespans receive the durations  $\frac{1}{4}, \frac{2}{4}, \frac{3}{4}, \frac{4}{4}, \frac{1}{4}$  and  $\frac{2}{4}$ . "Voice 2" continues the duration sequence with  $\frac{3}{4}, \frac{4}{4}$  and so forth. The playing-duration talea continues to wrap around the end of each voice's timespans to the beginning of the next voice's:

```

>>> talea_timespan_maker = new(
...     talea_timespan_maker,
...     playing_talea=rhythmmakertools.Talea(
...         counts=(1, 2, 3, 4),
...         denominator=4,
...     )
... )
>>> result = talea_timespan_maker(
...     music_specifiers=music_specifiers,
...

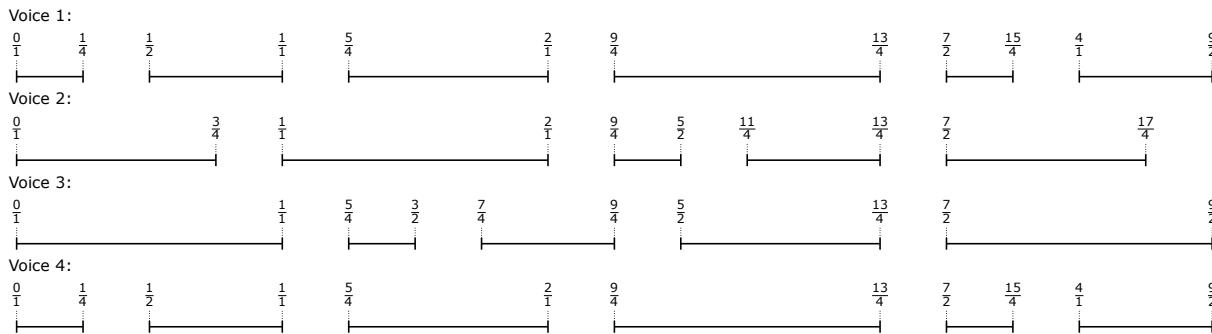
```

illustration algorithm, not because of any particular ordering in the timespan-maker's voice-name-to-music-specifier mapping. Timespan-makers iterate over the keys in that mapping when creating timespans, and plain dictionaries may lead to unexpected results. For that reason, these examples – and Consort itself – make use of the `OrderedDict` class from Python's `collections` module, which guarantees that keys be ordered by when they were inserted.

```

    ...      target_timespan=target_timespan,
...
)
>>> show(result, key='voice_name')

```

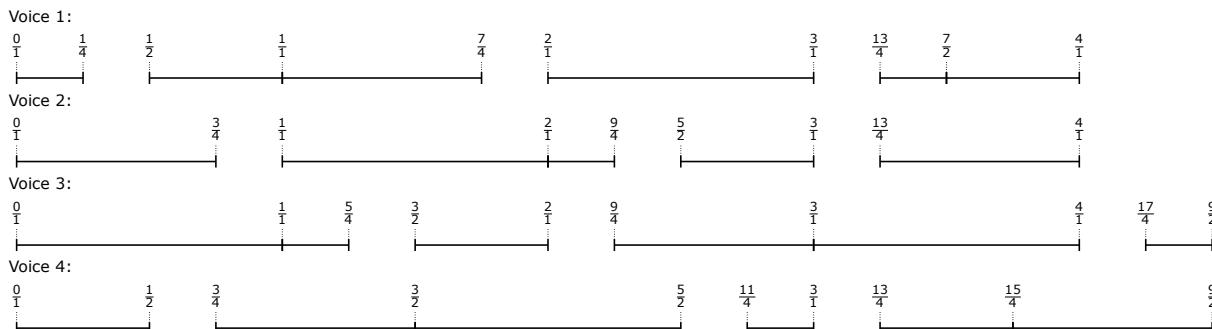


The talea timespan-maker's `playing_groupings` property controls how many timespans are created together as contiguous groups. When a grouping would run beyond the stop offset provided by the target timespan, that grouping's length is chosen again for the first timespan group of the next voice's timespans. Note how "Voice 2"'s final timespan, from  $1\frac{3}{4}$  to  $4\frac{1}{1}$ , should – according to the following talea timespan-maker's groupings pattern – begin a group of length 2. Because that grouping would run beyond the target timespan's  $1\frac{9}{4}$  stop offset, the following voice's first timespan group receives the 2-length grouping instead:

```

>>> talea_timespan_maker = new(
...     talea_timespan_maker,
...     playing_groupings=(1, 2),
... )
>>> result = talea_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
... )
>>> show(result, key='voice_name')

```



Reconfiguring the above talea timespan-maker with a different `silence_talea` produces patterned variations in the durations of silences between timespan groups:

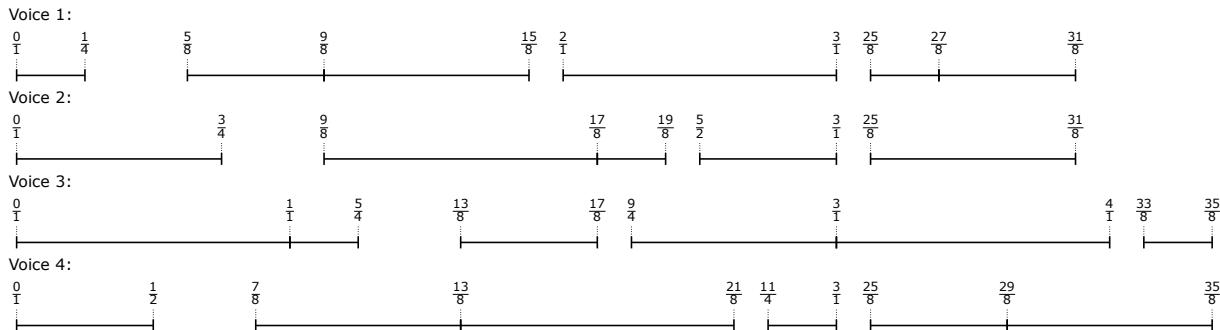
```

>>> talea_timespan_maker = new(
...     talea_timespan_maker,
...
```

```

...
    silence_talea=rhythmmakertools.Talea(
...
        counts=(3, 1, 1),
...
        denominator=8,
...
    ),
...
)
>>> result = talea_timespan_maker(
...
    music_specifiers=music_specifiers,
...
    target_timespan=target_timespan,
...
)
>>> show(result, key='voice_name')

```

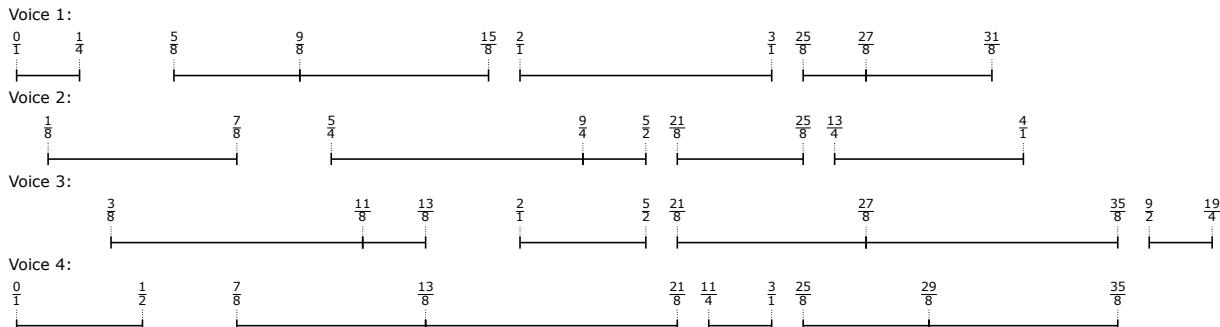


Likewise, changing the `initial_silence_talea` produces varying amounts of silence at the beginning of each voice:

```

>>> talea_timespan_maker = new(
...
    talea_timespan_maker,
...
    initial_silence_talea=rhythmmakertools.Talea(
...
        counts=(0, 1, 3),
...
        denominator=8,
...
    ),
...
)
>>> result = talea_timespan_maker(
...
    music_specifiers=music_specifiers,
...
    target_timespan=target_timespan,
...
)
>>> show(result, key='voice_name')

```

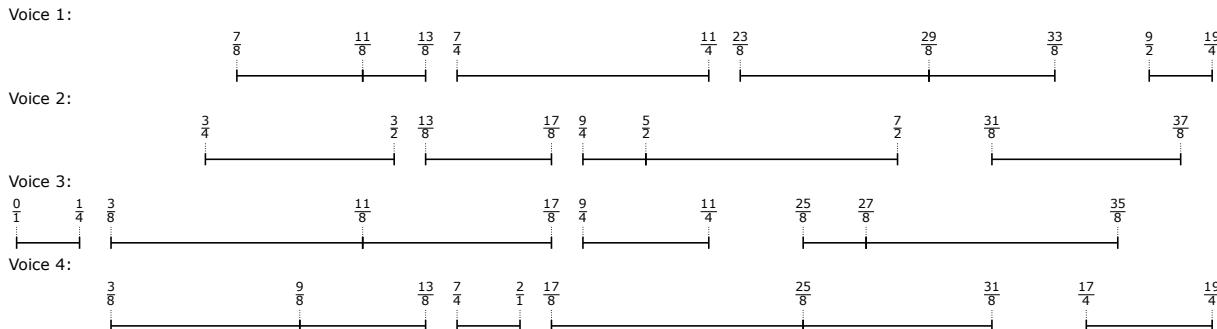


Talea timespan-makers also provide for transformations derived from the timespan inventory class itself. The `reflect` keyword configures the timespan-maker to reflect its output timespan inventory around its own axis, creating a “mirror image”:

```

>>> talea_timestspan_maker = new(
...     talea_timestspan_maker,
...     reflect=True,
... )
>>> result = talea_timestspan_maker(
...     music_specifiers=music_specifiers,
...     target_timestspan=target_timestspan,
... )
>>> show(result, key='voice_name')

```

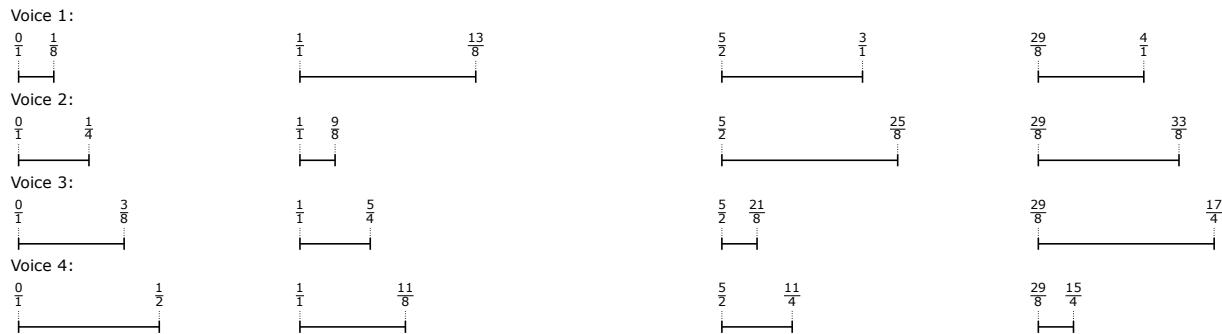


Synchronized talea timespan-makers can be created simply by configuring a new timespan-maker with its synchronize\_step flag set to true. This option causes the timespan-maker to create all of the timespans in every voice at once, then select some amount of silence between the end of that vertically-synchronized group and the beginning of the next. Note here how the duration of each timespan follows the timespan-maker's  $\frac{1}{4}$ ,  $\frac{2}{4}$ ,  $\frac{3}{4}$ ,  $\frac{4}{4}$ ,  $\frac{5}{4}$  pattern not from left-to-right by voice, but top-to-bottom by voice and then left-to-right:

```

>>> synchronized_talea_timestspan_maker = consort.TaleaTimespanMaker(
...     playing_talea=rhythmmakertools.Talea(
...         counts=(1, 2, 3, 4, 5),
...         denominator=8,
...     ),
...     silence_talea=rhythmmakertools.Talea(
...         counts=(4, 7),
...         denominator=8,
...     ),
...     synchronize_step=True,
... )
>>> result = synchronized_talea_timestspan_maker(
...     music_specifiers=music_specifiers,
...     target_timestspan=target_timestspan,
... )
>>> show(result, key='voice_name')

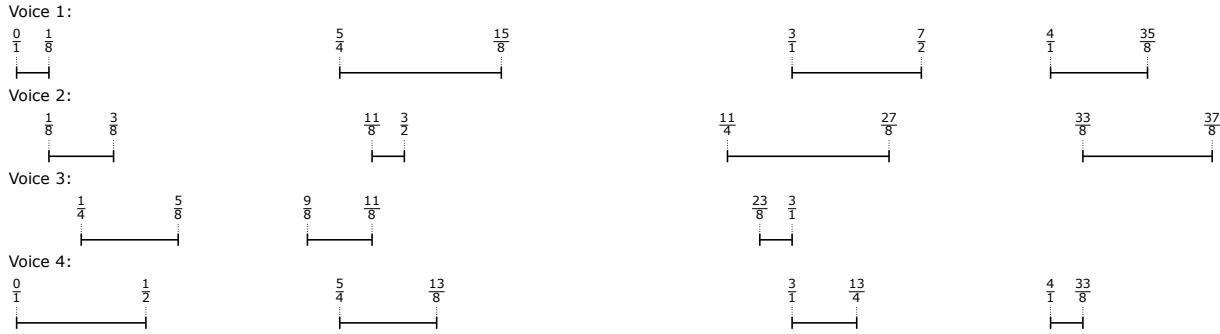
```



The silence durations between these synchronized groups are calculated from the end of the longest timespan in each group to the beginning of the next group. Therefore the second group starts at  $\frac{1}{1}$  because the first silence duration is  $\frac{4}{8}$  and the first group stops at  $\frac{1}{2}$ . Similarly, the third group starts at  $\frac{5}{2}$  ( $\frac{20}{8}$ ) because the second silence duration is  $\frac{7}{8}$  and the second group ended at  $\frac{13}{8}$ . This silence *stepping* can also be calculated from the beginning of one group to the next, rather than from the end of one to the beginning of the next, by changing the timespan-maker's `step_anchor` property from `Right` to `Left`. Such a change helps guarantee the timing of initial attacks across synchronized groups.

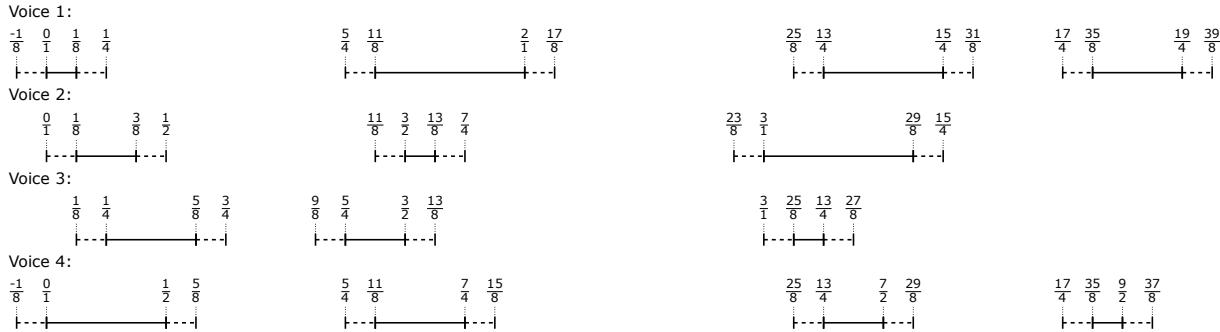
Initial silences behave differently in synchronized talea timespan-makers than with unsynchronized talea timespan-makers. Rather than adding silences only at the very beginning of each voice's timespans, silences are determined for each voice at the beginning of each synchronized group. Note how "Voice 3" is missing a fourth timespan. That timespan, with a duration of  $\frac{5}{8}$ , and an initial silence of  $\frac{2}{8}$  measured from the offset  $\frac{4}{1}$ , would have extended to  $\frac{39}{8} - \frac{1}{8}$  more than the target timespan's  $\frac{19}{4}$  stop offset:

```
>>> synchronized_talea_timespan_maker = new(
...     synchronized_talea_timespan_maker,
...     initial_silence_talea=rhythmmakertools.Talea(
...         counts=(0, 1, 2),
...         denominator=8,
...         ),
...     )
>>> result = synchronized_talea_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
...     )
>>> show(result, key='voice_name')
```



Unlike flooded timespan-makers, padding durations are included in the start offset and step duration calculations for talea timespan-makers:

```
>>> synchronized_talea_timespan_maker = new(
...     synchronized_talea_timespan_maker,
...     padding=(1, 8),
... )
>>> result = synchronized_talea_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
... )
>>> show(result, key='voice_name')
```



### 3.4.3 DEPENDENT TIMESPAN-MAKERS

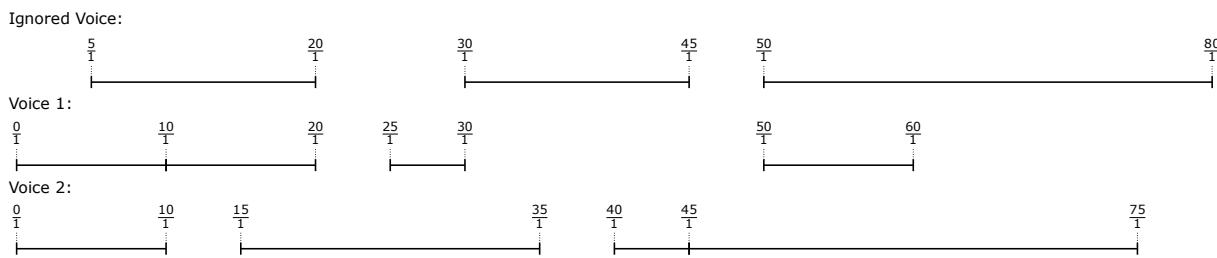
*Dependent* timespan-makers create timespans based on the disposition of timespans in an input timespan inventory, selecting the start and stop offsets of their generated timespans by filtering and inspecting timespans from their input with respect to their voice name or music specifier label. This behavior helps model how timespans used to generate phrasing for a pianist's pedaling voice can be derived by selecting only those timespans in a timespan inventory for entire ensemble which pertain to the pianist's right and left-hand voices, ignoring all others.

Consider the following timespan inventory, comprised of both overlapping and non-overlapping timespans created for the contexts "Voice 1", "Voice 2" and "Ignored Voice":

```

>>> timespan_inventory = timespantools.TimespanInventory([
...     consort.PerformedTimespan(0, 10, voice_name='Voice 1'),
...     consort.PerformedTimespan(0, 10, voice_name='Voice 2'),
...     consort.PerformedTimespan(5, 20, voice_name='Ignored Voice'),
...     consort.PerformedTimespan(10, 20, voice_name='Voice 1'),
...     consort.PerformedTimespan(15, 35, voice_name='Voice 2'),
...     consort.PerformedTimespan(25, 30, voice_name='Voice 1'),
...     consort.PerformedTimespan(30, 45, voice_name='Ignored Voice'),
...     consort.PerformedTimespan(40, 45, voice_name='Voice 2'),
...     consort.PerformedTimespan(45, 75, voice_name='Voice 2'),
...     consort.PerformedTimespan(50, 80, voice_name='Ignored Voice'),
...     consort.PerformedTimespan(50, 60, voice_name='Voice 1'),
... ])
...
>>> show(timespan_inventory, key='voice_name')

```

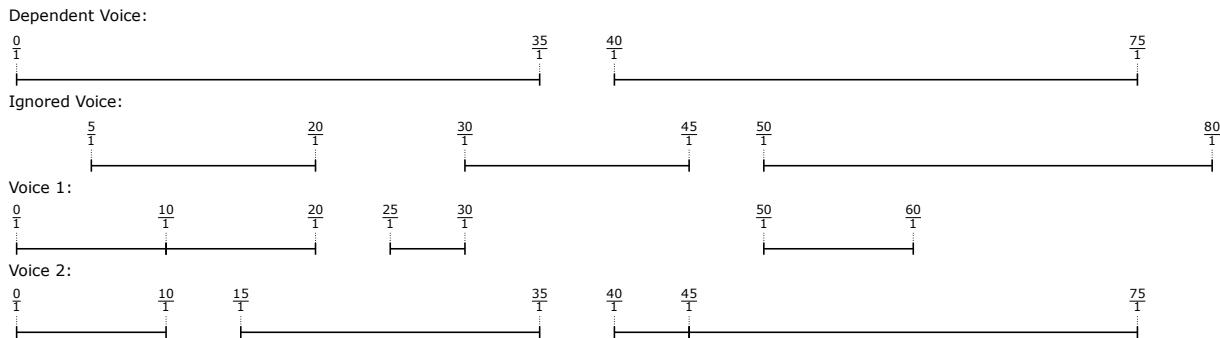


A dependent timespan-maker can be configured to depend on timespans created for the “Voice 1” and “Voice 2” contexts by specifying a tuple of voice names during instantiation. Passing the previously defined timespan inventory as an argument when calling the dependent timespan-maker adds the newly-created dependent timespans to it. For the sake of brevity, and because timespan-makers modify any timespan inventory passed as an argument to `__call__()` in-place, we pass a copy of this timespan inventory, created via a call to `new()`, instead of the original:

```

>>> music_specifiers = {'Dependent Voice': None}
>>> dependent_timespan_maker = consort.DependentTimespanMaker(
...     voice_names=(
...         'Voice 1',
...         'Voice 2',
...     )
... )
>>> result = dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     timespan_inventory=new(timespan_inventory),
... )
...
>>> show(result, key='voice_name')

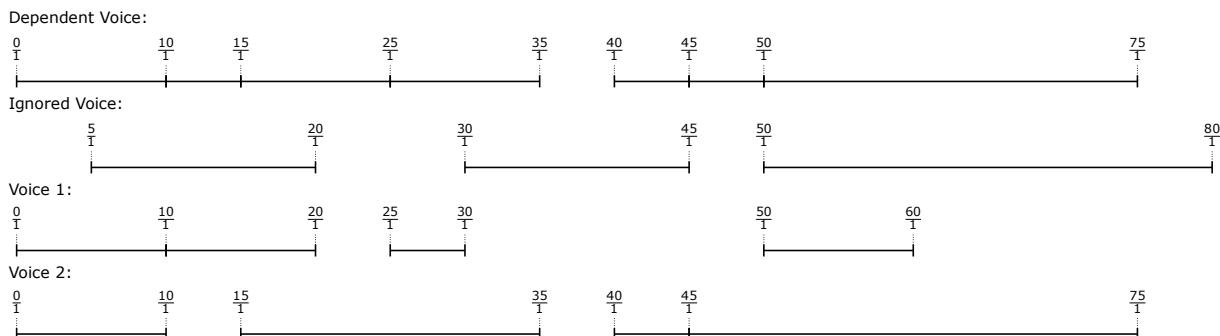
```



Note how the above timespan inventory shows the timespans for “Dependent Voice” outlining the start and stop offset for partitioned shards of the timespans for “Voice 1” and “Voice 2”, but ignoring the boundaries outlined by the timespans for the “Ignored Voice” context. Dependent timespans first select all timespans from their input timespan inventory matching their voice-names criteria, then partition them into shards in order to determine which offsets to use for timespan creation. Partitioning guarantees that the created dependent timespans do not exceed the bounds of the timespans they depend upon.

Configuring the dependent timespan-maker with its `include_inner_starts` flag set to true causes it to create contiguous groups of dependent timespans, as though splitting at every moment when a timespan it depends upon starts. Note that new dependent timespans begin at the offsets  $10\frac{1}{1}$ ,  $15\frac{1}{1}$  and  $25\frac{1}{1}$ , for example. This is because a new timespan begins in “Voice 1” at both  $10\frac{1}{1}$  and  $25\frac{1}{1}$ , and another new timespan begins in “Voice 2” at  $15\frac{1}{1}$ :

```
>>> new_dependent_timespan_maker = new(
...     dependent_timespan_maker,
...     include_inner_starts=True,
... )
>>> result = new_dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     timespan_inventory=new(timespan_inventory),
... )
>>> show(result, key='voice_name')
```

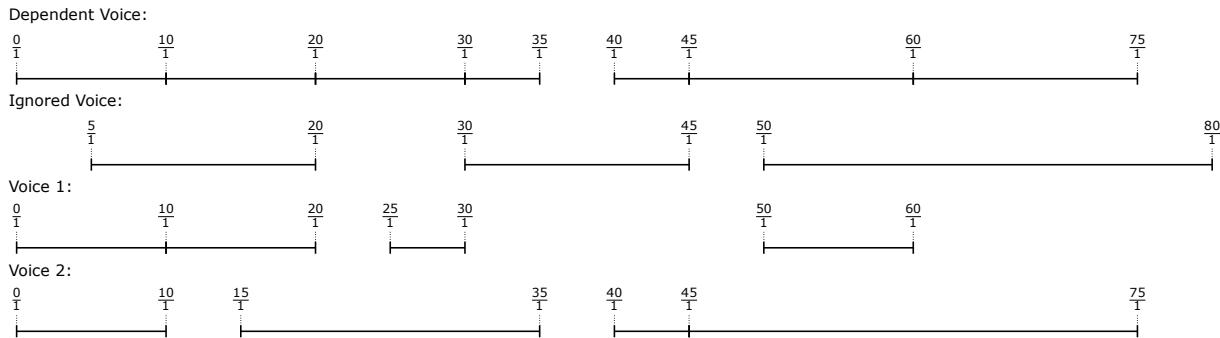


Likewise, the `include_inner_stops` flag causes the dependent timespan-maker to take into account the stop offsets of any timespan it depends upon:

```

>>> new_dependent_timespan_maker = new(
...     dependent_timespan_maker,
...     include_inner_stops=True,
... )
>>> result = new_dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     timespan_inventory=new(timespan_inventory),
... )
>>> show(result, key='voice_name')

```

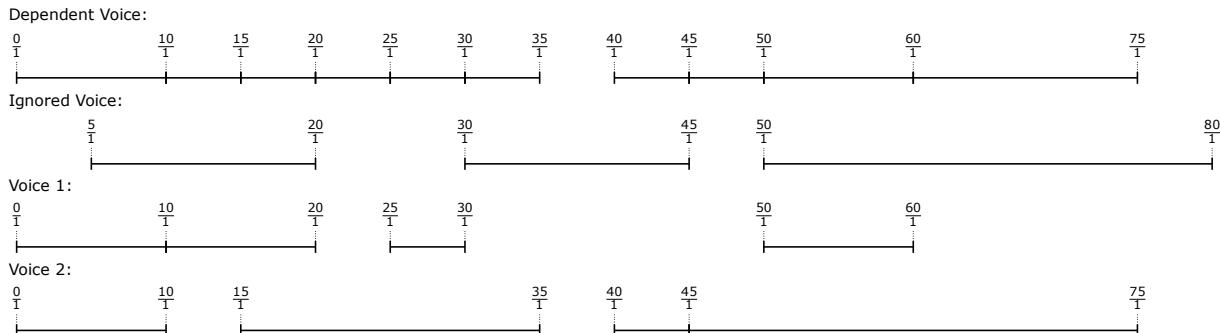


Both options can be combined together, resulting in dependent timespans outlining all offsets from the collection of timespans they depend upon:

```

>>> new_dependent_timespan_maker = new(
...     dependent_timespan_maker,
...     include_inner_starts=True,
...     include_inner_stops=True,
... )
>>> result = new_dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     timespan_inventory=new(timespan_inventory),
... )
>>> show(result, key='voice_name')

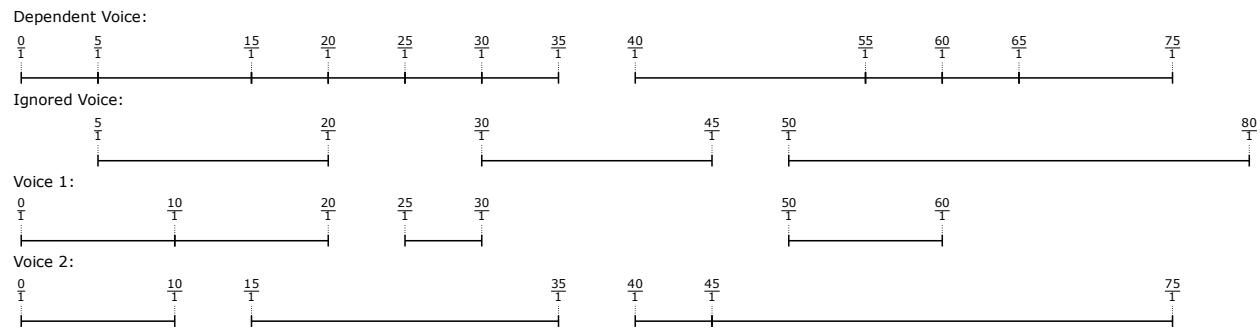
```



Dependent timespan-makers are capable of performing simple transformations on the offsets they extract from their input timespan inventory. Rotation allows the timespan-maker to rotate the durations outlined by the offsets extracted from the timespans they select. Specifying a rotation index of 1 causes each created group of dependent

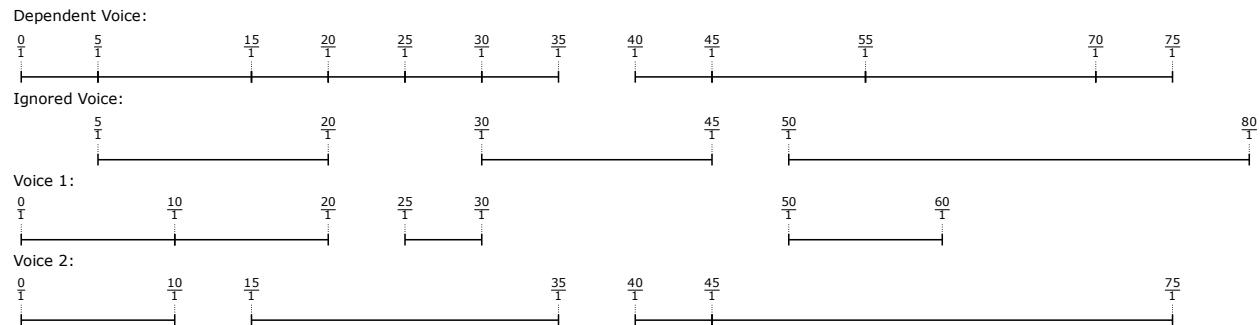
timespans to rotate its internal durations by that index. The 10-duration dependent timespan which previously occurred first in the “Dependent Voice” now occurs second, following a  $\frac{5}{1}$ -duration timespan:

```
>>> rotated_dependent_timespan_maker = new(
...     new_dependent_timespan_maker,
...     rotation_indices=(1,),
... )
>>> result = rotated_dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     timespan_inventory=new(timespan_inventory),
... )
>>> show(result, key='voice_name', range_=(0, 75))
```



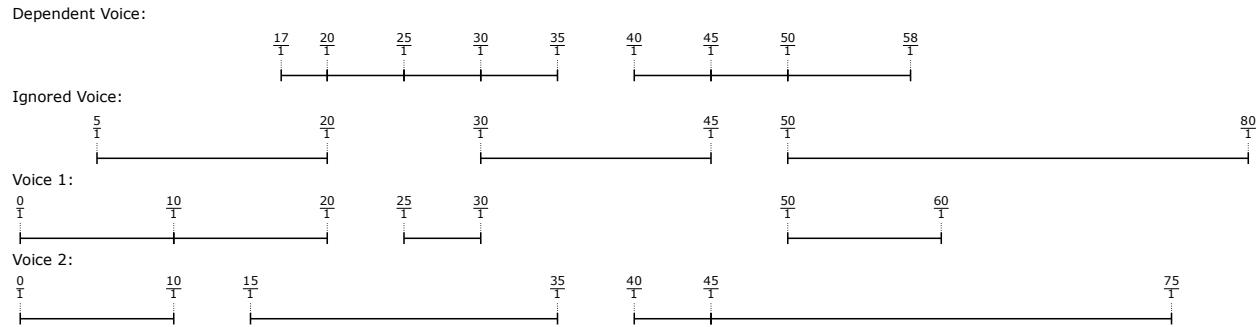
More than one rotation index can be specified, allowing for each group of dependent timespans – as created from each shard of the partitioned selected timespans – to be rotated independently:

```
>>> rotated_dependent_timespan_maker = new(
...     new_dependent_timespan_maker,
...     rotation_indices=(1, -1),
... )
>>> result = rotated_dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     timespan_inventory=new(timespan_inventory),
... )
>>> show(result, key='voice_name', range_=(0, 75))
```



Note that in the previous dependent timespan-maker examples no target timespan was specified. When passed a non-empty timespan inventory during calling, timespan-makers can treat that inventory's timespan as their target timespan if no target timespan was specified explicitly. However, passing a target timespan to a dependent timespan-maker causes that timespan-maker to perform a logical AND of the target timespan with any selected timespans in the input inventory. Here the dependent timespans are constrained between the offsets  $17\frac{1}{1}$  and  $58\frac{1}{1}$ :

```
>>> result = new_dependent_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=timespantools.Timespan(17, 58),
...     timespan_inventory=new(timespan_inventory),
... )
>>> show(result, key='voice_name', range_=(0, 75))
```



## 3.5 RHYTHM-MAKERS

Abjad's rhythm-makers, like Consort's timespan-makers, are highly-configurable factory classes which behave like partially evaluated functions, taking as input sequences of *divisions* – positive, non-reduced fraction tokens<sup>9</sup> representing the divisions in some phrase of music – and producing selections of score components as output. Abjad's *rhythmmakertools* library contains a variety of such classes, each providing a different strategy for rhythm generation, but unified by the same callable interface. Additionally, *rhythmmakertools* provides a collection of *specifier* classes which group related configuration values together for controlling the behavior of ties, beams, duration spelling and other notational aspects of each rhythm-maker's output. Like many other classes in Abjad – timespans, for example – both these specifiers and the rhythm-makers themselves can be templated via calls to `new()`. A tour of these rhythm-maker classes demonstrates how a wide range of rhythmic textures can be produced.

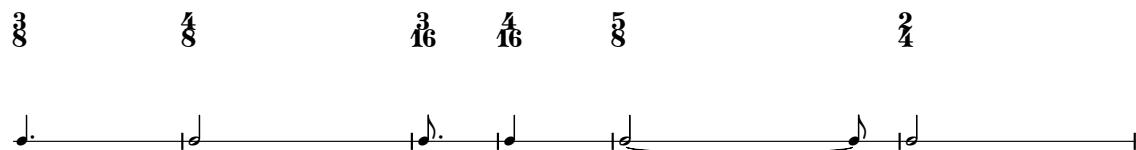
---

<sup>9</sup>Rather than coercing input into sequences of `Duration` objects, which reduce their denominators as much as possible, rhythm-makers treat all input as *non-reduced fractions*, allowing them to disambiguate  $\frac{4}{16}$  from  $\frac{2}{8}$  or  $\frac{6}{8}$  from  $\frac{3}{4}$  and to therefore treat those division tokens as distinct.

### 3.5.1 NOTE RHYTHM-MAKERS

*Note* rhythm-makers, arguably the simplest class of rhythm-maker, take a sequence of input divisions and “fill” them with notes, tied as necessary, such that the duration of each logical tie in each output division equals the duration of each input division:

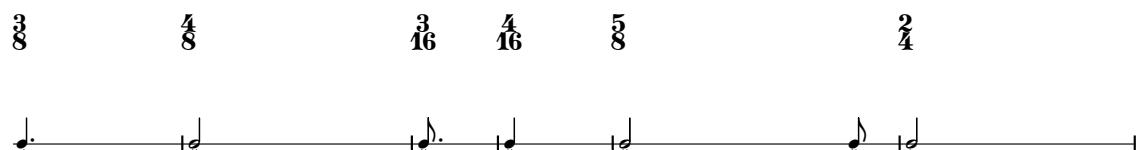
```
>>> note_rhythm_maker = rhythmmakertools.NoteRhythmMaker()
>>> divisions = [(3, 8), (4, 8), (3, 16), (4, 16), (5, 8), (2, 4)]
>>> show(note_rhythm_maker, divisions=divisions)
```



Like many objects implemented in Abjad and its extensions, rhythm-makers can be illustrated via a call to `show()`. Rhythm-maker illustrations take an optional `divisions` argument, specifying what durations should be used for the generated rhythmic output, grouping each of those divisions into measures for ease of visualization.

Rhythm-makers can be configured with a variety of specifiers, allowing for optional customization of their rhythm-generating behavior. For example, a `TieSpecifier` can be used to force a rhythm-maker to tie the last note of each output division to the first note of the next output division. When used with a note rhythm-maker, this effectively ties all notes in the output together:

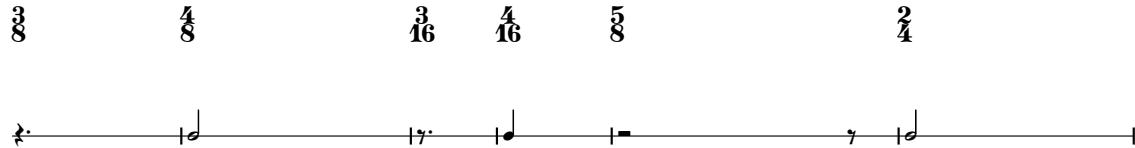
```
>>> note_rhythm_maker = new(
...     note_rhythm_maker,
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tie_across_divisions=True,
...         ),
...     )
>>> show(note_rhythm_maker, divisions=divisions)
```



Other rhythm-maker specifiers influence beaming, tuplet spelling, or can cause a rhythm-maker to convert patterned groups of leaves in its output from notes to rests or vice versa. Rhythm-makers configured with `output_masks` replace the contents of their output divisions with rests in a patterned per-division basis. A sequence of one or more `BooleanPattern` instances control the masking pattern. These patterns partition the rhythm-maker’s output divisions into segments of a given period, and then mask out divisions specified by indices within that period.

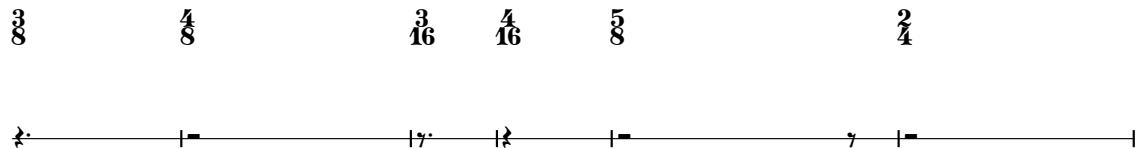
For example, a note rhythm-maker configured with a single output mask of a period of length 2 and a masking index 0 will cause the first of every two divisions to be silenced:

```
>>> mask = rhythmmakertools.BooleanPattern(indices=[0], period=2)
>>> note_rhythm_maker = rhythmmakertools.NoteRhythmMaker(output_masks=[mask])
>>> show(note_rhythm_maker, divisions=divisions)
```



Reducing the period of the boolean pattern from 2 to 1 silences every output division, effectively turning a note-generating rhythm-maker into a rest-generating rhythm-maker:

```
>>> mask = rhythmmakertools.BooleanPattern(indices=[0], period=1)
>>> note_rhythm_maker = rhythmmakertools.NoteRhythmMaker(output_masks=[mask])
>>> show(note_rhythm_maker, divisions=divisions)
```



### 3.5.2 TALEA RHYTHM-MAKERS

*Talea* rhythm-makers, like talea timespan-makers, create rhythmic output through the use of a talea – an infinitely cyclic pattern of durations. Talea rhythm-makers fill their output divisions with durations from their talea, splitting those durations across division boundaries when the divisions are over-full. The following trivial talea rhythm-maker uses a length-1 talea comprised of a single  $1/16$  duration:

```
>>> talea_rhythm_maker = rhythmmakertools.TaleaRhythmMaker(
...     talea=rhythmmakertools.Talea(
...         counts=[1],
...         denominator=16,
...         ),
...     )
>>> show(talea_rhythm_maker, divisions=divisions)
```



Extending the talea counts to a sequence of alternating  $\frac{1}{16}$  and  $\frac{1}{8}$  notes produces more complex results. Note how the  $\frac{1}{8}$  durations break over the boundaries of the  $\frac{3}{16}$  measure, but remain tied together. Talea rhythm-makers handle talea splitting and tying transparently:

```
>>> talea_rhythm_maker = new(
...     talea_rhythm_maker,
...     talea_counts=[1, 2],
... )
>>> show(talea_rhythm_maker, divisions=divisions)
```

With the talea counts changed to a descending sequence of durations, the pattern of split and tied durations also changes:

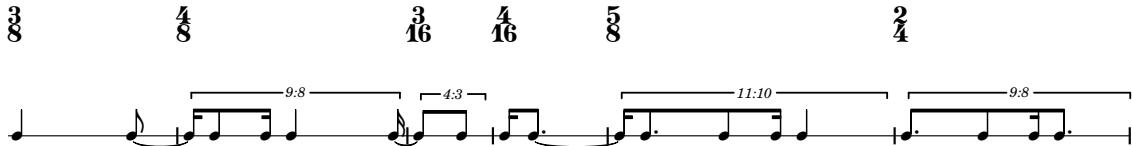
```
>>> talea_rhythm_maker = new(
...     talea_rhythm_maker,
...     talea_counts=[4, 3, 2, 1],
... )
>>> show(talea_rhythm_maker, divisions=divisions)
```

Talea rhythm-makers can be configured to treat input divisions as though they had more counts than they actually do via their `extra_counts_per_division` property, causing tuplets to appear in the output. The following talea, reconfigured from the previous, adds an extra count to every second and third input division. This causes the  $\frac{4}{8}$  and  $\frac{3}{16}$  as well as the  $\frac{5}{8}$  and  $\frac{2}{4}$  divisions to become tuplets, each with a pre-prolated contents duration  $\frac{1}{16}$  longer than their prolated duration. In other words, the  $\frac{4}{8}$  division contains a triplet whose contents sum to  $\frac{1}{16}$ , but scaled into a duration of  $\frac{8}{16}$ . Likewise, the  $\frac{3}{16}$  division contains a triplet whose contents sum to  $\frac{4}{16}$ , but scaled into a duration of  $\frac{3}{16}$ . Note how this tupletting causes the pattern of split and tied talea durations to shift. In the previous example, the second instance of the talea's  $\frac{3}{16}$  duration occurred entirely during the  $\frac{3}{16}$  division. Here, that same  $\frac{3}{16}$  talea duration begins on the final  $\frac{1}{16}$  of the  $\frac{4}{8}$  division's 9:8 triplet, tied into the 4:3 triplet in the following division:

```

>>> talea_rhythm_maker = new(
...     talea_rhythm_maker,
...     extra_counts_per_division=[0, 1, 1],
... )
>>> show(talea_rhythm_maker, divisions=divisions)

```

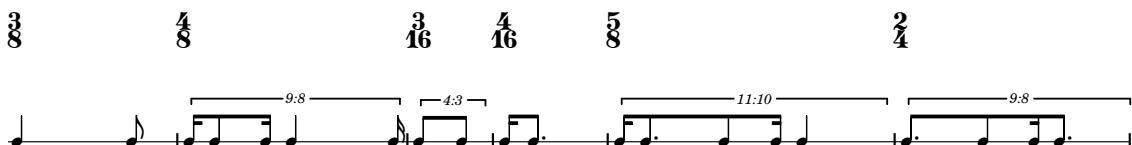


Like note rhythm-makers, talea rhythm-makers can be configured with a tie specifier, causing the last note of each division to be tied to the first note of the next division. In note rhythm-makers, the contents of each non-masked division is guaranteed to be a single logical tie. Therefore, tying across divisions results in the entire output of the note rhythm-maker merging into the same logical tie. The output divisions of talea rhythm-makers generally contain more than one logical tie, and therefore tying across divisions tends to produce the effect of elided downbeats with intermittent attacks:

```

>>> talea_rhythm_maker = new(
...     talea_rhythm_maker,
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tie_across_divisions=True,
...     ),
... )
>>> show(talea_rhythm_maker, divisions=divisions)

```



Talea rhythm-makers can also be configured to produce intermittent silences, either by specifying negative count values in the rhythm-maker's talea, or by configuring the rhythm-maker with a special *burnish* specifier which casts logical ties generated by the rhythm-maker as either notes or rests, in a patterned way. Here, the 3-count in the talea rhythm-maker's talea is changed to -3, resulting in the production of 3/16-duration silences:

```

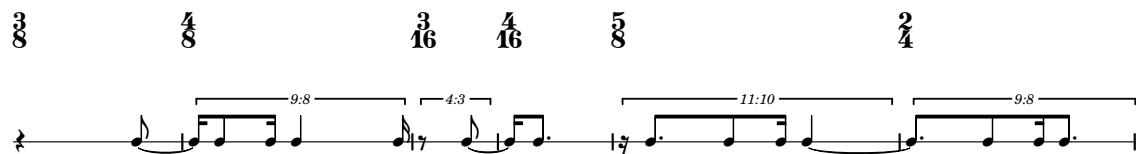
>>> talea_rhythm_maker_with_rests = new(
...     talea_rhythm_maker,
...     talea_counts=[4, -3, 2, 1],
... )
>>> show(talea_rhythm_maker_with_rests, divisions=divisions)

```



Alternatively, configuring the rhythm-maker with the following `BurnishSpecifier` allows the first logical tie of every other division to be converted to rests:

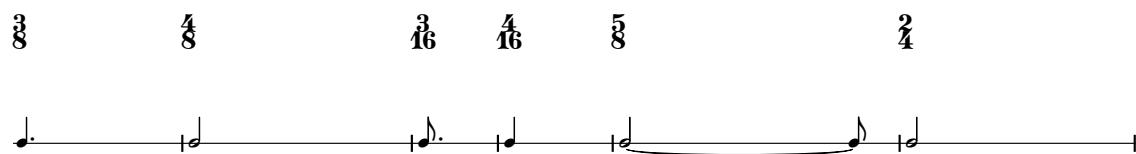
```
>>> talea_rhythm_maker_with_rests = new(
...     talea_rhythm_maker,
...     burnish_specifier=rhythmmakertools.BurnishSpecifier(
...         left_classes=[Rest],
...         left_counts=[1, 0],
...         ),
...     )
>>> show(talea_rhythm_maker_with_rests, divisions=divisions)
```



### 3.5.3 INCISED RHYTHM-MAKERS

*Incised* rhythm-makers behave similarly to note rhythm-makers, but allow for *incising* patterned sequences of notes and rests from the beginnings and ends of each output division, or even from the beginning and end of the entire sequence of divisions – the rhythm-maker *output*. Configuring an incised rhythm-maker to perform incision requires an `InciseSpecifier` instance. An unconfigured incised rhythm-maker behaves identically to an unconfigured note rhythm-maker:

```
>>> incised_rhythm_maker = rhythmmakertools.IncisedRhythmMaker()
>>> show(incised_rhythm_maker, divisions=divisions)
```



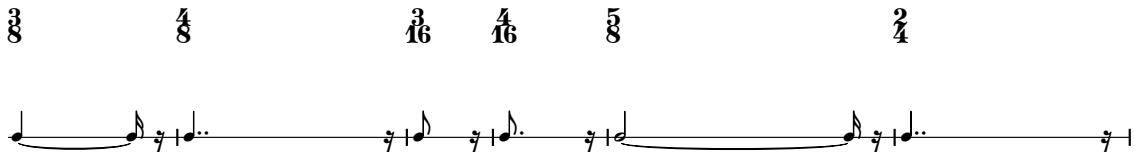
Incise specifiers define talea and group counts for prefix and suffix incision. The talea – as determined by the numerators given in the `prefix_talea` and `suffix_talea` sequences combined with the `talea_denominator` – define the durations to be selected from, as well as the *sign* of the component to be incised. Negative talea items indicate rests, while positive talea items indicate notes. The count properties – `suffix_counts` and `prefix_counts` – indicate how many talea durations should be selected at each pass. The following incised rhythm-maker incises a single  $\frac{1}{16}$  rest at the end of each division:

```
>>> incised_rhythm_maker = rhythmmakertools.IncisedRhythmMaker(
...     inciseSpecifier=rhythmmakertools.InciseSpecifier(
```

```

...
    suffix_counts=[1],
...
    suffix_talea=[-1],
...
    talea_denominator=16,
),
)
>>> show(incised_rhythm_maker, divisions=divisions)

```



Extending the suffix talea results in a more complex incision pattern:

```

>>> incised_rhythm_maker = new(
...     incised_rhythm_maker,
...     inciseSpecifier__suffix_talea=[-1, -2, -3],
... )
>>> show(incised_rhythm_maker, divisions=divisions)

```

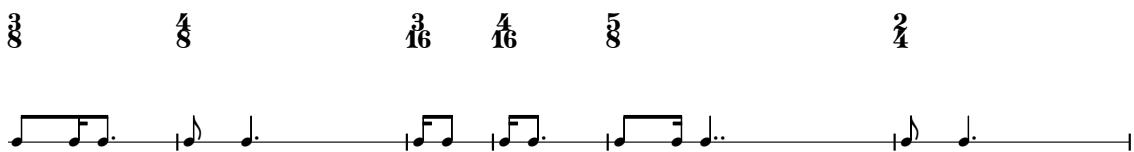


Prefix incision occurs in an identical fashion, by specifying sequences of group counts and talea numerators. Here, alternating groups of notes of length-1 and length-2 are incised from the beginning of each division:

```

>>> incised_rhythm_maker = rhythmtools.IncisedRhythmMaker(
...     inciseSpecifier=rhythmtools.InciseSpecifier(
...         prefix_counts=[2, 1],
...         prefix_talea=[2, 1],
...         talea_denominator=16,
...     ),
... )
>>> show(incised_rhythm_maker, divisions=divisions)

```



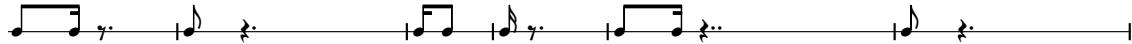
Setting the incise specifier's `fill_with_notes` property to `False` causes the rhythm-maker to fill the unincised portions of its output divisions with rests rather than notes. Filled mainly with silence, the incision pattern becomes much clearer:

```

>>> incised_rhythm_maker = new(
...     incised_rhythm_maker,
...     inciseSpecifier__fill_with_notes=False,
... )
>>> show(incised_rhythm_maker, divisions=divisions)

```

$\frac{3}{8}$        $\frac{4}{8}$        $\frac{3}{16}$      $\frac{4}{16}$      $\frac{5}{8}$        $\frac{2}{4}$



Incised rhythm-makers can be configured to only incise the outer divisions of their output. The following rhythm-maker cuts  $\frac{1}{8}$  rests from the beginning of the first division in its output, and the end of the last division:

```
>>> incised_rhythm_maker = rhythmmakertools.IncisedRhythmMaker(
...     inciseSpecifier=rhythmmakertools.InciseSpecifier(
...         outerDivisionsOnly=True,
...         prefixCounts=[1],
...         prefixTalea=[-1],
...         suffixCounts=[1],
...         suffixTalea=[-1],
...         taleaDenominator=8,
...         ),
...     )
>>> show(incised_rhythm_maker, divisions=divisions)
```

$\frac{3}{8}$        $\frac{4}{8}$        $\frac{3}{16}$      $\frac{4}{16}$      $\frac{5}{8}$        $\frac{2}{4}$



Incised rhythm-makers also respect many of the same rhythm-maker specifiers as the other rhythm-maker classes defined in Abjad's `rhythmmakertools` library. Like note rhythm-makers, they can be configured to tie the last note of each output division to the first note of the next output division via an instance of the `TieSpecifier` class:

```
>>> incised_rhythm_maker = new(
...     incised_rhythm_maker,
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tieAcrossDivisions=True,
...         ),
...     )
>>> show(incised_rhythm_maker, divisions=divisions)
```

$\frac{3}{8}$        $\frac{4}{8}$        $\frac{3}{16}$      $\frac{4}{16}$      $\frac{5}{8}$        $\frac{2}{4}$



### 3.5.4 EVEN-DIVISION RHYTHM-MAKERS

*Even-division* rhythm-makers attempt to divide each input division into runs of notes with a basic duration  $1/\text{denominator}$ , where the denominator is specified on a per-division basis by the rhythm-maker's configurable denominator

sequence property. When an input division does not exactly fit some multiple of this basic duration, some combination of augmentation or tupletting will be used to produce notes with durations as close to that basic duration as possible:

```
>>> even_division_rhythm_maker = rhythmmakertools.EvenDivisionRhythmMaker()
>>> show(even_division_rhythm_maker, divisions=divisions)
```

$\frac{3}{8}$        $\frac{4}{8}$        $\frac{3}{16}$      $\frac{4}{16}$      $\frac{5}{8}$        $\frac{2}{4}$



Extending the even-division rhythm-maker's denominator sequence to a 2-length pattern produces divisions filled alternatingly with  $\frac{1}{8}$  and  $\frac{1}{16}$  notes. Note that the  $\frac{3}{16}$  measure is filled with a dotted  $\frac{1}{8}$  note:

```
>>> even_division_rhythm_maker = new(
...     even_division_rhythm_maker,
...     denominators=[8, 16],
... )
>>> show(even_division_rhythm_maker, divisions=divisions)
```

$\frac{3}{8}$        $\frac{4}{8}$        $\frac{3}{16}$      $\frac{4}{16}$      $\frac{5}{8}$        $\frac{2}{4}$



Extending the denominator sequence even further results in more complex output. Note here how the  $\frac{5}{8}$  division is rendered as a tuplet:

```
>>> even_division_rhythm_maker = new(
...     even_division_rhythm_maker,
...     denominators=[8, 4, 16],
... )
>>> show(even_division_rhythm_maker, divisions=divisions)
```

$\frac{3}{8}$        $\frac{4}{8}$        $\frac{3}{16}$      $\frac{4}{16}$      $\frac{5}{8}$        $\frac{2}{4}$

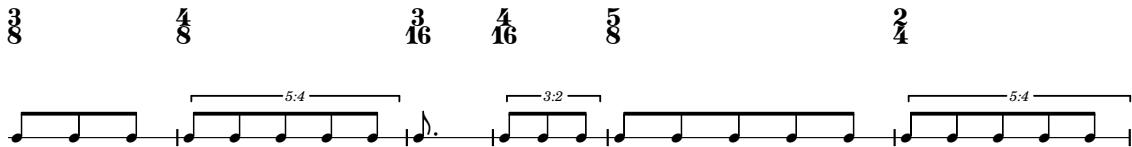


Like the talea rhythm-maker, even-division rhythm-makers can be configured to add extra counts to each output division, forcing the rhythm-maker to treat input divisions as longer than they actually are. Artificially lengthened divisions are rendered as tuplets. With the following rhythm-maker which attempts to fill each output division with a run of  $\frac{1}{8}$  notes, every other division is extended by one count, causing the  $\frac{4}{8}$  and  $\frac{2}{4}$  divisions to be rendered as 5:4 tuplets, and the  $\frac{4}{16}$  division to be rendered as a 3:2 triplet:

```

>>> even_division_rhythm_maker = new(
...     even_division_rhythm_maker,
...     denominators=[8],
...     extra_counts_per_division=(0, 1),
... )
>>> show(even_division_rhythm_maker, divisions=divisions)

```

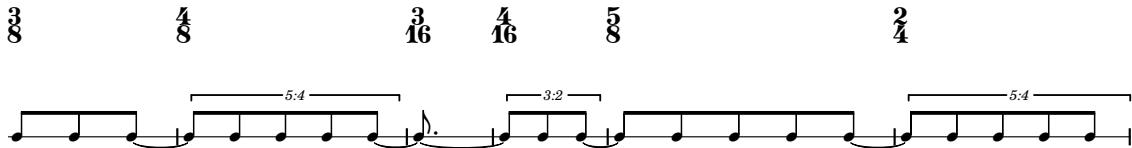


When configured with a tie specifier, the even-division rhythm-maker obscures the downbeat of every output division:

```

>>> even_division_rhythm_maker = new(
...     even_division_rhythm_maker,
...     tieSpecifier=rhythmmakertools.TieSpecifier(
...         tie_across_divisions=True,
...     ),
... )
>>> show(even_division_rhythm_maker, divisions=divisions)

```

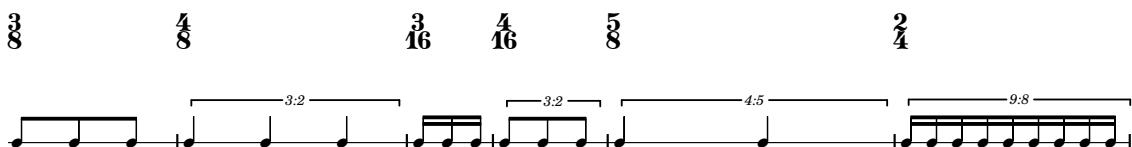


And when reconfigured with the earlier 8-4-16 denominator pattern, the even-division rhythm-maker produces rich tupletted rhythmic output:

```

>>> even_division_rhythm_maker = new(
...     even_division_rhythm_maker,
...     denominators=[8, 4, 16],
... )
>>> show(even_division_rhythm_maker, divisions=divisions)

```

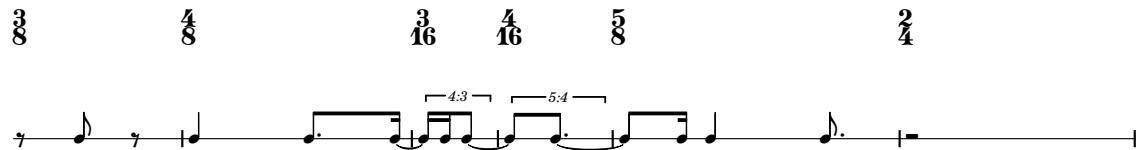


### 3.5.5 COMPOSITE RHYTHM-MAKERS

Consort provides a class for aggregating multiple rhythm-maker instances together into a *composite* rhythm-maker, which applies its aggregated rhythm-makers conditionally against input divisions to generate rhythmic output. The following contrived composite rhythm-maker uses the previously note rhythm-maker – which only generates rests

- for the last of any sequence of input divisions, the previously defined incised rhythm-maker for the first of any sequence of input divisions, and the previously defined talea rhythm-maker for all other input divisions:

```
>>> composite_rhythm_maker = consort.CompositeRhythmMaker(
...     default=talea_rhythm_maker,
...     last=note_rhythm_maker,
...     first=incised_rhythm_maker,
... )
>>> show(composite_rhythm_maker, divisions=divisions)
```



### 3.6 METER

**TODO:** Provide a definition of meter. Cite Lerdahl.

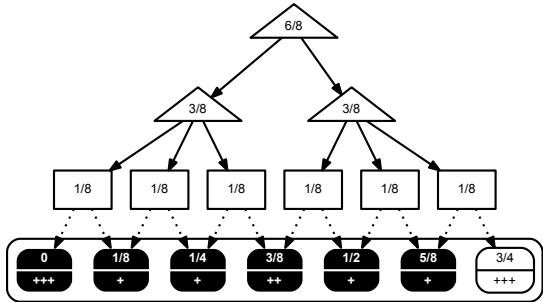
Abjad models meter as a *rhythm-tree* of nested, durated nodes which outline a series of strongly and weakly accented offsets. The accent strength of a particular offset found in a meter's rhythm-tree derives from the number of nodes in that tree sharing that offset as a start or stop. The more nodes in the rhythm-tree which share an offset, the greater the weight – the accentedness – of that offset is taken to be. Abjad can construct the rhythm tree for any meter from a numerator / denominator pair such as a rational duration or time signature. Meter construction involves the progressive division of the numerator of the input pair into groups of two and threes<sup>10</sup>, and the decomposition of any other prime factors into groups of threes and twos. Division by two always occurs before division by three, giving preference to even metrical structures above odd or otherwise prime divisions. Constructing rhythm-trees in this fashion gives results which generally align with common practice expectations.

Consider the following  $\frac{6}{8}$  meter and its graph representation:

```
>>> six_eight_meter = metertools.Meter((6, 8))
>>> graph(six_eight_meter)
```

---

<sup>10</sup>The factors 4 and 5 are also used in meter rhythm-tree generation as they provide better typical results during meter rewriting.

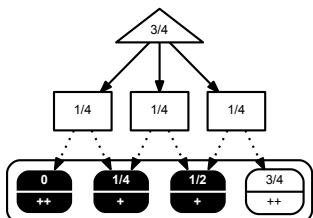


The triangular and rectangular boxes indicate nodes in the rhythm-tree itself. Rectangular boxes represent “beats” – the leaves of the rhythm-tree – while triangular boxes indicate larger metrical groupings. The ovals at the bottom of the graph indicate – at their top – the start or stop offset of the nodes connected to them from above and – at their bottom – the relative weight of their accent. The final oval on the right indicates the offset and accent weight of the “next” downbeat.

The topmost triangle in the above graph represent the “highest” metrical grouping in a  $\frac{6}{8}$  meter. Tracing the leftmost and rightmost arrows down through the topmost node’s children gives the offsets 0 and  $\frac{3}{4}$ : the first down-beat and next downbeat in a  $\frac{6}{8}$  meter. Offsets 0 and  $\frac{3}{4}$  also have the strongest accent weights as they occur as either the start offset or stop offset of nodes at three levels of hierarchy in the rhythm tree. At the second level the  $\frac{6}{8}$  grouping divides into two  $\frac{3}{8}$  groupings, following common practice expectations: metrical groupings tend to subdivide into groups of two before they subdivide into groups of three.<sup>11</sup> Both second-level nodes share the offset of  $\frac{3}{8}$ , which also occurs in the third level, giving  $\frac{3}{8}$  a weight of two. The third level contains the  $\frac{1}{8}$  duration beats, grouped by their parents in the second level into two groups three  $\frac{1}{8}$  duration nodes. The offsets  $\frac{1}{8}$ ,  $\frac{1}{4}$ ,  $\frac{1}{2}$  and  $\frac{5}{8}$  are not shared by any nodes except at the lowest metrical level and therefore all receive an accent weight of one.

Consider the following other examples of meters modeled in Abjad. A  $\frac{3}{4}$  meter consists of a top-level  $\frac{3}{4}$  metrical grouping divided into three  $\frac{1}{4}$  duration beats:

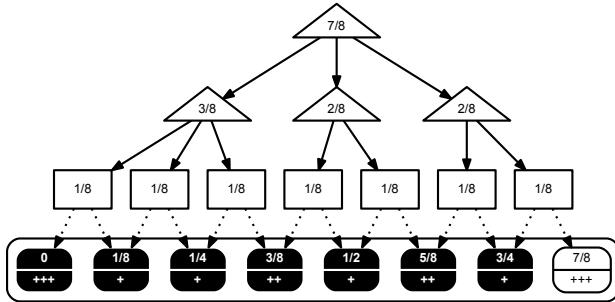
```
>>> three_four_meter = metertools.Meter((3, 4))
>>> graph(three_four_meter)
```



<sup>11</sup>Consider a  $\frac{12}{8}$  meter. Western musicians tend to subdivide twelve into either two groups of six or four groups of three rather than into three groups of four.

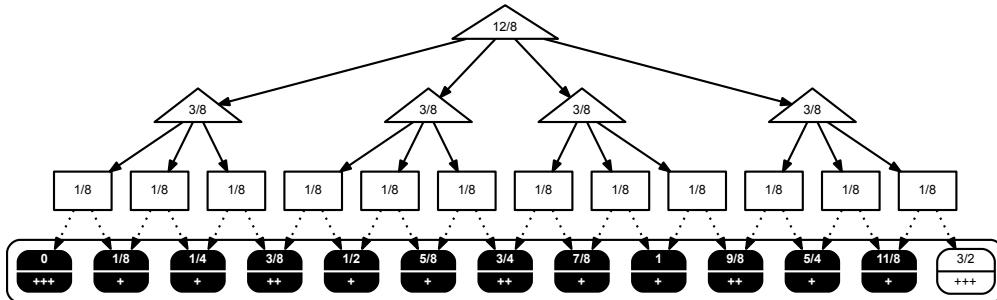
By default, a  $\frac{7}{8}$  meter subdivides its top-level metrical grouping into  $\frac{3}{8} + \frac{2}{8} + \frac{2}{8}$  groupings:

```
>>> seven_eight_meter = metertools.Meter((7, 8))
>>> graph(seven_eight_meter)
```



A  $\frac{12}{8}$  meter subdivides into four  $\frac{3}{8}$  duration groupings, each containing three  $\frac{1}{8}$  duration beats:

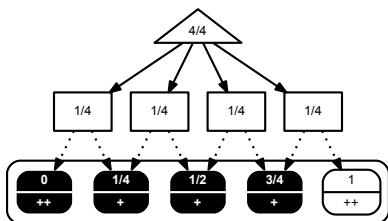
```
>>> twelve_eight_meter = metertools.Meter((12, 8))
>>> graph(twelve_eight_meter)
```



Abjad also permits alternate representations of meters which ostensibly share the same numerator and denominator.

The default interpretation of  $\frac{4}{4}$  generates a top-level rhythmic grouping with a duration of  $\frac{4}{4}$  and four  $\frac{1}{4}$  beats as children:<sup>12</sup>

```
>>> four_four_meter = metertools.Meter((4, 4))
>>> graph(four_four_meter)
```



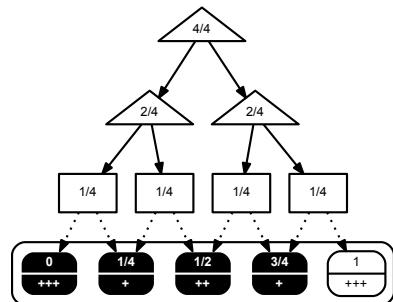
<sup>12</sup>A “flat”  $\frac{4}{4}$  metrical structure is useful for meter rewriting as it allows the meter rewriting algorithm to ignore many common rhythmic idioms like  $\frac{1}{4} + \frac{1}{2} + \frac{1}{4}$  and  $\frac{1}{4} + \frac{3}{4}$ .

While meter objects are usually instantiated from numerator / denominator pairs, with their rhythm-tree structure determined programmatically from that input pair, they can also be instantiated from strings parsable as rhythm-trees, or from rhythm-tree objects themselves. All meters, because they are implemented in terms of rhythm-trees, can be represented by a Lisp-like rhythm-tree syntax:

```
>>> print(four_four_meter.pretty_rtm_format)
(4/4 (
  ^^I1/4
  ^^I1/4
  ^^I1/4
  ^^I1/4))
```

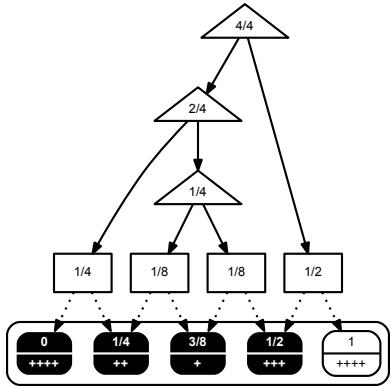
Instantiating meters from explicit rhythm tree syntax allows composers to choose alternate representations of metrical structures. For example, a  $\frac{4}{4}$  meter which strongly emphasizes beat three is possible by subdividing the top-level  $\frac{4}{4}$  metrical grouping into two  $\frac{2}{4}$  duration groupings, which are then subdivided each into two  $\frac{1}{4}$  duration beats. Such a metrical structure effectively treats  $\frac{4}{4}$  as identical to  $\frac{2}{2}$ :

```
>>> arbitrary_meter_1 = metertools.Meter('4/4 ((2/4 (1/4 1/4)) (2/4 (1/4 1/4))))')
>>> graph(arbitrary_meter_1)
```



Unusual metrical structures are also possible, such as the following  $\frac{4}{4}$  meter which divides into two parts, with the first part dividing into two again, and the second grouping of that divided into two again:

```
>>> arbitrary_meter_2 = metertools.Meter('4/4 ((2/4 (1/4 (1/4 (1/8 1/8)))) 1/2))')
>>> graph(arbitrary_meter_2)
```



### 3.7 REWRITING METERS

Notated rhythms can be expressed in multiple ways while maintaining the same attack-point and duration structure.

*Meter rewriting* formalizes the process of re-notating a rhythm according to the offset structure inherent to some meter while maintaining the original attack-points and durations by fusing and splitting logical ties according to their *validity*. In the context of meter rewriting, validity expresses whether any logical tie – trivial or not – *aligns* to offsets found in nodes at a particular depth in a meter’s rhythm-tree. Alignment requires that a given logical tie either starts or stops at offsets found in the collection of offsets defined by a subtree of nodes in a meter’s rhythm-tree. Meter rewriting proceeds by testing logical ties against offsets outlined first by the root node of a meter and, if found invalid, against those offsets found in nodes progressively deeper in the meter’s rhythm-tree.

For example, the offsets outlined by the root node of a  $\frac{6}{8}$  meter can be found by examining its depth-wise offset inventory:

```

>>> six_eight_meter = metertools.Meter((6, 8))
>>> six_eight_meter.depthwise_offset_inventory[0]
(Offset(0, 1), Offset(3, 4))
  
```

A logical tie three-quarters in duration starting at the offset 0 would be considered valid in the context of  $\frac{6}{8}$  because it aligns to the offsets outlined at depth-0 of the meter’s rhythm-tree. In contrast, a logical tie two quarters in duration starting at  $\frac{1}{8}$  – therefore outlining the timespan of  $\frac{1}{8}:\frac{5}{8}$  – would be considered invalid due to misalignment:

```

>>> six_eight_measure = Measure((6, 8), "r8 c'2 r8")
>>> show(six_eight_measure)
  
```



Meter rewriting splits misaligned logical ties at any possible offset found in the currently considered depth. If no offsets at that depth intersect with the misaligned tie, the depth is increased and the process repeats. At a depth of 1 in a  $\frac{6}{8}$  meter we finally find an offset intersecting the timespan of the misaligned  $\frac{1}{8}:\frac{5}{8}$  logical tie at the offset  $\frac{3}{8}$ :

```
>>> six_eight_meter.depthwise_offset_inventory[1]
(Offset(0, 1), Offset(3, 8), Offset(3, 4))
```

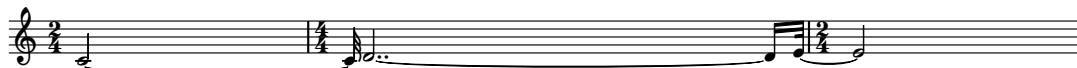
Rewriting the contents of the  $\frac{6}{8}$  measure against a  $\frac{6}{8}$  meter splits the inner half-note at the  $\frac{3}{8}$  offset boundary:

```
>>> mutate(six_eight_measure).rewrite_meter(six_eight_meter)
>>> show(six_eight_measure)
```



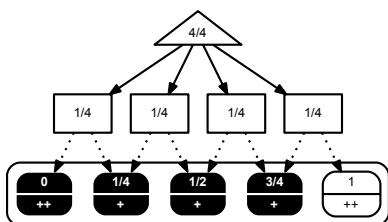
As another example, consider the following rhythm:

```
>>> parseable = "abj: | 2/4 c'2 ~ |"
>>> parseable += "| 4/4 c'32 d'2.. ~ d'16 e'32 ~ |"
>>> parseable += "| 2/4 e'2 |"
>>> staff = Staff(parseable)
>>> show(staff)
```



The middle measure is notated in a perfectly valid manner. However, the double-dotted D does not align with or break against any of the offsets of a  $\frac{4}{4}$  metrical structure:  $\frac{0}{4}, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}$  or  $\frac{4}{4}$ :

```
>>> four_four_meter = metertools.Meter((4, 4))
>>> graph(four_four_meter)
```



Rewriting the inner measure against a  $\frac{4}{4}$  meter breaks the inner logical tie at  $\frac{1}{4}$  and  $\frac{3}{4}$ , slightly improving readability:

```
>>> mutate(staff[1]).rewrite_meter(four_four_meter)
>>> show(staff)
```



Rewriting the same measure against a  $\frac{2}{2}$  meter breaks the inner logical tie at the offset  $\frac{1}{2}$ , as two double-dotted quarter notes. While identical to  $\frac{4}{4}$  in duration,  $\frac{2}{2}$  strongly emphasizes this duple division:

```
>>> two_two_meter = metertools.Meter((2, 2))
>>> staff = Staff(parseable)
>>> mutate(staff[1]).rewrite_meter(two_two_meter)
>>> show(staff)
```



### 3.7.1 DOT COUNT

Meter rewriting can control for various qualities of how rhythms are notated. For example, the maximum number of dots allowed for any notated rhythmic value can be constrained. Logical ties encountered during the rewriting process whose individual notes exceed the maximum number of permitted dots will be rewritten, if a maximum dot count has been specified.

Consider this series of progressively rewritten rhythms, beginning with the following unrewritten  $\frac{3}{4}$  measure:

```
>>> measure = Measure((3, 4), "c'32 d'8 e'8 fs'4...")
>>> show(measure)
```



After meter rewriting, the final F-sharp is still notated as a triple-dotted quarter-note, valid because its stop offset aligns perfectly with the containing measure's stop offset:

```
>>> mutate(measure).rewrite_meter((3, 4))
>>> show(measure)
```



Capping the maximum number of dots to 2 causes the F-sharp to be rewritten as a double-dotted eighth-note tied to a quarter-note:

```
>>> measure = Measure((3, 4), "c'32 d'8 e'8 fs'4...")
>>> mutate(measure).rewrite_meter((3, 4), maximum_dot_count=2)
>>> show(measure)
```



Constraining the maximum number of dots to 1 further subdivides the F-sharp logical tie:

```
>>> measure = Measure((3, 4), "c'32 d'8 e'8 fs'4...")
>>> mutate(measure).rewrite_meter((3, 4), maximum_dot_count=1)
>>> show(measure)
```



Finally, with no dots permitted at all, the rhythmic presentation of the measure changes considerably. Every dotted rhythm has been subdivided:

```
>>> measure = Measure((3, 4), "c'32 d'8 e'8 fs'4...")
>>> mutate(measure).rewrite_meter((3, 4), maximum_dot_count=0)
>>> show(measure)
```

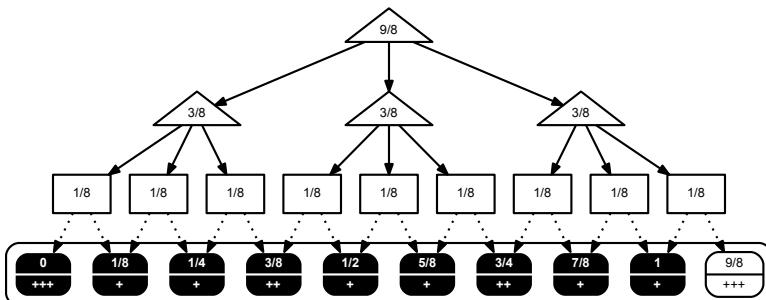


Prudent application of dot count constraints can clarify awkward rhythmic spellings by forcing the appearance of offsets in logical ties inherent to the rewriting meter.

### 3.7.2 BOUNDARY DEPTH

In meter rewriting, *boundary depth* forces emphasis of offsets found at different *depths* in a meter's rhythm-tree by marking as invalid logical ties which do not immediately align to those offsets on the very first pass of the recursive meter rewriting process. Use of boundary depth often clarifies distinctive groupings in more deeply nested meters, such as the “3-ness” inherent to a  $\frac{9}{8}$  meter. The nodes with a depth of 1 in the following  $\frac{9}{8}$  meter's rhythm-tree – the three  $\frac{3}{8}$  inner nodes marked by triangles – outline the offsets  $\frac{0}{8}$ ,  $\frac{3}{8}$ ,  $\frac{6}{8}$  and  $\frac{9}{8}$ . When rewriting a  $\frac{9}{8}$  rhythm with a boundary depth of 1, any logical ties not aligning with – either starting or stopping at – those offsets would be marked as invalid and therefore rewritten:

```
>>> nine_eight_meter = metertools.Meter((9, 8))
>>> graph(nine_eight_meter)
```



```
>>> measure = Measure((9, 8), "c'2 d'2 e'8")
>>> show(measure)
```



```
>>> mutate(measure).rewrite_meter(nine_eight_meter)
>>> show(measure)
```



After rewriting, without any boundary depth specified, the D half-note in the above  $\frac{9}{8}$  measure has been split into two quarter-notes, tied together. The second of these quarter-notes begins at  $\frac{6}{8}$ , therefore aligning with the start of the third  $\frac{3}{8}$  node at depth 1 in the  $\frac{9}{8}$  meter's rhythm tree. Likewise, the first half of the split half-note ends at the same  $\frac{6}{8}$  offset. However, the initial C half-note, while aligning with the beginning of the meter and therefore treated as valid, does not emphasize any of the meter's inner offsets as outlined by the offsets  $\frac{3}{8}$  and  $\frac{6}{8}$ :

```
>>> measure = Measure((9, 8), "c'2 d'2 e'8")
>>> mutate(measure).rewrite_meter(
...     nine_eight_meter,
...     boundary_depth=1,
... )
>>> show(measure)
```



After rewriting with boundary depth set to 1, not only has the D half-note been split in half, but the initial C half-note has been split at its  $\frac{3}{8}$  offset. Because the initial half note aligned at the  $\frac{0}{8}$  offset – as outlined by the root node in the  $\frac{9}{8}$  meter's rhythm tree –, but not at the  $\frac{3}{8}$  offset – as outlined by the first and second  $\frac{3}{8}$ -duration nodes at depth 1 of the same meter's rhythm tree –, it was marked invalid and therefore split at the first available offset:  $\frac{3}{8}$ .

### 3.7.3 RECURSIVE METER REWRITING

Meter rewriting treats the contents of tuplets with non-trivial prolation as existing within their own metrical scope, isolated from any other meter. The numerator and denominator of the triplet's pre-prolated contents duration act as the numerator and denominator of their “virtual” meter. Thus, a 6:5 tuplet encountered in any context will be rewritten as though under some 6-numerator meter:

```
>>> parseable = "abj: | 4/4 c'16 ~ c'4 d'8. ~ "
>>> parseable += "2/3 { d'8. ~ 3/5 { d'16 e'8 ~ e'16 f'16 ~ } }
```

```
>>> parseable += "f'4 |"
>>> measure = parse(parseable)
>>> show(measure)
```

A musical staff in treble clef with a key signature of one sharp. It contains two measures. The first measure has a duration of f'4. The second measure starts with a bracket labeled '3:2' above and '5:3' below, indicating a complex meter change. The notes in the second measure are eighth and sixteenth notes.

```
>>> mutate(measure).rewrite_meter(
...     measure,
...     boundary_depth=1,
... )
>>> show(measure)
```

The same musical staff as above, but with different note heads. The first measure still has a duration of f'4. The second measure now has a bracket labeled '3:2' above and '5:3' below. The notes are represented by different shapes, such as vertical stems and horizontal dashes, to reflect the specific meter rewriting.

If, while rewriting the contents of one tuplet, a second tuplet is encountered as a child of that first tuplet, the meter rewriting algorithm will recursively descend into that second tuplet – and any further tuplet children at any depth. This recursive descent allows any encountered component to be rewritten in a relevant metrical context.

### 3.7.4 EXAMPLES

Meter rewriting can clarify structural differences between meters with identical durations, such as  $\frac{3}{4}$  and  $\frac{6}{8}$ , or between various possible representations of other prime-numerator meters like  $\frac{5}{4}$ ,  $\frac{7}{8}$  and so forth. The following series of  $\frac{3}{4}$  measures contain rhythms which emphasize  $\frac{3}{4}$ -ness or  $\frac{6}{8}$ -ness to different degrees. While perfectly valid and totally legible, they can still be rewritten to more strongly express one meter over the other:

```
>>> staff = Staff(context_name='RhythmicStaff')
>>> staff.extend("{ c'2 c'4 } { c'4. c'4. } { c'2 ~ c'8 c'8 }")
>>> attach(TimeSignature((3, 4)), staff)
>>> show(staff)
```

A musical staff in treble clef with a key signature of one sharp. It shows a measure with a time signature of (3, 4). The rhythm consists of a dotted half note, a quarter note, an eighth note, a dotted half note, an eighth note, and a sixteenth note. A bracket indicates a group of three quarter-duration nodes:  $\frac{1}{4}$ ,  $\frac{1}{4}$ ,  $\frac{2}{4}$  and  $\frac{3}{4}$ .

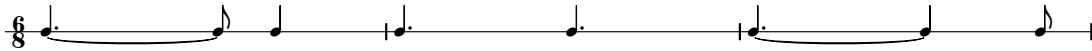
Rewriting under a  $\frac{3}{4}$  meter with a boundary depth of 1 forces emphasis of the offsets found in the  $\frac{3}{4}$  meter's depth-1 group of three quarter-duration nodes:  $\frac{1}{4}$ ,  $\frac{1}{4}$ ,  $\frac{2}{4}$  and  $\frac{3}{4}$ . Both the outer measures' contents align to these offsets, but the inner measure's hemiola contents do not, and are rewritten:

```
>>> for container in staff:
...     mutate(container).rewrite_meter((3, 4), boundary_depth=1)
...
>>> show(staff)
```



Conversely, rewriting with a  $\frac{6}{8}$  meter – again with a boundary depth of 1 – forces emphasis of the offsets outlined by  $\frac{6}{8}$ 's two depth-1  $\frac{3}{8}$ -duration nodes:  $\frac{6}{8}$ ,  $\frac{3}{8}$  and  $\frac{6}{8}$ . Only the inner measure's contents align perfectly these offsets, so the outer measures are rewritten, better demonstrating  $\frac{6}{8}$ 's duple-ness:

```
>>> staff = Staff(context_name='RhythmicStaff')
>>> staff.extend("{ c'2 c'4 } { c'4. c'4. } { c'2 ~ c'8 c'8 }")
>>> attach(TimeSignature((6, 8)), staff)
>>> for container in staff:
...     mutate(container).rewrite_meter((6, 8), boundary_depth=1)
...
>>> show(staff)
```



## 3.8 FINDING METERS

Not only can meters be used to alter rhythmic structures, they can also be derived from them. A meter's weighted-offset pattern can be used as one-dimensional kernel, or convolution matrix, to determine how strongly an arbitrary collection of offsets appears to express that meter. Given a collection of meters to choose from, each meter can be matched against those offsets and the meter most closely aligning selected as the winner. This process is called *meter fitting*.

### 3.8.1 OFFSET COUNTERS

Before convolving a meter with a collection of offsets, those offsets need to be extracted and counted. Abjad's `metertools` provides an `OffsetCounter` class which maps offsets against counts. Offset counters can be instantiated from any expression containing offsets or whose elements can be expressed as timespans and therefore possess both start and stop offsets. Offsets which appear multiple times in the input expression will result in a higher count in the offset counter, and will in turn have a greater influence during meter fitting.

Consider the following score example:

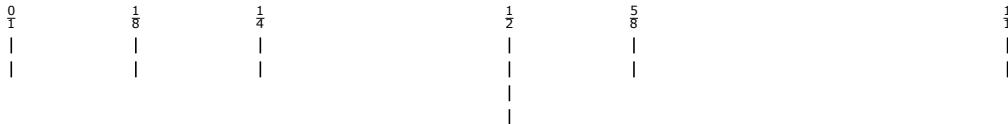
```
>>> upper_staff = Staff("c'8 d'4. e'8 f'4.")
>>> lower_staff = Staff(r'\clef bass c4 b,4 a,2')
>>> piano_staff = scoretools.StaffGroup(
...     [upper_staff, lower_staff],
...     context_name='PianoStaff',
... )
>>> show(piano_staff)
```



The start and stop offsets of all of the leaves of this score can be counted by selecting the score's leaves and instantiating an offset counter from them. Because all score components can be expressed as timespans via `inspect_(some_component).get_timespan()` the offset counter can retrieve both their start and stop offsets within the score:

```
>>> leaves = piano_staff.select_leaves(allow_discontiguous_leaves=True)
>>> piano_staff_counter = metertools.OffsetCounter(leaves)
>>> print(format(piano_staff_counter))
metertools.OffsetCounter(
{
    durationtools.Offset(0, 1): 2,
    durationtools.Offset(1, 8): 2,
    durationtools.Offset(1, 4): 2,
    durationtools.Offset(1, 2): 4,
    durationtools.Offset(5, 8): 2,
    durationtools.Offset(1, 1): 2,
}
)
```

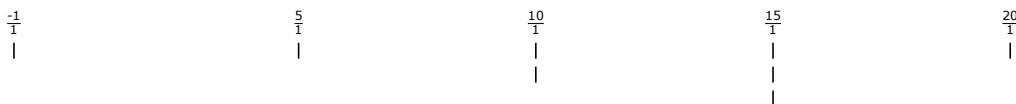
```
>>> show(piano_staff_counter)
```



Note that the offset  $\frac{1}{2}$  shows a count of 4. This is because  $\frac{1}{2}$  acts as both the start or stop offset for four separate leaves in the score.

Offset counters can also be generated from timespan inventories, allowing meter convolution to be used without reference to any score objects at all:

```
>>> timespans = timespantools.TimespanInventory([
...     timespantools.Timespan(-1, 10),
...     timespantools.Timespan(5, 15),
...     timespantools.Timespan(15, 20),
...     timespantools.Timespan(10, 15),
... ])
>>> timespan_counter = metertools.OffsetCounter(timespans)
>>> show(timespan_counter)
```



### 3.8.2 METRIC ACCENT KERNELS

As demonstrated earlier, Abjad's model of meter describes a sequence of offsets with varying degrees of weight – accentedness – attributed to each offset, as determined by the hierarchical tree structure of that meter. This model allows us to explain how downbeats have a stronger weight than upbeats, and how the  $\frac{3}{8}$  offset in a  $\frac{6}{8}$  measure is less strong than its downbeat but still stronger than the offsets at  $\frac{1}{8}$ ,  $\frac{2}{8}$ ,  $\frac{4}{8}$  or  $\frac{5}{8}$ . However, the default model of a  $\frac{6}{8}$  meter makes no explicit reference to offsets such as  $\frac{1}{16}$  or  $\frac{3}{16}$ . A common practice understanding of meter tells us that these offsets should be taken as less accented than those at  $\frac{1}{8}$ ,  $\frac{2}{8}$ ,  $\frac{3}{8}$  and so forth. They effectively represent an even lower level of leaves on the rhythm-tree for  $\frac{6}{8}$ . Likewise, offsets with a denominator of 32 or 64 should be explainable in an identical fashion.

Abjad's `MetricAccentKernel` class provides an object model for both the act of progressively subdividing the weighted offsets of a meter down to some arbitrary denominator, and for the process of convolving those offsets as a one-dimensional convolution kernel against an offset counter. Metric accent kernels allow meters to be fitted against offset counters containing offsets not explicitly modeled by the rhythm-trees of those same meters.

Here a metric accent kernel is generated from a  $\frac{4}{4}$  meter, extending its denominator to a *limit* of  $\frac{1}{16}$ . The weights at each offset are normalized such that they sum to 1. Normalization prevents very long meters from having undue influence during meter fitting:

```
>>> meter = metertools.Meter((4, 4))
>>> kernel_44 = metertools.MetricAccentKernel.from_meter(meter, denominator=16)
>>> for offset, weight in sorted(kernel_44.kernel.items()):
...     print('{!s}\t{!s}'.format(offset, weight))
...
0^^I4/33
1/16^^I1/33
1/8^^I2/33
3/16^^I1/33
1/4^^I1/11
5/16^^I1/33
3/8^^I2/33
7/16^^I1/33
1/2^^I1/11
9/16^^I1/33
5/8^^I2/33
11/16^^I1/33
3/4^^I1/11
13/16^^I1/33
7/8^^I2/33
15/16^^I1/33
1^^I4/33
```

The  $\frac{4}{4}$  metric accent kernel can be called against an offset counter – as though it were a function – to generate the convolution response. The count at each offset in the input offset counter is multiplied against the weight at the corresponding offset in the metric accent kernel. If no corresponding offset exists in the kernel, the weight is taken as 0. The weighted counts are then added together and returned:

```
>>> response = kernel_44(piano_staff_counter)
>>> float(response)
0.5454545454545454
```

The following loop demonstrates the logic underlying the above meter convolution process example:

```
>>> total = Multiplier(0, 1)
>>> for offset, count in sorted(piano_staff_counter.items()):
...     weight = Multiplier(0, 1)
...     if offset in kernel_44.kernel:
...         weight = kernel_44.kernel[offset]
...     weighted_count = weight * count
...     total += weighted_count
...     message = '{!s}:\tcount: {}, weight: {!s}, multiplied: {!s}, total: {!s}'
...     message = message.format(offset, count, weight, weighted_count, total)
...     print(message)
...
0:^^Icount: 2, weight: 4/33, multiplied: 8/33, total: 8/33
1/8:^^Icount: 2, weight: 2/33, multiplied: 4/33, total: 4/11
1/4:^^Icount: 2, weight: 1/11, multiplied: 2/11, total: 6/11
1/2:^^Icount: 4, weight: 1/11, multiplied: 4/11, total: 10/11
5/8:^^Icount: 2, weight: 2/33, multiplied: 4/33, total: 34/33
1:^^Icount: 2, weight: 4/33, multiplied: 8/33, total: 14/11
```

Now consider the metric accent kernels for  $\frac{3}{4}$ ,  $\frac{7}{8}$  and  $\frac{5}{4}$  meters:

```
>>> kernel_34 = metertools.MetricAccentKernel.from_meter((3, 4), denominator=16)
>>> kernel_78 = metertools.MetricAccentKernel.from_meter((7, 8), denominator=16)
>>> kernel_54 = metertools.MetricAccentKernel.from_meter((5, 4), denominator=16)
```

A convolution response can be generated for each of these kernels against the piano staff offset counter:

```
>>> float(kernel_34(piano_staff_counter))
0.5384615384615384

>>> float(kernel_78(piano_staff_counter))
0.4482758620689655

>>> float(kernel_54(piano_staff_counter))
0.4186046511627907
```

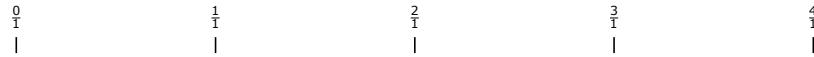
Note that the previously recorded response for a  $\frac{4}{4}$  meter is still higher than any of these three responses.

### 3.8.3 METER FITTING

Meter fitting involves the progressive comparison of a collection of permitted meters against the offsets and counts in an offset counter. Starting from the lowest offset, the metric accent kernel for each meter is convolved with all those offsets in the offset counter with which it overlaps. The response from each convolution is recorded and the meter with the highest associated response is selected to represent those offsets. The process repeats, starting at the right-most offset of the last meter selected, until no more offsets remain to compare against.

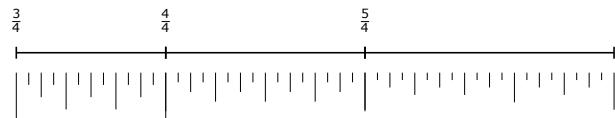
Consider the following offsets, which are separated from one another by the duration of a whole note, outlining the start and stop offsets of a series of  $\frac{4}{4}$  meters:

```
>>> offset_counter = metertools.OffsetCounter([
...     (0, 4), (4, 4), (8, 4), (12, 4), (16, 4),
... ])
>>> show(offset_counter, range_=(0, 5))
```

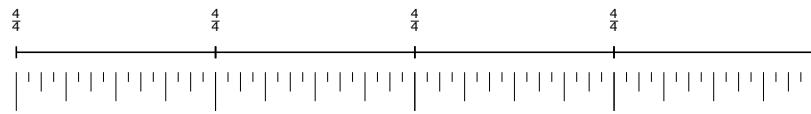


Given the following collection of meters, after meter fitting is performed only  $\frac{4}{4}$  meters should be selected as they perfectly align with the previously defined collection of offsets:

```
>>> permitted_meters = metertools.MeterInventory([(3, 4), (4, 4), (5, 4)])
>>> show(permitted_meters, range_=(0, 5))
```



```
>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     meters=permitted_meters,
... )
>>> show(fitted_meters, range_=(0, 5))
```

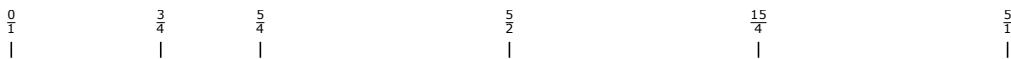


If we change the input offsets to no longer outline only  $\frac{4}{4}$  meters, the meter fitting process will arrive at a different solution. Note how the following fitted meters emphasize the  $\frac{3}{4}$  and  $\frac{5}{4}$  durations inherent to the new input offsets. The initial  $\frac{3}{4}$  fitted meter perfectly matches the offsets  $0\frac{1}{1}$  and  $3\frac{1}{4}$ , while the  $\frac{4}{4}$  meter after it matches the  $5\frac{1}{2}$  offset against its third beat. The next meter's end-beat offset aligns against the offset counter's  $5\frac{1}{2}$  offset allowing the meter fitting process to perfectly match the remaining offsets against a pair of  $\frac{5}{4}$  meters:

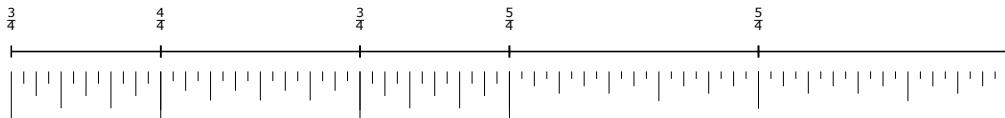
```

>>> offset_counter = metertools.OffsetCounter([
...     (0, 4), (3, 4), (5, 4), (10, 4), (15, 4), (20, 4),
... ])
>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     meters=permitted_meters,
... )
>>> show(offset_counter, range_=(0, 5))

```



```
>>> show(fitted_meters, range_=(0, 5))
```

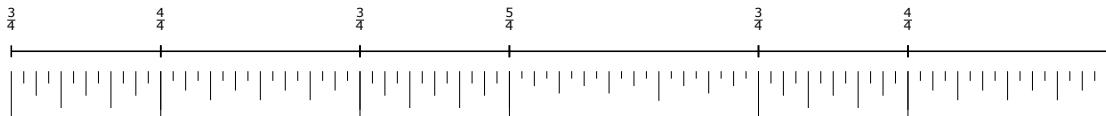


Meter fitting can also control for how many times meters are permitted to immediately repeat. The above example ends with two  $\frac{5}{4}$  meters in a row. If desired, this repetition can be prevented entirely by the `maximum_run_length` keyword to `Meter.fit_meters_to_expr()`. With `maximum_run_length` set to 1, the meter fitting process can no longer select two  $\frac{5}{4}$  meters for the end of the fitted meter sequence. Instead, a pair of  $\frac{3}{4}$  and  $\frac{4}{4}$  meters replace the previously final  $\frac{5}{4}$  meter:

```

>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     meters=permitted_meters,
...     maximum_run_length=1,
... )
>>> show(fitted_meters, range_=(0, 5))

```



### 3.8.4 EXAMPLES

The output of timespan-makers motivates meter fitting more forcefully than the above trivial examples. Densely layered timespan textures often involve hundreds or even thousands of offsets with many points of simultaneity or overlap. Careful management of meter fitting can result in convincing metrical solutions to such textures.

Recall the quartet timespan texture from earlier in this chapter:

```

>>> music_specifiers = collections.OrderedDict([
...     ('Voice 1', None),
...     ('Voice 2', None),
... ])

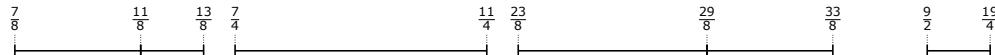
```

```

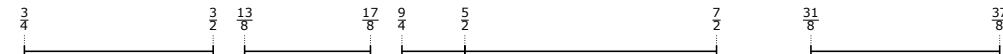
...     ('Voice 3', None),
...     ('Voice 4', None),
...     ])
>>> target_timespan = timespantools.Timespan(0, (19, 4))
>>> timespan_inventory = talea_timespan_maker(
...     music_specifiers=music_specifiers,
...     target_timespan=target_timespan,
...     )
>>> show(timespan_inventory, key='voice_name')

```

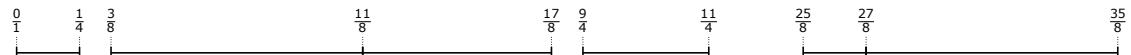
Voice 1:



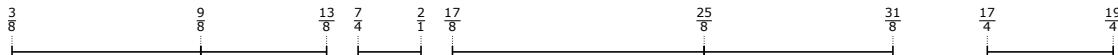
Voice 2:



Voice 3:



Voice 4:

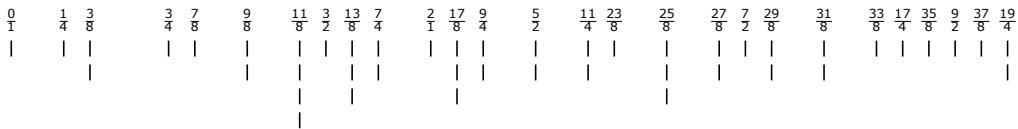


Constructing an offset counter from this timespan inventory shows several offsets with strong simultaneities, notably at  $1\frac{1}{8}$ ,  $1\frac{3}{8}$ ,  $1\frac{7}{8}$  and  $2\frac{5}{8}$ :

```

>>> offset_counter = metertools.OffsetCounter(timespan_inventory)
>>> show(offset_counter, range_=(0, (19, 4)))

```



As before, a collection of permitted meters are fit against the offset counter. Compare the weights in the counter to the meter sequence below it. Offsets in the counter with weights greater than 1 tend to have been fitted against strong beats in each meter. The initial  $\frac{6}{8}$  meter receive a 2-count accent on its half-way  $\frac{3}{8}$  offset. The second  $\frac{6}{8}$  meter is matches strong accents on both its downbeat, its half-beat and on its end-beat. Likewise, the  $\frac{7}{8}$  meter matches strong accents on its down-beat and end-beat, as well as its inner 2-count groupings:

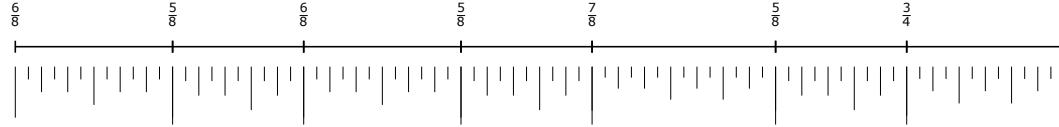
```

>>> permitted_meters = metertools.MeterInventory([
...     (5, 8), (3, 4), (6, 8), (7, 8), (4, 4),
...     ])
>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     meters=permitted_meters,
...     maximum_run_length=1,
...     )
>>> show(offset_counter, range_=(0, (19, 4)))

```

0	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{4}$	$\frac{7}{8}$	$\frac{9}{8}$	$\frac{11}{8}$	$\frac{3}{2}$	$\frac{13}{8}$	$\frac{7}{4}$	$\frac{2}{1}$	$\frac{17}{8}$	$\frac{9}{4}$	$\frac{5}{2}$	$\frac{11}{4}$	$\frac{23}{8}$	$\frac{25}{8}$	$\frac{27}{8}$	$\frac{7}{2}$	$\frac{29}{8}$	$\frac{31}{8}$	$\frac{33}{8}$	$\frac{17}{4}$	$\frac{35}{8}$	$\frac{9}{2}$	$\frac{37}{8}$	$\frac{19}{4}$

```
>>> show(fitted_meters, range_=(0, (19, 4)))
```

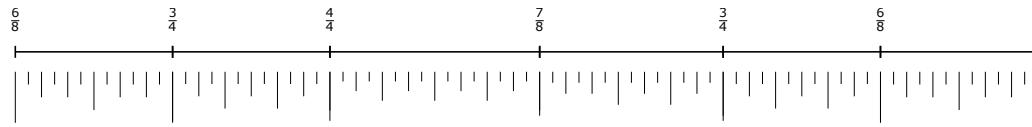


Removing the  $\frac{5}{8}$  meter from the permitted meters inventory gives less convincing results. Notably, the strong simultaneities at offsets  $1\frac{1}{8}$  and  $1\frac{7}{8}$  no longer align with any downbeats in the fitted meters:

```
>>> permitted_meters = metertools.MeterInventory([
...     (3, 4), (6, 8), (7, 8), (4, 4),
... ])
>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     maximum_run_length=1,
...     meters=permitted_meters,
... )
>>> show(offset_counter, range_=(0, (19, 4)))
```

0	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{4}$	$\frac{7}{8}$	$\frac{9}{8}$	$\frac{11}{8}$	$\frac{3}{2}$	$\frac{13}{8}$	$\frac{7}{4}$	$\frac{2}{1}$	$\frac{17}{8}$	$\frac{9}{4}$	$\frac{5}{2}$	$\frac{11}{4}$	$\frac{23}{8}$	$\frac{25}{8}$	$\frac{27}{8}$	$\frac{7}{2}$	$\frac{29}{8}$	$\frac{31}{8}$	$\frac{33}{8}$	$\frac{17}{4}$	$\frac{35}{8}$	$\frac{9}{2}$	$\frac{37}{8}$	$\frac{19}{4}$

```
>>> show(fitted_meters, range_=(0, (19, 4)))
```

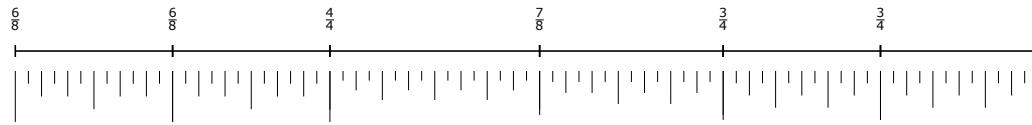


Lifting the constraint on meter repetition does not improve the results. Both the  $1\frac{1}{8}$  and  $1\frac{7}{8}$  offsets still align with weak beats in the fitted meters. The only real change is the swapping of  $\frac{6}{8}$  and  $\frac{3}{4}$  meters in the first and last pairs of fitted meters:

```
>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     meters=permitted_meters,
... )
>>> show(offset_counter, range_=(0, (19, 4)))
```

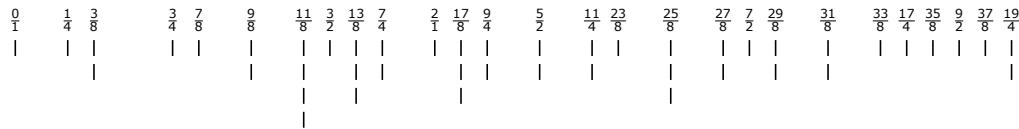
0	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{4}$	$\frac{7}{8}$	$\frac{9}{8}$	$\frac{11}{8}$	$\frac{3}{2}$	$\frac{13}{8}$	$\frac{7}{4}$	$\frac{2}{1}$	$\frac{17}{8}$	$\frac{9}{4}$	$\frac{5}{2}$	$\frac{11}{4}$	$\frac{23}{8}$	$\frac{25}{8}$	$\frac{27}{8}$	$\frac{7}{2}$	$\frac{29}{8}$	$\frac{31}{8}$	$\frac{33}{8}$	$\frac{17}{4}$	$\frac{35}{8}$	$\frac{9}{2}$	$\frac{37}{8}$	$\frac{19}{4}$

```
>>> show(fitted_meters, range_=(0, (19, 4)))
```

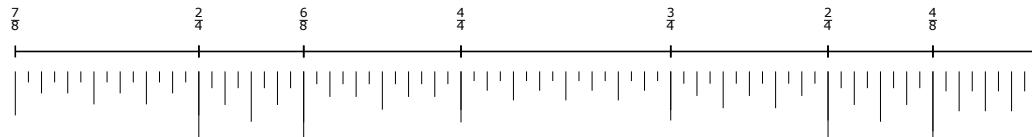


However, reinstating the repetition constraint and permitting a greater variety of meters gives an even closer fitting than the very first example. Not only do the  $1\frac{1}{8}$  and  $1\frac{7}{8}$  offsets align at downbeats, but the three-weight offset at  $2\frac{5}{8}$  matches the downbeat of a  $\frac{3}{4}$  meter with two-weight accents on both its second and third beat as well as the following downbeat:

```
>>> permitted_meters = metertools.MeterInventory([
...     (2, 4), (4, 8), (3, 4), (6, 8), (7, 8), (4, 4),
... ])
>>> fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     maximum_run_length=1,
...     meters=permitted_meters,
... )
>>> show(offset_counter, range_=(0, (19, 4)))
```

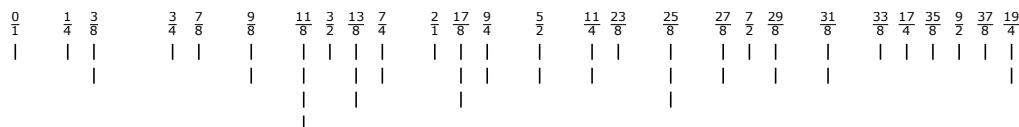


```
>>> show(fitted_meters, range_=(0, (19, 4)))
```

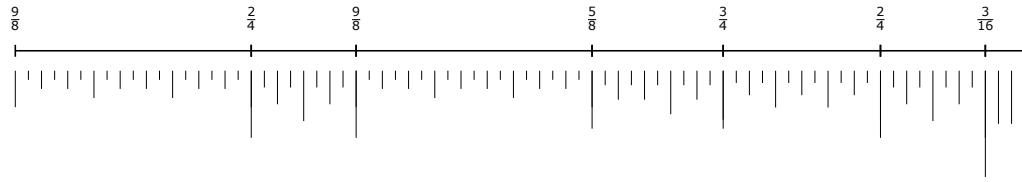


Increasing the number of permitted offsets even further does not necessarily improve results further. A better initial meter fitting – like the initial  $\frac{9}{8}$  here – often diminishes the choices possible afterward:

```
>>> permitted_meters = metertools.MeterInventory([
...     (2, 4), (3, 16), (3, 4), (4, 4), (4, 8), (5, 16),
...     (5, 4), (5, 8), (6, 8), (7, 8), (9, 8),
... ])
>>> poorly_fitted_meters = metertools.Meter.fit_meters_to_expr(
...     expr=offset_counter,
...     maximum_run_length=1,
...     meters=permitted_meters,
... )
>>> show(offset_counter, range_=(0, (19, 4)))
```



```
>>> show(poorly_fitted_meters, range_=(0, (19, 4)))
```



The process of meter fitting described here is not perfect. While useful for creating generally convincing metrical solutions to dense timespan structures, meter fitting could probably be better solved through human intervention. The fitting process often over-emphasizes local attack point maxima while ignoring larger or more elegant metrical patterns. Likewise, the process often provides an accurate fitting for the very first meter, but causes a cascade of poor solutions for every following meter. Where procedural meter fitting wins over human intervention is in speed. Solutions to enormous timespan inventories with hundreds or thousands of timespans can be found in less than a second, facilitating the rapid sketching and revising of timespan-based musical structures.

Meter fitting can be considered as a kind of constraint problem. The set of permitted meters act as a search space while each progressive selection of offsets from the input offset counter act as the problem to solve against. In that light, improvements might involve searching for meter solutions with increased *lookahead* – that is, fitting the current meter based not only on its response to a selection of offsets, but also on how well any meter following it would score. Searching the offset counter for patterns, changing the weighting algorithm of metric accent kernels, or jumping directly to the attack point maxima in the input offset counter and solving forwards and backwards from them are also possible avenues for improvement.

### 3.9 SYNTHESIZING TIME TECHNIQUES

The techniques outlined in this chapter – timespan inventories, timespan-makers, rhythm-makers, meter fitting and rewriting – describe various ways of modeling, creating and manipulating aspects of musical score. Taken separately, none of them can ever result in the rhythmic framework for a polyphonic piece of music. However, by combining all of them together, it is possible to construct powerful tools for generating arbitrarily large amounts of notation. The following `build_score()` function sketches one possible approach to combining these techniques:

```
def build_score(  
    performed_rhythm_maker,  
    permitted_meters,  
    score_template,  
    timespan_inventory,  
):
```

```

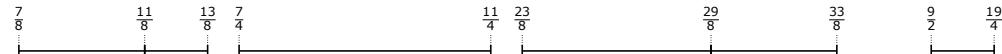
fitted_meters, meter_boundaries = get_meters_and_meter_boundaries(
    timespan_inventory, permitted_meters)
all_voicewise_timespans = get_all_voicewise_timespans(timespan_inventory)
seed = 0
score = score_template()
for voice in iterate(score).by_class(Voice):
    if voice.name not in all_voicewise_timespans:
        all_voicewise_timespans[voice.name] = \
            timespantools.TimespanInventory()
    voice_timespans = all_voicewise_timespans[voice.name]
    previous_stop_offset = Offset(0)
    for shard in voice_timespans.partition(include_tangent_timespans=True):
        if shard.start_offset != previous_stop_offset:
            silent_music = make_silent_music(
                meter_boundaries=meter_boundaries,
                start_offset=previous_stop_offset,
                stop_offset=shard.start_offset,
            )
            voice.append(silent_music)
    performed_music = make_performed_music(
        meter_boundaries=meter_boundaries,
        rhythm_maker=performed_rhythm_maker,
        seed=seed,
        timespans=shard,
    )
    voice.append(performed_music)
    seed += 1
    previous_stop_offset = shard.stop_offset
if previous_stop_offset != meter_boundaries[-1]:
    silent_music = make_silent_music(
        meter_boundaries=meter_boundaries,
        start_offset=previous_stop_offset,
        stop_offset=meter_boundaries[-1],
    )
    voice.append(silent_music)
for phrase in voice:
    rewrite_meters(phrase, fitted_meters, meter_boundaries)
add_time_signature_context(score, fitted_meters)
return score

```

`build_score()` creates a score from a rhythm-maker, a sequence of permitted meters, a score template and a timespan inventory. The timespan inventory must contain performed timespans whose voice names align with the voice names in the score produced by the score template. For example, Abjad's `GroupedRhythmicStavesScoreTemplate` class can produce scores containing voices with names like "Voice 1", "Voice 2" and so forth. The timespan inventory used for this example – created earlier in this chapter – contains performed timespans with the voice names "Voice 1", "Voice 2", "Voice 3" and "Voice 4", requiring a score template capable of creating scores with four staves of one voice each:

```
>>> show(timespan_inventory, key='voice_name')
```

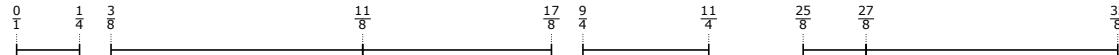
Voice 1:



Voice 2:



Voice 3:



Voice 4:



```
>>> score_template = templatetools.GroupedRhythmicStavesScoreTemplate(  
...     staff_count=4, with_clefs=True,  
...     )  
>>> score = score_template()  
>>> print(format(score))  
\context Score = "Grouped Rhythmic Staves Score" <<  
  \context StaffGroup = "Grouped Rhythmic Staves Staff Group" <<  
    \context RhythmicStaff = "Staff 1" {  
      \clef "percussion"  
      \context Voice = "Voice 1" {  
      }  
    }  
    \context RhythmicStaff = "Staff 2" {  
      \clef "percussion"  
      \context Voice = "Voice 2" {  
      }  
    }  
    \context RhythmicStaff = "Staff 3" {  
      \clef "percussion"  
      \context Voice = "Voice 3" {  
      }  
    }  
    \context RhythmicStaff = "Staff 4" {  
      \clef "percussion"  
      \context Voice = "Voice 4" {  
      }  
    }  
>>  
>>
```

Score building proceeds by first fitting the permitted meters against the timespan inventory and calculating their *meter boundaries* – the offsets of each downbeat, as well as the virtual downbeat, the “end-beat”, after the final meter. Then the timespans in the input timespan inventory are separated into new timespan inventories according to their associated voice names, with these new inventories stored in a dictionary whose keys are those same voice names. With the timespan and metrical structures organized, the `build_score()` produces an unpopulated score from its score template. For each voice in this score, an inventory of timespans is retrieved from the `all_voicewise_-`

timespans dictionary. If no inventory exists, an empty one is created. The associated performed timespans are partitioned into contiguous groups in order to facilitate rhythm generation. Each partitioned group is passed, along with the input rhythm-maker, the fitted meter boundaries, and a seed value – the number of performed timespan groups encountered so far – to a rhythm generation subroutine, with the resulting phrase container appended into the current voice. If a gap is encountered between two partitioned groups, between the beginning of the first partitioned group and the beginning of the score – the offset 0, or between the end of the last partitioned group and the final meter boundary, "silent music" will be created in the form of rests grouped into a container, with that container then inserted into the current voice. Once all voices in the score have been populated, their meters are rewritten, then the score is formatted and finally returned.

An analysis of each of the functions called within `build_score()` follows.

### 3.9.1 ORGANIZING METER

Consider the first function called inside `build_score()`, `get_meters_and_meter_boundaries()`. A meter sequence of equal or greater duration to the input timespan inventory can be produced through fitting a collection of permitted meters against that same input timespan inventory. The sequence of offsets found at the boundaries of each fitted meter can then be determined by computing the cumulative sums of the durations of the fitted meters. These boundary offsets will be used to split timespans before they are fed to rhythm-makers for rhythm generation in order to ensure that no generated rhythm crosses any bar-lines:

```
def get_meters_and_meter_boundaries(timespan_inventory, permitted_meters):
    offset_counter = metertools.OffsetCounter(timespan_inventory)
    fitted_meters = metertools.Meter.fit_meters_to_expr(
        expr=offset_counter,
        maximum_run_length=1,
        meters=permitted_meters,
    )
    meter_durations = [Duration(_) for _ in fitted_meters]
    meter_boundaries = mathtools.cumulative_sums(
        meter_durations,
        start=Offset(0),
    )
    return fitted_meters, meter_boundaries
```

### 3.9.2 ORGANIZING TIMESPANS

Each of the performed timespans in the input timespan inventory was configured with a voice name, allowing that timespan to be associated with a voice context in the score hierarchy produced by the input score template. As

demonstrated earlier, Abjad components can be indexed by context name regardless of the depth of the named context in the score hierarchy:

```
>>> timespan = timespan_inventory[0]
>>> print(format(timespan))
consort.tools.PerformedTimespan(
    start_offset=durationtools.Offset(0, 1),
    stop_offset=durationtools.Offset(1, 4),
    original_start_offset=durationtools.Offset(9, 2),
    original_stop_offset=durationtools.Offset(19, 4),
    voice_name='Voice 3',
)

>>> score = score_template()
>>> voice = score[timespan.voice_name]
>>> for component in inspect_(voice).get_parentage():
...     component
...
Voice()
<RhythmicStaff-'Staff 3'{1}>
<StaffGroup-'Grouped Rhythmic Staves Staff Group'<<4>>>
<Score-'Grouped Rhythmic Staves Score'<<1>>>
```

In order to populate each voice in the score, the timespans in the timespan inventory need to be separated into *voice-wise* timespan inventories, each only containing timespans associated with the same voice. This can be accomplished by building a dictionary whose keys are voice names and whose values are timespan inventories:

```
def get_all_voicewise_timespans(timespan_inventory):
    voicewise_timespans = {}
    for timespan in timespan_inventory:
        voice_name = timespan.voice_name
        if voice_name not in voicewise_timespans:
            voicewise_timespans[voice_name] = timespantools.TimespanInventory()
            voicewise_timespans[voice_name].append(timespan)
    return voicewise_timespans
```

### 3.9.3 POPULATING VOICES

Creating *performed* music requires first splitting a contiguous inventory of performed timespans by meter boundaries, collecting the shards together into a new inventory and then calculating the durations of those timespans. The resulting sequence of durations, along with an input rhythm-maker and an optional seed value can be sent to another function for rhythm generation:

```
def make_performed_music(rhythm_maker, timespans, meter_boundaries, seed=0):
    split_timespans = timespantools.TimespanInventory()
```

```

for shard in timespans.split_at_offsets(meter_boundaries):
    split_timespans.extend(shard)
durations = [_.duration for _ in split_timespans if _.duration]
music = make_music(rhythm_maker, durations, seed=seed)
return music

```

Similarly, *silent* music – rests between groups of performed timespans in each voice – can be created by constructing a timespan which outlines the gap between other performed timespans, splitting that timespan by meter boundaries, collecting the durations of the split shards, and finally passing those durations along with a rest-generating rhythm-maker to another function for rhythm generation. As demonstrated in subsection 3.5.1, rest-generating rhythm-makers can be created from note rhythm-makers which use output masks to silence all output divisions:

```

def make_silent_music(start_offset, stop_offset, meter_boundaries):
    silence_timespan = timespantools.Timespan(start_offset, stop_offset)
    shards = silence_timespan.split_at_offsets(meter_boundaries)
    durations = [_.duration for _ in shards if _.duration]
    mask = rhythmmakertools.silence_all()
    rhythm_maker = rhythmmakertools.NoteRhythmMaker(output_masks=[mask])
    music = make_music(rhythm_maker, durations)
    return music

```

Both silent and performed music generation relies on the same core rhythm generating function, `make_music()`. This function calls its input rhythm-maker with a sequence of durations and an optional seed value, which may be interpreted variably depending on what kind of rhythm-maker was used, but which generally results in the rotation of any sequence-like configured value in that rhythm-maker, such as talea. The product of this first operation is a sequence of component selections. Some massaging converts this selections into a sequence of trivially-prolated containers and non-trivially prolated tuplets which can then be wrapped inside a larger container representing a complete phrase:

```

def make_music(rhythm_maker, durations, seed=0):
    music = rhythm_maker(durations, rotation=seed)
    for i, division in enumerate(music):
        if len(division) == 1 and isinstance(division[0], scoretools.Tuplet):
            music[i] = division[0]
        else:
            music[i] = scoretools.Container(division)
    music = scoretools.Container(music)
    return music

```

### 3.9.4 REWRITING METERS

After all phrases for a voice have been generated, their meters are rewritten. This process involves simultaneously iterating through both the divisions in each phrase along with pairs of meters and meter downbeat offsets – that

is, the offset from the origin of the score where each meter begins.<sup>13</sup> Unprolated division containers are rewritten according to the meter active when they begin. Because a division may start *after* a meter has begun it is necessary to calculate the difference between each unprolated division's start offset and each meter's start offset. This difference is passed to the `MutationAgent`'s `rewrite_meter()` method via the `initial_offset` keyword, allowing the division to be properly aligned against the desired meter. Prolated divisions – tuples – are rewritten solely with respect to the duration of their contents, not to the prevailing meter. That is, a 5:4 tuplet in a  $\frac{4}{8}$  measure is rewritten with a meter of  $\frac{5}{8}$ , not of  $\frac{4}{8}$ :

```
def rewrite_meters(phrase, meters, meter_boundaries):
    pairs = list(zip(meters, meter_boundaries))
    for division in phrase:
        division_offset = inspect_(division).get_timespan().start_offset
        while 1 < len(pairs) and pairs[1][1] <= division_offset:
            pairs.pop(0)
        if isinstance(division, scoretools.Tuplet):
            contents_duration = division._contents_duration
            meter = metertools.Meter(contents_duration)
            mutate(division).rewrite_meter(
                meter,
                boundary_depth=1,
            )
        else:
            meter, meter_boundary = pairs[0]
            initial_offset = division_offset - meter_boundary
            mutate(division).rewrite_meter(
                meter,
                boundary_depth=1,
                initial_offset=initial_offset,
            )
```

### 3.9.5 SCORE POST-PROCESSING

The final step in this score building process adds a “floating time signature context” to the score, filled with typographic spacer skips and time signature commands, one for each of the fitted meters. This context appears as a row of time signatures floating above the topmost staff in the score, allowing time signatures to be omitted from every staff in the score and thereby improving proportional notation spacing. The time signature context's `context_name` property instructs LilyPond to look for a context definition with that name and apply any typographic overrides found there during the typesetting process. While LilyPond does not come packaged with a `TimeSignatureContext`

---

<sup>13</sup>More optimizations to this joint container / meter iteration are obviously possible. Some have even been implemented in Consort's source. Such optimization include associating meters with their own timespans, and storing these meter-timespans in an optimized interval-tree data structure allowing rapid retrieval by both offset and timespan intersection. For the sake of pedagogical clarity, a more naive approach is used here.

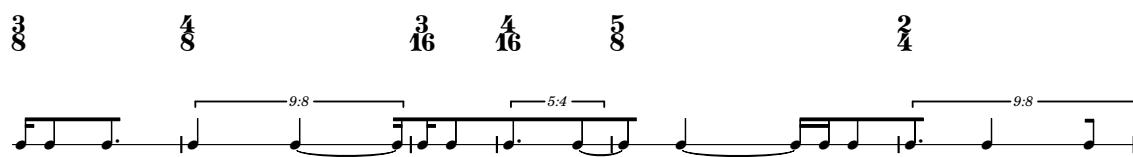
definition, examples of what such a definition looks like can be found in the various stylesheet sections of the source code appendices to this document:

```
def add_time_signature_context(score, meters):
    time_signatures = [_.implied_time_signature for _ in meters]
    measures = scoretools.make_spacer_skip_measures(time_signatures)
    time_signature_context = scoretools.Context(
        [measures],
        context_name='TimeSignatureContext',
        name='TimeSignatureContext',
    )
    score.insert(0, time_signature_context)
```

### 3.9.6 EXAMPLES

Calling the `build_score()` function generates a score as described above. In this example, the function's arguments comprise a talea rhythm-maker, an inventory of meters, as well as the previously defined score template and timespan inventory:

```
>>> permitted_meters = metertools.MeterInventory([
...     (2, 4), (4, 8), (3, 4), (6, 8), (7, 8), (4, 4),
... ])
>>> performed_rhythm_maker = rythmmakertools.TaleaRhythmMaker(
...     beamSpecifier=rythmmakertools.BeamSpecifier(
...         beamEachDivision=True,
...         beamDivisionsTogether=True,
...     ),
...     extraCountsPerDivision=(0, 1),
...     talea=rythmmakertools.Talea([1, 2, 3, 4, 5], 16),
... )
>>> show(performed_rhythm_maker, divisions=divisions)
```



For clarity, the phrases and their internal divisions have been annotated via Consort's `annotate` function. This function draws thick brackets underneath each staff, with the lower bracket indicating the overall phrase grouping and the inner brackets indicating divisions within each phrase. Entirely silent phrasing brackets are drawn with dashed rather than solid lines:

```
>>> score = build_score(
...     performed_rhythm_maker=performed_rhythm_maker,
...     permitted_meters=permitted_meters,
```

```

    ...
    score_template=score_template,
    ...
    timespan_inventory=timespan_inventory,
    ...
)
>>> consort.annotate(score)
>>> show(score)

```

Large-scale variations are possible simply by altering the arguments to the `build_score()` function. For example, the timespan inventory can be reflected around its axis and stretched from a duration of  $19\frac{1}{4}$  to  $25\frac{1}{4}$ . This will change the overall duration and phrase density of the resulting score while maintaining the character of its surface rhythms:

```

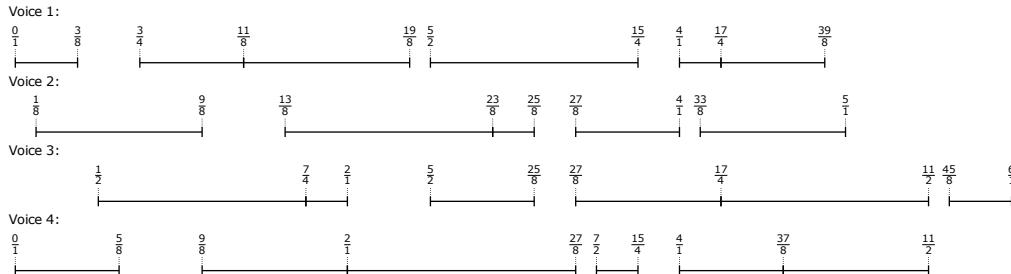
>>> multiplier = Duration(24, 4) / Duration(19, 4)
>>> timespan_inventory = timespan_inventory.reflect()

```

```

>>> timespan_inventory = timespan_inventory.stretch(multiplier)
>>> timespan_inventory = timespan_inventory.round_offsets(Duration(1, 8))
>>> show(timespan_inventory, key='voice_name')

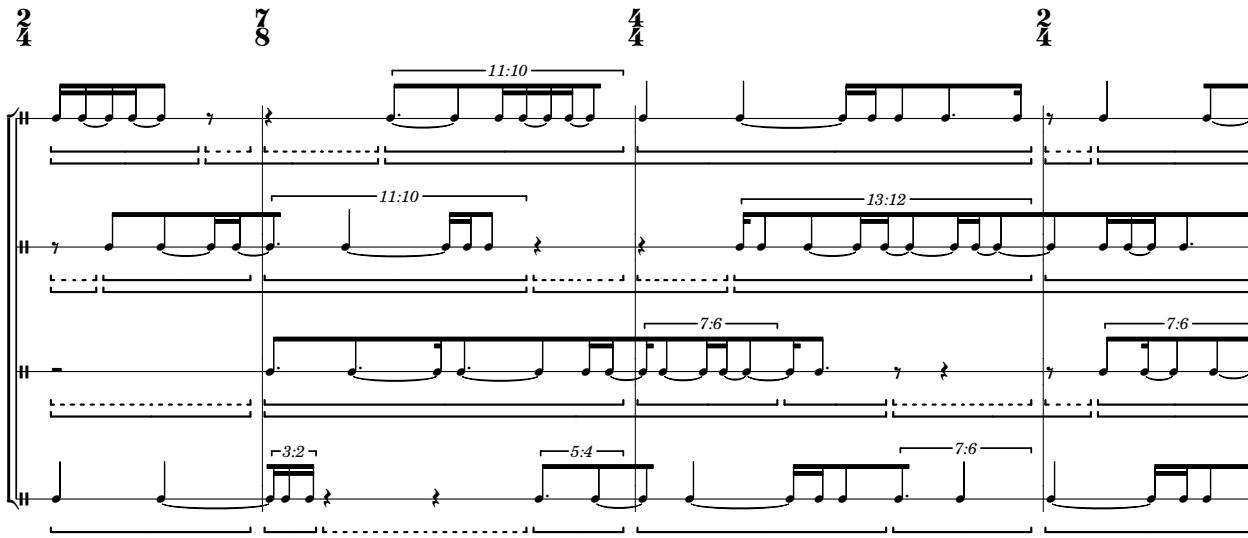
```

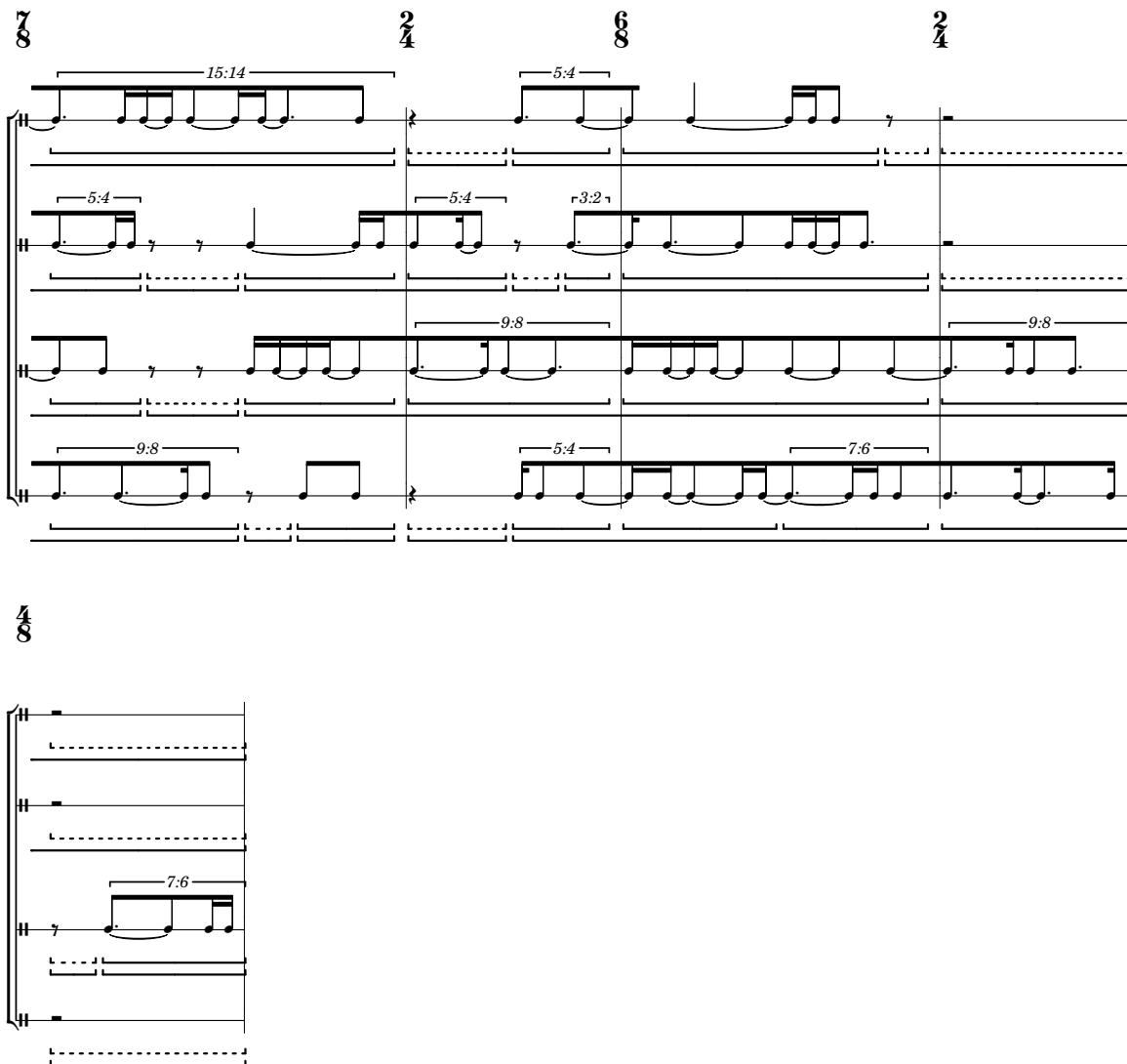


```

>>> score = build_score(
...     performed_rhythm_maker=performed_rhythm_maker,
...     permitted_meters=permitted_meters,
...     score_template=score_template,
...     timespan_inventory=timespan_inventory,
... )
>>> consort.annotate(score)
>>> show(score)

```





Many other variations and extensions to the basic score-building algorithm outlined above are possible. The input rhythm-maker could be varied or replaced by a composite rhythm-maker. The score generation process could be extended so that rhythm-makers could be stored on the performed timespans themselves, allowing each performed timespan to specify its local rhythmic language. Multiple timespan inventories could be combined together using the various logical operations to implement masking or fusing when timespans from one inventory overlap those from another. In fact, all of these extensions and variations are employed in Consort's own score generation process, as described in chapter 4.

This page intentionally left blank.

# 4

## *Consort*: a model of composition



ONSORT, A PYTHON LIBRARY I have written as an extension to Abjad, models the process of composing notated musical scores in terms of a repeated cycle containing two distinct stages: *specification* and *interpretation*. During specification, the composer describes the structure of the score procedurally in terms of what musical materials – themselves modeled as bundles of procedures for generating notation – should appear at particular moments in time. Then, during interpretation, those procedural descriptions are gradually converted into notation, turning the abstract specification of structure into a concrete segment of score. What follows is a detailed analysis of the various algorithms and subroutines employed during Consort’s specification and interpretation stages.

## 4.1 SPECIFICATION

*Specification* describes how *out-of-time materials* – both concrete and programmatic – should be deployed *in-time* in a *segment* of musical score as notation. Materials encompass abstractions – such as pitch sets or collections of performance technique indications – and procedures for producing, altering or embellishing music such as rhythm-makers or attachment-handlers. Score segments comprise any contiguous passage of music, demarcating an area of compositional concern. Consort treats scores as comprised of at least one segment, but potentially many more concatenated together. Any segment may of course contain arbitrarily complex inner structuring. Separation of scores into distinct segments acts then mainly as an aid for the composer, both by simplifying the complexity of the current specification under consideration, and by allowing the typesetting engine – LilyPond – to display more manageable amounts of notation than the full score, thus speeding up the cycle of specifying, interpreting and visualizing.

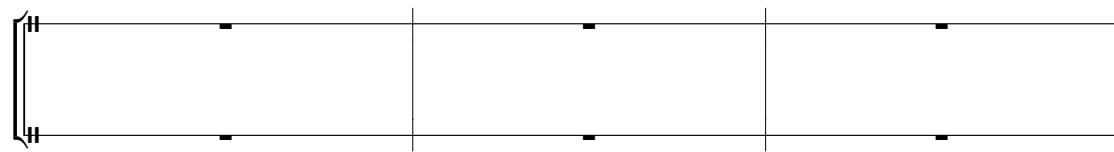
### 4.1.1 SEGMENT-MAKERS

Composers specify segments by creating and progressively configuring *segment-makers*, classes which conceptually mirror the rhythm- and timespan-makers described in chapter 3, but on a much larger scale. Such configuration parameters include tempo, permitted meters, desired duration, and score template. Score templates, introduced in subsection 2.8.4, are notation factories which build scores comprised of staff groups, staves, voices, clefs and instruments, as necessary to model the context hierarchy of a score to which no count-time components have yet been added. All segment-makers in a score project must use the same score template if they are to appear contiguously in the complete typeset score, as this mechanism makes use of LilyPond’s context concatenation behavior, demonstrated in subsection 5.2.3. The following defines a segment-maker with a desired duration of 9 seconds, only  $\frac{3}{4}$  meters permitted, a score template comprised of two rhythmic staves, and a tempo of 60 quarter-notes per minute:

```
>>> segment_maker = consort.SegmentMaker(  
...     desired_duration_in_seconds=9,  
...     permitted_time_signatures=[(3, 4)],  
...     score_template=templatetools.GroupedRhythmicStavesScoreTemplate(  
...         staff_count=2,  
...         with_clefs=True,  
...     ),  
...     tempo=indicatortools.Tempo((1, 4), 60),  
... )
```

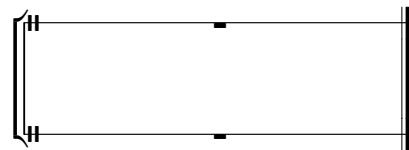
This segment-maker can be illustrated via `show()`, like many other objects in Abjad. Illustration here invokes the segment-maker’s interpretation stage. The `verbose=False` flag prevents it from printing a considerable amount of diagnostic information:

```
>>> show(segment_maker, verbose=False)
```



By changing the tempo from quarter-equals-60 to quarter-equals-20, the overall notated duration of the segment shrinks by two thirds, but the duration in seconds remains the same. This mechanism allows segments to be planned relative one another in terms of their “actual” durations, even if their tempi differ:

```
>>> slower_segment_maker = new(
...     segment_maker,
...     tempo=indicatortools.Tempo((1, 4), 20),
... )
>>> show(slower_segment_maker, verbose=False)
```



Most importantly, segment-makers may be configured with any number of *music settings*, which object-model both *when* and in *which* voices musical materials should be deployed.

#### 4.1.2 MUSIC SETTINGS

Music settings represent a layer of musical texture, in one or more voices, of arbitrary length. Music settings aggregate a timespan-maker, a target timespan and any number of *music specifiers*. The timespan-maker provides the overall phrasing and density structure, the optional timespan identifier defines in what portion of the current segment the timespan-maker’s texture should be spooled out, and the music specifiers define both in which voices the timespan texture should appear as well as how those timespans should ultimately be rendered as notation. The

order in which composers configure segment-makers with music settings defines each music setting’s *layer*. Layer determines how overlapping events in a single voice will mask one another. The first setting defined receives layer 0, the second layer 1 and so forth, with each higher layer number indicating higher precedence or “foregroundness”. The timespans created by music settings which are defined later during segment specification “hide” any timespans created by those music settings defined earlier. Score materials, including music settings, music specifiers, timespan-makers and any other class pertinent to score creation – potentially even other segment-makers, may be defined from scratch in the same code module as the segment-maker currently being configured, templated from another material, or simply imported into the segment definition’s namespace.

#### 4.1.3 MUSIC SPECIFIERS

Music specifiers bundle all of the information necessary for a segment-maker to generate the notational content for a sequence of one or more divisions, grouped as a phrase. This information includes optional rhythm-maker (section 3.5), *grace-handler* (subsection 4.4.2), *pitch-handler* (subsection 4.4.3) and *attachment-handler* (subsection 4.4.4) definitions – all classes which describe strategies for creating or modifying notation –, as well as a variety of other properties including an optional *minimum phrase duration* – described further in subsection 4.3.4 –, and *seed* (subsection 4.3.5).

Music specifiers can also be configured with a *labels*, a tuple of one or more arbitrary strings, identifying some quality of those performed timespans. As demonstrated in subsection 3.4.3, composers configure dependent timespan-makers in order to create timespans according to the disposition of performed timespans associated with specific voices. Dependent timespan-makers can also be configured to select depended-upon timespans based on the *labels* property of those performed timespans’ music specifiers, allowing an additional category for filtering. This helps model creating a pedal voice mirroring not simply a pianist’s left- and right-hand events, but only those that actually depress keys, ignoring guero events or other off-the-key percussive techniques for which no pedaling is desired.

Consider the following timespan inventory, populated with performed timespans associated with one of two voices, and annotated with music specifiers which are either labeled or not:

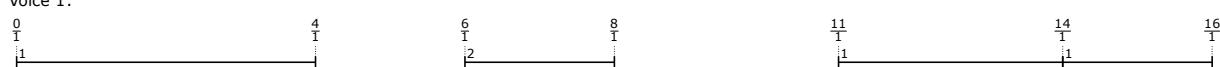
```
>>> unlabeled_musicSpecifier = consort.MusicSpecifier()
>>> labeled_musicSpecifier = consort.MusicSpecifier(labels=['labeled'])
>>> timespanInventory = timespantools.TimespanInventory([
...     consort.PerformedTimespan(
...         layer=1,
...         start_offset=0,
...         stop_offset=4,
```

```

...
    musicSpecifier=labeled_musicSpecifier,
...
    voiceName='Voice 1',
),
)
consort.PerformedTimespan(
    layer=1,
    startOffset=2,
    stopOffset=7,
    musicSpecifier=labeled_musicSpecifier,
    voiceName='Voice 2',
),
)
consort.PerformedTimespan(
    layer=2,
    startOffset=6,
    stopOffset=8,
    musicSpecifier=unlabeled_musicSpecifier,
    voiceName='Voice 1',
),
)
consort.PerformedTimespan(
    layer=2,
    startOffset=10,
    stopOffset=(25, 2),
    musicSpecifier=unlabeled_musicSpecifier,
    voiceName='Voice 2',
),
)
consort.PerformedTimespan(
    layer=1,
    startOffset=11,
    stopOffset=14,
    musicSpecifier=labeled_musicSpecifier,
    voiceName='Voice 1',
),
)
consort.PerformedTimespan(
    layer=1,
    startOffset=14,
    stopOffset=16,
    musicSpecifier=labeled_musicSpecifier,
    voiceName='Voice 1',
),
)
consort.PerformedTimespan(
    layer=1,
    startOffset=15,
    stopOffset=16,
    musicSpecifier=labeled_musicSpecifier,
    voiceName='Voice 2',
),
),
])
)
>>> show(timespanInventory, key='voiceName')

```

Voice 1:

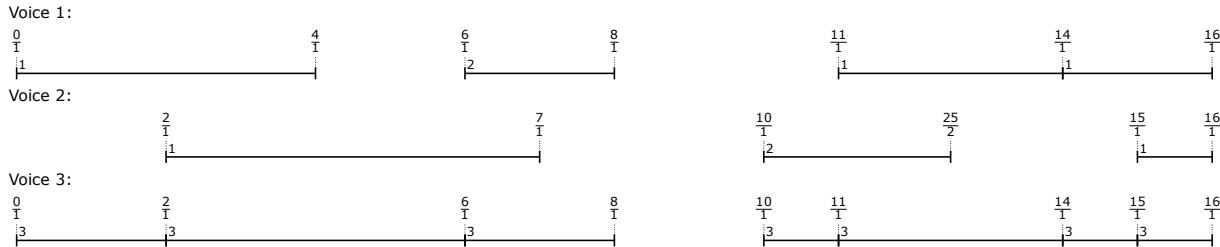


Voice 2:



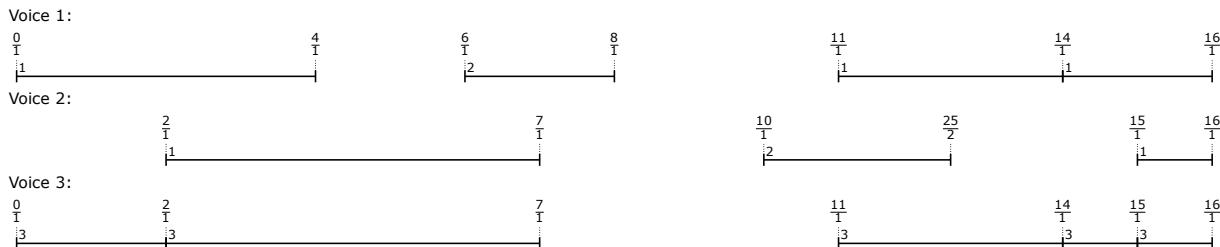
A dependent timespan-maker, configured to select timespans associated with “Voice 1” and “Voice 2” produces dependent timespans in the manner illustrated in subsection 3.4.3:

```
>>> dependent_timespan_maker = consort.DependentTimespanMaker(
...     include_inner_starts=True,
...     voice_names=('Voice 1', 'Voice 2'),
... )
>>> result = dependent_timespan_maker(
...     layer=3,
...     music_specifiers={'Voice 3': None},
...     timespan_inventory=timespan_inventory[:],
... )
>>> show(result, key='voice_name')
```



Reconfiguring the above dependent timespan-maker to additionally filter timespans whose music specifier is labeled “labeled” produces a more restricted output. Note that the unlabeled  $6/1$ - $8/1$  timespan in “Voice 1” and the  $10/1$ - $25/2$  timespan in “Voice 2” are ignored:

```
>>> dependent_timespan_maker = new(
...     dependent_timespan_maker,
...     labels=['labeled'],
... )
>>> result = dependent_timespan_maker(
...     layer=3,
...     music_specifiers={'Voice 3': None},
...     timespan_inventory=timespan_inventory[:],
... )
>>> show(result, key='voice_name')
```



Music specifiers can be grouped into sequences called, unsurprisingly, *music specifier sequences*, allowing a music setting to specify that the timespans it creates for a particular voice should be annotated with multiple different music specifiers in a patterned way:

```

>>> musicSpecifierSequence = consort.MusicSpecifierSequence(
...     musicSpecifiers=['A', 'B', 'C'],
... )

```

Recall from subsection 3.4.2 that talea timespan-makers can create contiguous groups of 1 or more timespan associated with a specific voice. Music specifier sequences can be used to annotate each timespan in a contiguous group with a different music specifier, or to simply choose a different music specifier for each contiguous group as a whole.

This behavior is controlled via the music specifier's *application rate* property:

```

>>> musicSpecifiers = {'Voice': musicSpecifierSequence}
>>> targetTimespan = timespantools.Timespan(0, (7, 4))
>>> timespanMaker = consort.TaleaTimespanMaker(
...     playingGroupings=(3,),
... )
>>> timespanInventory = timespanMaker(
...     musicSpecifiers=musicSpecifiers,
...     targetTimespan=targetTimespan,
... )
>>> print(format(timespanInventory))
timespantools.TimespanInventory(
    [
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(1, 4),
            musicSpecifier='A',
            voiceName='Voice',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 4),
            stop_offset=durationtools.Offset(1, 2),
            musicSpecifier='A',
            voiceName='Voice',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 2),
            stop_offset=durationtools.Offset(3, 4),
            musicSpecifier='A',
            voiceName='Voice',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 1),
            stop_offset=durationtools.Offset(5, 4),
            musicSpecifier='B',
            voiceName='Voice',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(5, 4),
            stop_offset=durationtools.Offset(3, 2),
            musicSpecifier='B',
            voiceName='Voice',
        ),
    ]
)

```

```

),
consort.tools.PerformedTimespan(
    start_offset=durationtools.Offset(3, 2),
    stop_offset=durationtools.Offset(7, 4),
    musicSpecifier='B',
    voice_name='Voice',
),
]
)

```

Changing the application rate from the default value of “phrase” to “division” causes a different music specifier from the sequence to be annotated to each timespan, rather than each contiguous group of timespans:

```

>>> musicSpecifierSequence = new(
...     musicSpecifierSequence,
...     application_rate='division',
... )
>>> musicSpecifiers = {'Voice': musicSpecifierSequence}
>>> timespanInventory = timespanMaker(
...     musicSpecifiers=musicSpecifiers,
...     targetTimespan=targetTimespan,
... )
>>> print(format(timespanInventory))
timespantools.TimespanInventory(
[
    consortium.tools.PerformedTimespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(1, 4),
        musicSpecifier='A',
        voice_name='Voice',
    ),
    consortium.tools.PerformedTimespan(
        start_offset=durationtools.Offset(1, 4),
        stop_offset=durationtools.Offset(1, 2),
        musicSpecifier='B',
        voice_name='Voice',
    ),
    consortium.tools.PerformedTimespan(
        start_offset=durationtools.Offset(1, 2),
        stop_offset=durationtools.Offset(3, 4),
        musicSpecifier='C',
        voice_name='Voice',
    ),
    consortium.tools.PerformedTimespan(
        start_offset=durationtools.Offset(1, 1),
        stop_offset=durationtools.Offset(5, 4),
        musicSpecifier='B',
        voice_name='Voice',
    ),
    consortium.tools.PerformedTimespan(
        start_offset=durationtools.Offset(5, 4),
        stop_offset=durationtools.Offset(3, 2),
        musicSpecifier='B',
        voice_name='Voice',
    ),
]
)

```

```

        musicSpecifier='C',
        voiceName='Voice',
    ),
    consort.tools.PerformedTimespan(
        startOffset=durationtools.Offset(3, 2),
        stopOffset=durationtools.Offset(7, 4),
        musicSpecifier='A',
        voiceName='Voice',
    ),
),
]
)

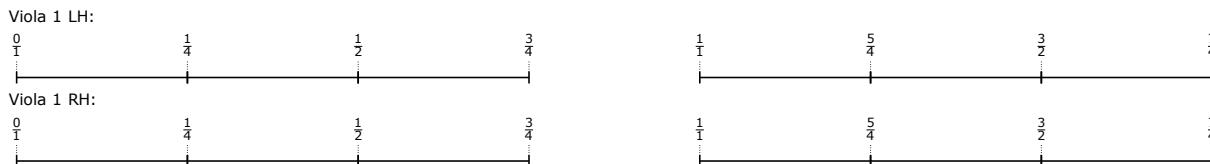
```

Consort provides an additional variation of music specifier called a *composite music specifier*. Composite music specifiers allow for the definition of music for voices in tandem, such as the fingering- and bowing-voice in a split-hands string notation. When passed as part of a voice-name-to-music-specifier mapping to a timespan-maker, that timespan-maker will create timespans for the first voice of the composite music specifier and then create timespans for the second voice as though it had its own dependent timespan-maker solely for those timespans from the first voice. This behavior ensures a degree of synchronization between pairs of voices which should always appear at the same time in a score:

```

>>> composite_musicSpecifier = consort.CompositeMusicSpecifier(
...     primary_musicSpecifier='one',
...     primary_voiceName='Viola 1 RH',
...     rotationIndices=(0, 1, -1),
...     secondary_voiceName='Viola 1 LH',
...     secondary_musicSpecifier=consort.MusicSpecifierSequence(
...         applicationRate='phrase',
...         musicSpecifiers=['two', 'three', 'four'],
...     ),
... )
>>> musicSpecifiers = {'Viola 1': composite_musicSpecifier}
>>> timespanInventory = timespanMaker(
...     musicSpecifiers=musicSpecifiers,
...     targetTimespan=targetTimespan,
... )
>>> show(timespanInventory, key='voiceName')

```



## 4.2 INTERPRETATION

At any point during specification, a segment-maker may be interpreted to produce an illustration. Score interpretation proceeds conceptually much like compilation in classical computing, where a compiler parses an instruction set written in some source language into an intermediate representation and then transforms that same intermediate representation into instructions executable on a target platform. In Consort's interpretation stage, the compiler is the segment-maker itself, and the source instruction set its configuration – its tempo, permitted meters, music settings and so forth. Timespan inventories produced by each music setting's timespan-maker, populated with timespans annotated with music specifiers perform the role of the intermediate representation. This intermediate representation acts as a *maquette*, blocking out where in the resulting score segment various materials should be deployed. The target of score interpretation is, unsurprisingly, a fully-fledged score aggregated from Abjad score components. Interpretation takes place in two broad stages – rhythmic interpretation, followed by non-rhythmic interpretation – with the first stage producing a score populated solely with rhythmic information, and the second stage applying grace notes, pitches, indicators, spanners and various typographic overrides to the previously-constructed rhythmic skeleton.

## 4.3 RHYTHMIC INTERPRETATION

Broadly speaking, rhythmic interpretation proceeds from coarse- to fine-grained. The segment-maker creates a *maquette* – a model of the locations of musical materials in the score – by calling each of its music-settings in turn to populate a timespan inventory. It then resolves overlap conflicts within that inventory, fits meters against the resolved inventory's offsets, splits and prunes the contents of the inventory according to its fitted metrical structure, and finally converts the finished timespan maquette into an actual score. This process, like interpretation overall, can be roughly divided into work flows of *maquette creation* and *music creation*, although in practice the two flows are interleaved significantly as they actually influence one another. When creating the maquette, music settings with *independent* timespan-makers – those which do not depend on the contents of a previously created timespan inventory, specifically flooded and talea timespan-makers – are called in a first pass, and those with dependent timespan-makers in a second. These two passes only differ significantly in that meters are fitted against the segment's timespan maquette during the independent timespan-maker pass, but not during the dependent.

### 4.3.1 POPULATING THE MAQUETTE

To populate the maquette, the segment-maker calls each of its music settings to produce timespans according to their configured timespan-makers, *timespan-identifiers* – optional specifications of which portion of the segment’s overall timespan to operate within – and voice-associated music specifiers. Timespan identifiers may include timespans, inventories of timespans, or even expressions callable against the segment-makers own timespan which evaluate to an inventory of timespans.

Music settings exist without any reference to a segment-maker, its desired duration – and therefore desired timespan –, or its score template. In order to know which target timespan or timespans a music setting’s timespan-maker should operate within – in the case of procedural timespan identifiers such as *ratio-parts expressions* which must be called against a preexisting timespan in order to determine what part or parts of that timespan to use – the music setting must resolve its timespan identifier against the segment’s desired duration. Target timespan resolution must also take into account offset quantization, as the target timespans resulting from the evaluation of a ratio-parts expression may not align against a power-of-two-denominator offset grid such as  $\frac{1}{8}$ ,  $\frac{1}{16}$  or  $\frac{1}{32}$ . Because timespan-makers produce their output relative to the start-offset of their target timespan, a misaligned target timespan – starting at an offset like  $\frac{1}{3}$  or  $\frac{15}{7}$  rather than  $\frac{1}{4}$  or 0 – will cause all generated timespans to be misaligned.

Music settings associate their music specifiers with strings containing voice-name abbreviations. These abbreviations are always underscore-delimited strings such as `violin_1` or `piano_1h` – necessitated by Python’s keyword argument syntax so that they can be used as keys during class instantiation – which represent voices in a score, without having established a concrete reference to those voice contexts. In order to match its music specifiers against actual voice contexts in a score, the music setting must resolve its voice-name abbreviations against a score template, looking up each abbreviation on the template and returning the real name of the associated context. This lookup process allows music settings to construct well-formed voice-name-to-music-specifier mappings, implemented as ordered dictionaries and ordered by the actual *score index* – effectively, the vertical location – of each looked-up context in the segment-maker’s under-construction score. As demonstrated in section 3.4, timespan-makers require these mappings to produce their output. Additionally, voice-name resolution guarantees that the values in the resolved voice-name-to-music-specifier mapping are always either a `MusicSpecifierSequence` or `CompositeMusicSpecifier` instance via coercion, where any composite music-specifier’s primary and secondary music specifiers are themselves coerced into music specifier sequences. This coercion ensures that all arguments to the music setting’s

timespan-maker are in a well-formed and predictable state.<sup>1</sup>

Consider the following string quartet score template, which will be used with the above music setting. Note the names of the various contexts defined in it, made visible when formatted as LilyPond syntax, where each context name is given by a quoted string on the lines beginning with \context. The score contains a time signature context, and four staff groups, one for each instrument in the quartet. These staff groups then contain two staves, for the left and right hands of each performer, with each staff containing a single voice:

```
>>> score_template = consort.StringQuartetScoreTemplate()
>>> score = score_template()
>>> print(format(score))
\context Score = "String Quartet Score" <<
  \tag #'time
  \context TimeSignatureContext = "Time Signature Context" {
  }
  \tag #'violin-1
  \context StringPerformerGroup = "Violin 1 Performer Group" \with {
    instrumentName = \markup {
      \hcenter-in
      #10
      "Violin 1"
    }
    shortInstrumentName = \markup {
      \hcenter-in
      #10
      "Vln. 1"
    }
  } <<
  \context BowingStaff = "Violin 1 Bowing Staff" {
    \clef "percussion"
    \context Voice = "Violin 1 Bowing Voice" {
    }
  }
  \context FingeringStaff = "Violin 1 Fingering Staff" {
    \clef "treble"
    \context Voice = "Violin 1 Fingering Voice" {
    }
  }
>>
\tag #'violin-2
\context StringPerformerGroup = "Violin 2 Performer Group" \with {
  instrumentName = \markup {
    \hcenter-in
  }
```

---

<sup>1</sup>Timespan-makers actually delegate the creation of performed and silent timespans to music specifier sequences. While not demonstrated explicitly in section 3.4, this delegation allows timespan-makers to use both music specifier sequences and composite music specifiers interchangeably, with the former creating timespans associated with one voice and the latter with two. When the values of a timespan-maker's input voice-name-to-music-specifier mapping are neither music specifier sequences nor composite music specifiers – as was the case in all of the examples in section 3.4 – they implicitly coerce those values into music specifier sequences.

```

        #10
        "Violin 2"
    }
shortInstrumentName = \markup {
    \hcenter-in
    #10
    "Vln. 2"
}
} <<
\context BowingStaff = "Violin 2 Bowing Staff" {
    \clef "percussion"
\context Voice = "Violin 2 Bowing Voice" {
}
}
\context FingeringStaff = "Violin 2 Fingering Staff" {
    \clef "treble"
\context Voice = "Violin 2 Fingering Voice" {
}
}
>>
\tag #'viola
\context StringPerformerGroup = "Viola Performer Group" \with {
instrumentName = \markup {
    \hcenter-in
    #10
    Viola
}
shortInstrumentName = \markup {
    \hcenter-in
    #10
    Va.
}
} <<
\context BowingStaff = "Viola Bowing Staff" {
    \clef "percussion"
\context Voice = "Viola Bowing Voice" {
}
}
\context FingeringStaff = "Viola Fingering Staff" {
    \clef "alto"
\context Voice = "Viola Fingering Voice" {
}
}
>>
\tag #'cello
\context StringPerformerGroup = "Cello Performer Group" \with {
instrumentName = \markup {
    \hcenter-in
    #10
    Cello
}
shortInstrumentName = \markup {
    \hcenter-in
    #10
}

```

```

        Vc.
    }
} <<
\context BowingStaff = "Cello Bowing Staff" {
    \clef "percussion"
    \context Voice = "Cello Bowing Voice" {
    }
}
\context FingeringStaff = "Cello Fingering Staff" {
    \clef "bass"
    \context Voice = "Cello Fingering Voice" {
    }
}
>>
>>

```

Consort's score templates provide abbreviation-to-voice-name mappings via their `context_name_abbreviations` property. Likewise, they provide mappings for use with composite music specifiers which map “parent” context abbreviations to their relevant child abbreviation pairs:

```

>>> for abbr, context_name in score_template.context_name_abbreviations.items():
...     print(abbr, context_name)
...
('viola_rh', 'Viola Bowing Voice')
('cello_rh', 'Cello Bowing Voice')
('violin_1_lh', 'Violin 1 Fingering Voice')
('violin_2_lh', 'Violin 2 Fingering Voice')
('violin_2_rh', 'Violin 2 Bowing Voice')
('cello', 'Cello Performer Group')
('viola', 'Viola Performer Group')
('violin_1_rh', 'Violin 1 Bowing Voice')
('viola_lh', 'Viola Fingering Voice')
('violin_2', 'Violin 2 Performer Group')
('cello_lh', 'Cello Fingering Voice')
('violin_1', 'Violin 1 Performer Group')

>>> for parent, child_pair in score_template.composite_context_pairs.items():
...     print(parent, child_pair)
...
('violin_2', ('violin_2_rh', 'violin_2_lh'))
('cello', ('cello_rh', 'cello_lh'))
('viola', ('viola_rh', 'viola_lh'))
('violin_1', ('violin_1_rh', 'violin_1_lh'))

```

Consider the following music setting, which includes a timespan-maker definition, a ratio-parts expression as its timespan identifier, and three music specifiers, one of which is a composite music specifier. String values for the music specifiers are used simply for pedagogical reasons:

```

>>> music_setting = consort.MusicSetting(
...     timespan_identifier=consort.RatioPartsExpression(
...         parts=(0, 2),
...         ratio=(1, 3, 2),
...     ),
...     timespan_maker=consort.FloodedTimespanMaker(),
...     violin_2_lh='A',
...     viola_lh=('B', 'C', 'D'),
...     cello=consort.CompositeMusicSpecifier(
...         primary_music_specifier='one',
...         secondary_music_specifier=consort.MusicSpecifierSequence(
...             music_specifiers=['two', 'three', 'four'],
...         ),
...     )
... )
...

```

Resolving the above music specifier against Consort's string quartet score template dereferences the correct context names, allowing them to be indexed into any score created by that score template. Note that not only are the two non-composite music specifiers now associated with voices in the score, but they have been recreated as music specifier sequences. Likewise, the composite music specifier has been reconfigured such that it knows the specific names of the two voices associated with its primary and secondary music specifiers, themselves recreated as music specifier sequences:

```

>>> result = music_setting.resolve_music_specifiers(score_template)
>>> for context_name, resolved_music_specifier in result.items():
...     print('CONTEXT:', context_name)
...     print(format(resolved_music_specifier))
...
('CONTEXT:', 'Violin 2 Fingering Voice')
consort.tools.MusicSpecifierSequence(
    music_specifiers=datastructuretools.CyclicTuple(
        ['A']
    ),
)
('CONTEXT:', 'Viola Fingering Voice')
consort.tools.MusicSpecifierSequence(
    music_specifiers=datastructuretools.CyclicTuple(
        ['B', 'C', 'D']
    ),
)
('CONTEXT:', 'Cello Performer Group')
consort.tools.CompositeMusicSpecifier(
    primary_music_specifier=consort.tools.MusicSpecifierSequence(
        music_specifiers=datastructuretools.CyclicTuple(
            ['one']
        ),
    ),
    primary_voice_name='Cello Bowing Voice',
)

```

```

secondary_music_specifier=consort.tools.MusicSpecifierSequence(
    music_specifiers=datastructuretools.CyclicTuple(
        ['two', 'three', 'four']
    ),
),
secondary_voice_name='Cello Fingering Voice',
)

```

Resolving a music setting's timespan identifier against the segment-maker's segment timespan results in one or more target timespans which can be used as argument to the music setting's timespan-maker. We can simulate resolution against an actual timespan-maker's timespan by simply creating a "segment timespan" from scratch:

```

>>> segment_timespan = timespantools.Timespan(0, 8)
>>> show(segment_timespan)

```



```

>>> target_timespans = music_setting.resolve_target_timespans(segment_timespan)
>>> show(target_timespans, range_=(0, 8))

```



Note that the start and stop offsets of the target timespans resolved above do not all align at offsets with power-of-two denominators, such as  $\frac{1}{2}$ ,  $\frac{1}{4}$  or  $\frac{1}{8}$ . By specifying a timespan quantization, the target timespans generated during resolution can be quantized to a grid:

```

>>> target_timespans = music_setting.resolve_target_timespans(
...     segment_timespan,
...     timespan_quantization=Duration(1, 16),
... )
>>> show(target_timespans, range_=(0, 8))

```



Additionally, a subtractive mask can be applied to the target timespans:

```

>>> music_setting = new(
...     music_setting,
...     timespan_identifier__mask_timespan=timespantools.Timespan(
...         start_offset=(1, 2),
...         stop_offset=7,
...     ),
... )
>>> target_timespans = music_setting.resolve_target_timespans(segment_timespan)
>>> show(target_timespans, range_=(0, 8))

```



Once resolved, each music setting can call its timespan-maker to create timespans with the appropriate voice-name-to-music-specifier mapping, target timespans and layer, adding the contents of the resulting inventory to the growing maquette of performed and silent timespans produced by previous music settings. The populating process repeats until no more music settings remain.

#### 4.3.2 RESOLVING CASCADING OVERLAP

One of the driving motivations behind Consort is the ability to create musical textures consisting of multiple overlapping layers, each created by an independent maker and each with different materials from the other layers, allowing multiple materials of various provenances to appear in the same instrumental voice. Still, because acoustic instruments cannot simply “create” arbitrary numbers of voices like a synthesizer might, any overlap in material allocated for a given voice needs to be resolved. Consort handles resolution of overlap via tournament, choosing only one material from a collection of overlapping candidates.

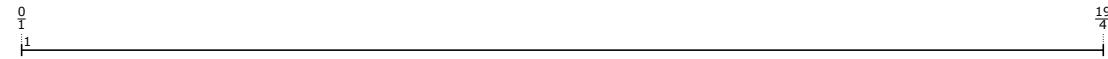
Consider the following three timespan inventories, created with three different timespan-makers and music-specifier mappings. The first inventory is created via a flooded timespan-maker, filling the entirety of a target timespan of  $0\frac{1}{1}$  to  $1\frac{9}{4}$  in voices “Voice 2” and “Voice 3”. This inventory behaves like a constant “background layer” for those two voices:

```
>>> layer_1_timespan_maker = consort.FloodedTimespanMaker()
>>> layer_1_target_timespan = timespantools.Timespan(0, (19, 4))
>>> layer_1_music_specifiers = collections.OrderedDict([
...     ('Voice 2', None),
...     ('Voice 3', None),
... ])
>>> layer_1 = layer_1_timespan_maker(
...     layer=1,
...     music_specifiers=layer_1_music_specifiers,
...     target_timespan=layer_1_target_timespan,
... )
>>> show(layer_1, key='voice_name', range_=(0, (21, 4)))
```

Voice 2:



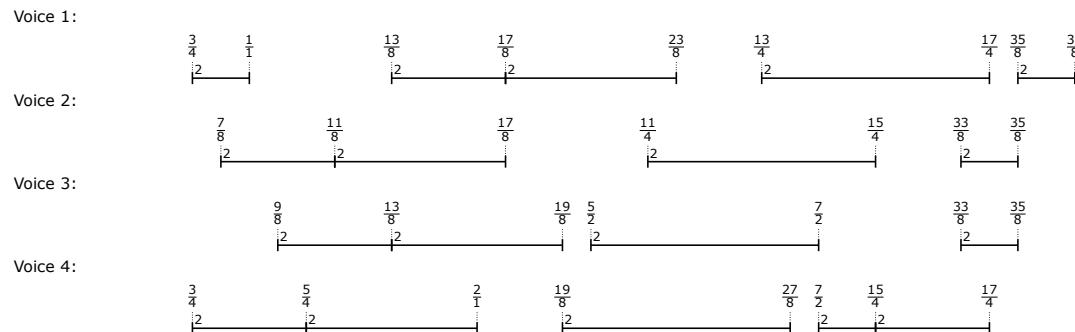
Voice 3:



The second timespan inventory is created by a talea timespan-maker. This inventory covers all four voices – “Voice 1”, “Voice 2”, “Voice 3” and “Voice 4” – with a texture of evenly distributed phrases and silences. However, un-

like the first timespan inventory, this texture only spans the target timespan of  $\frac{3}{4}$  to  $\frac{19}{4}$ , guaranteeing that the background layer for “Voice 2” and “Voice 3” is untouched in the span of  $\frac{0}{1}$  to  $\frac{3}{4}$ :

```
>>> layer_2_timespan_maker = consort.TaleaTimespanMaker(
...     initial_silence_talea=rhythmmakertools.Talea(
...         counts=(0, 1, 3),
...         denominator=8,
...     ),
...     playing_groupings=(1, 2),
...     playing_talea=rhythmmakertools.Talea(
...         counts=(1, 2, 3, 4),
...         denominator=4,
...     ),
...     silence_talea=rhythmmakertools.Talea(
...         counts=(5, 3, 1),
...         denominator=8,
...     ),
... )
>>> layer_2_target_timespan = timespantools.Timespan((3, 4), (19, 4))
>>> layer_2_music_specifiers = collections.OrderedDict([
...     ('Voice 1', None),
...     ('Voice 2', None),
...     ('Voice 3', None),
...     ('Voice 4', None),
... ])
>>> layer_2 = layer_2_timespan_maker(
...     layer=2,
...     music_specifiers=layer_2_music_specifiers,
...     target_timespan=layer_2_target_timespan,
... )
>>> show(layer_2, key='voice_name', range_=(0, (21, 4)))
```



The third timespan inventory consists of groups of near-simultaneous attacks in three voices – “Voice 1”, “Voice 3” and “Voice 4”. This inventory’s talea timespan-maker has padded  $\frac{1}{4}$ -duration silences around the beginning and end of each group, guaranteeing that any performed timespans with layers lower than 3 will be masked not only by this layer’s performed timespans, but by its silent timespans as well. Additionally, the third timespan inventory was created with a target timespan of  $\frac{6}{4}$  to  $\frac{21}{4}$ . While – due to the complexities of the talea timespan-maker’s patterns

- the generated timespan texture may not extend all of the way to its target timespan’s stop offset at  $2\frac{1}{4}$ , it will certainly not contain any performed or silent timespans earlier than  $\frac{5}{4}$  – the target timespan’s start offset minus the timespan-maker’s padding –, leaving lower layers untouched from  $\frac{0}{1}$  to  $\frac{5}{4}$ :

```

>>> layer_3_timespan_maker = consort.TaleaTimespanMaker(
...     initial_silence_talea=rhythmmakertools.Talea(
...         counts=(0, 0, 0, 1),
...         denominator=8,
...         ),
...     padding=Duration(1, 4),
...     playing_talea=rhythmmakertools.Talea(
...         counts=(2, 3, 4),
...         denominator=8,
...         ),
...     silence_talea=rhythmmakertools.Talea(
...         counts=(6,),
...         denominator=4,
...         ),
...     synchronize_step=True,
... )
>>> layer_3_target_timespan = timespanools.Timespan((6, 4), (21, 4))
>>> layer_3_music_specifiers = collections.OrderedDict([
...     ('Voice 1', None),
...     ('Voice 3', None),
...     ('Voice 4', None),
... ])
>>> layer_3 = layer_3_timespan_maker(
...     layer=3,
...     music_specifiers=layer_3_music_specifiers,
...     target_timespan=layer_3_target_timespan,
... )
>>> show(layer_3, key='voice_name', range_=(0, (21, 4)))

```

Voice 1:



Voice 3:



Voice 4:



Recall from section 3.4 that timespan-makers can modify a timespan inventory in-place when called, rather than generating a new one from scratch. The following code – greatly simplified from Consort’s `SegmentMaker.populate_multiplexed_maquette()` method – demonstrates the process of calling multiple timespan-makers with their corresponding target timespans and voice-name-to-music-specifier mappings to progressively populate a single timespan inventory in-place. Note the use of two Python builtin iterators `zip()` and `enumerate()`. The `zip()` iterators iterates over the iterables with which it was instantiated, yielding the first item of each of its iterables as a tuple, then the

second of each of its iterables, then the third, and so forth. The `enumerate()` iterator yields each item of its input iterable paired with that item's index, filling the role in Python for the verbose *for loop* loop idiom found in many C-like languages, such as Java or Javascript: `for(int x = 10; x < 20; x = x + 1) { ... }.`

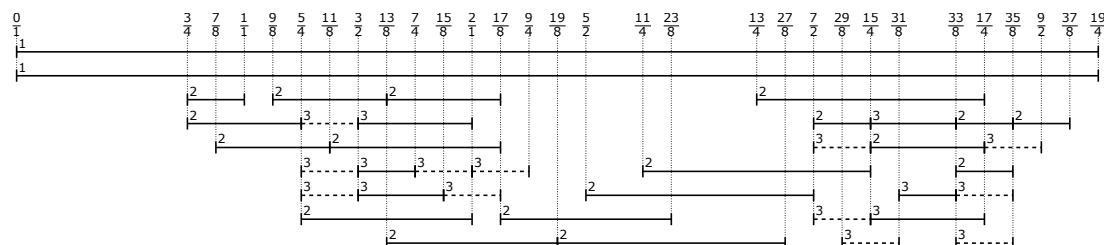
```

>>> timespan_makers = (
...     layer_1_timespan_maker,
...     layer_2_timespan_maker,
...     layer_3_timespan_maker,
... )
>>> music_specifiers = (
...     layer_1_music_specifiers,
...     layer_2_music_specifiers,
...     layer_3_music_specifiers,
... )
>>> target_time spans = (
...     layer_1_target_time span,
...     layer_2_target_time span,
...     layer_3_target_time span,
... )
>>> triples = zip(timespan_makers, music_specifiers, target_time spans)
>>> timespan_inventory = timespan tools.TimespanInventory()
>>> for layer, triple in enumerate(triples, 1):
...     timespan_inventory = triple[0](
...         layer=layer,
...         music_specifiers=triple[1],
...         target_time span=triple[2],
...         timespan_inventory=timespan_inventory,
...     )
...
...

```

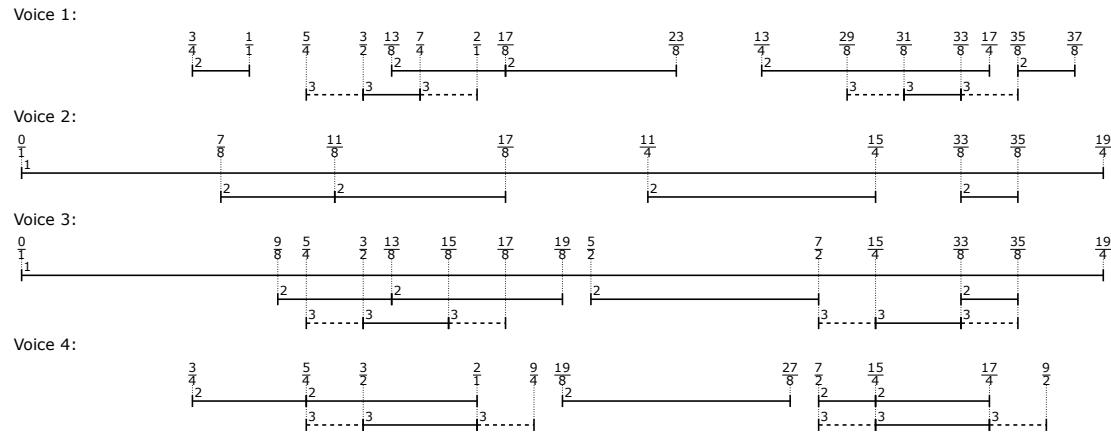
Recall that the result of this process is still a timespan inventory, just as described in section 3.2. While this document has taken pains to clarify the internal structure of timespan-maker-generated timespan inventories by sorting and displaying voice-names in their illustrations – just as in the above three timespan inventory illustrations – that behavior derives from the timespan inventory's illustration protocol implementation, and does not reflect their actual, flat structure:

```
>>> show(timespan_inventory, range_=(0, (21, 4)))
```



Just because timespan inventories are often displayed with voice-names, and contain performed timespans with voice-name attributes, does not mean that they are automatically or internally structured that way. Such sorting requires an additional pass – demultiplexing. Visualizing the inventory, exploded by voice-name, then sorted by layer, shows the overlap in each voice:

```
>>> show(timespan_inventory, key='voice_name', sortkey='layer', range_=(0, (21, 4)))
```



In order to resolve cascading overlap, the segment-maker must first demultiplex the performed and silent timespans in the still-multiplexed maquette into separate timespan inventories by their voice-name attributes. The segment-maker further separates each demultiplexed-by-voice-name timespan inventory into multiple timespan inventories according to their contents' layer attributes, with the lowest-layered inventory first, and the highest-layered inventory last. This results in one inventory per-voice, per-timespan-maker from the maquette population process.<sup>2</sup> With the maquette fully demultiplexed, the segment-maker can proceed through each layer in each voice, from lowest to highest. It progressively subtracts the timespans in each higher inventory from the lowest inventory, effectively cutting out holes outlining the shapes of that higher inventory's timespans. The segment-maker then adds that higher inventory's timespans into the lowest-layered inventory. This process masks lower-layered timespans with higher ones. The process repeats until no more timespan inventories remain for that voice, then moves onto the inventories for the next voice. The resolved, demultiplexed results are finally collected into a timespan inventory mapping, associating voice names with resolved timespan inventories, and returned.

---

<sup>2</sup>Why not keep all timespan-maker output separated from the very beginning? Working with a single multiplexed timespan inventory for much of the rhythmic interpretation process simplifies many of the procedures used therein, such as dependent timespan-maker evaluation, splitting, consolidation, etc. Compared to many of the later rhythmic operations, such as meter rewriting, multiplexing and demultiplexing timespan inventories are computationally trivial.

```

>>> demultiplexed_maquette = consort.SegmentMaker.resolve_maquette(
...     timespan_inventory,
... )

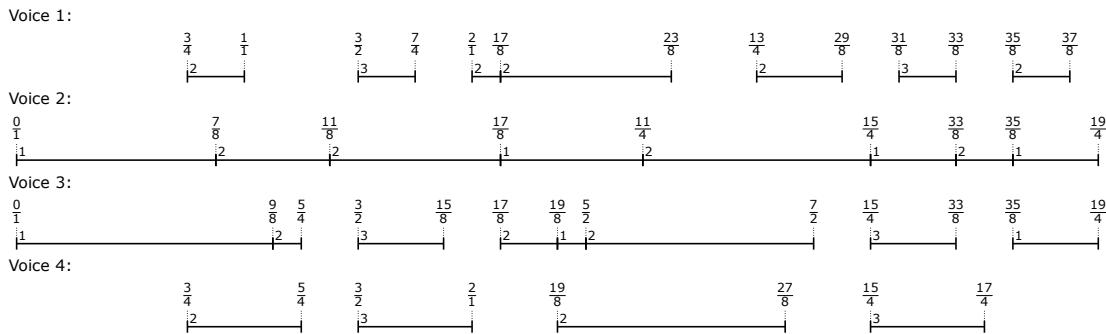
```

After resolution, no overlap remains in the timespans for any voice. Note too that no silent timespans – like those created as padding in the third timespan-maker above – remain either. Silent timespans act solely as a means of “holding space” for a layer, masking but not replacing timespans in lower layers:

```

>>> show(demultiplexed_maquette, range_=(0, (21, 4)))

```



Unlike many of the timespan-handling functions demonstrated in this chapter as well as in chapter 3, `resolve_maquette()` returns a `TimespanInventoryMapping` rather than a `TimespanInventory`. The timespan inventory mapping already explicitly uses voice-names as keys, obviating the need for a `key='voice_name'` keyword argument pair in the call to `show()`.

### 4.3.3 FINDING METERS, REVISITED

Consort’s segment-maker implements a variation on the meter-fitting algorithm described in section 3.8. Each segment-maker may be configured with an inventory of permitted meters, as well as maximum meter run length, in order to drive the meter fitting algorithm. When counting offsets, segment-makers include the offsets found on the performed timespans in their maquette but discard those from silent timespans, removing any influence from timespans created solely for silencing other timespans. The start offset of each performed timespan is weighed twice as much as their stop offset. This imbalance helps emphasize simultaneous phrase starts across different voices. Additionally, segment-maker’s weight their own desired stop offset at a much higher value than any count derived from the offsets in their maquette. This attempts to influence the meter fitting process into selecting a series of meters which end as close to their desired stop offset as possible. After fitting meters, the segment-maker caches both the fitted meters and their boundaries as properties on its instance, affording later retrieval by other subroutines.

#### 4.3.4 SPLITTING, PRUNING & CONSOLIDATION

Once meters have been fitted against the resolved maquette, the timespans in the maquette must be split at the measure boundaries outlined by those meters. Splitting guarantees that no timespans cross any bar-lines and that therefore no containers generated by those timespans when notating them as score components cross any bar-lines either. While LilyPond can typeset bar-line crossing notes, chords and even tuplets, the scores I have composed via Consort do not currently make use of such constructions. As described in subsection 3.1.3, operations on timespans which change offsets – generating new timespans in the process, rather than modifying the operated-upon timespan in-place – preserve their unmodified properties via templating. Splitting is no exception, and those timespans split maintain their music specifiers, layer identifiers and voice-names:

```
>>> performed_timespan = consort.PerformedTimespan(
...     layer=3,
...     start_offset=(1, 2),
...     stop_offset=(13, 8),
...     voice_name='Percussion Voice',
... )
>>> shards = performed_timespan.split_at_offset((9, 16))
>>> print(format(shards))
timespantools.TimespanInventory(
    [
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(1, 2),
            stop_offset=durationtools.Offset(9, 16),
            layer=3,
            original_stop_offset=durationtools.Offset(13, 8),
            voice_name='Percussion Voice',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(9, 16),
            stop_offset=durationtools.Offset(13, 8),
            layer=3,
            original_start_offset=durationtools.Offset(1, 2),
            voice_name='Percussion Voice',
        ),
    ]
)
```

After splitting, the segment-maker prunes timespans considered either too short or malformed. Performed timespans may be configured with a `minimum_duration` property. Timespan-makers may set this property on timespans they create when they are themselves configured with a `TimespanSpecifier`. Any performed timespan whose actual duration is less than its minimum duration – if it has been configured with a minimum duration – will be removed from the maquette. Likewise any timespan with a duration of 0 – therefore malformed – will also be removed.

While the latter pruning guarantees correctness of the maquette – malformed timespans cannot be rendered as notation at all, and may cause other problems when partitioning due to ambiguities in their start / stop offset semantics –, the former allows for a kind of compositional control over the maquette. When notated with certain rhythm-makers, overly short divisions – especially those shorter than  $\frac{1}{8}$ -duration – may give undesirable results. Note that silent timespans have no configurable minimum duration. Their `minimum_duration` always returns 0. They maintain this dummy property so that the segment-maker’s timespan-pruning algorithms can treat silent and performed timespans identically.

Next, Consort’s segment-maker *consolidates* contiguous performed timespans with identical music specifiers, caching the durations of the consolidated timespans in a new timespan’s `divisions` property. Each new consolidated timespan outlines the start and stop offset of its consolidated group:

```
>>> timespans = timespantools.TimespanInventory([
...     consort.PerformedTimespan(
...         start_offset=0,
...         stop_offset=10,
...         musicSpecifier='foo',
...     ),
...     consort.PerformedTimespan(
...         start_offset=10,
...         stop_offset=20,
...         musicSpecifier='foo',
...     ),
...     consort.PerformedTimespan(
...         start_offset=20,
...         stop_offset=25,
...         musicSpecifier='bar',
...     ),
...     consort.PerformedTimespan(
...         start_offset=40,
...         stop_offset=50,
...         musicSpecifier='bar',
...     ),
...     consort.PerformedTimespan(
...         start_offset=50,
...         stop_offset=58,
...         musicSpecifier='bar',
...     ),
... ],
... )
>>> show(timespans)
```



```
>>> consolidated_timespans = consort.SegmentMaker.consolidate_timespans(timespans)
>>> show(consolidated_timespans)
```



Consolidation transforms performed timespans from free-floating cells in the maquette into components of larger phrases. The cached divisions also prepare these consolidated timespans for *inscription* by defining the correct input for a rhythm-maker: a sequence of divisions.

```
>>> print(format(consolidated_timespans))
timespanTools.TimespanInventory(
    [
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(20, 1),
            divisions=(
                durationtools.Duration(10, 1),
                durationtools.Duration(10, 1),
            ),
            musicSpecifier='foo',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(20, 1),
            stop_offset=durationtools.Offset(25, 1),
            divisions=(
                durationtools.Duration(5, 1),
            ),
            musicSpecifier='bar',
        ),
        consort.tools.PerformedTimespan(
            start_offset=durationtools.Offset(40, 1),
            stop_offset=durationtools.Offset(58, 1),
            divisions=(
                durationtools.Duration(10, 1),
                durationtools.Duration(8, 1),
            ),
            musicSpecifier='bar',
        ),
    ],
)
```

If the music specifier of the consolidated timespan was configured with a *minimum phrase duration*, and the consolidated timespan falls under that threshold, it too is discarded.

#### 4.3.5 INSCRIPTION

*Inscription* describes the process of generating *music* from a performed timespan's divisions and rhythm-maker and *inscribing* the timespan with the result. Consort's segment-maker performs inscription by iterating over the timespans for each voice in the demultiplexed, consolidated maquette, in score order. For each performed timespan encountered, the segment-maker retrieves that performed timespan's music specifier, and increments that music

specifiers count in a counter. This allows each music specifier to maintain a seed value while inscribing each performed timespan, even in fragmentary textures, and to produce continuously varied results from each successive rhythm-maker belonging to the same music specifier. Recall from subsection 3.9.3 that rhythm-makers can be called not only with a list of divisions, but also a seed value, rotating the rhythm-makers sequence-like properties when creating its rhythmic output. All of the handlers discussed in section 4.4 employ similar – or even more complex – techniques for maintaining state across different phrases sharing the same music specifier. The segment-maker also retrieves the performed timespan’s division list – created during consolidation, as described in subsection 4.3.4 – and its rhythm-maker. Rhythm-maker retrieval, like seed retrieval, is non-trivial. A performed timespan’s music specifier may not have rhythm-maker defined, or that performed timespan may not even have a music specifier defined. If a performed timespan *has* a rhythm-maker defined on its music specifier, the segment-maker retrieves that. If the timespan has a music specifier, but no defined rhythm-maker, the segment-maker constructs a note rhythm-maker which ties all of its divisions together. If the timespan has no music specifier defined at all, the segment-maker returns a fully-masked note rhythm-maker.

With the performed timespan’s seed, rhythm-maker and division list ready, the segment-maker creates the performed timespan’s music. This proceeds almost identically to the `make_music()` function described in subsection 3.9.3. Consort’s segment-maker makes one additional adjustment on top of that algorithm, replacing trivially-prolated tuplets – tuplets with ratios of 1:1 – with unprolated containers. Next, the segment-maker performs *rest consolidation* on the newly-generated phrase, grouping all of the phrase’s unprolated rests – those not appearing in tuplets – into their own containers, and leaving all other components – all notes and chords, and any rests found within a tuplet – in their original division within the phrase. Rest consolidation allows the segment-maker to not only regroup the contents of a phrase into silent and non-silent segments, but to actually split the performed timespan itself, creating larger gaps within the maquette, and improving the chances for notating full-bar rests when finally filling in silences between phrases.

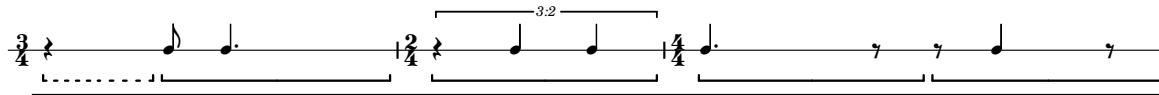
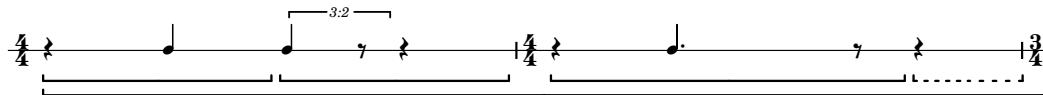
Consider the following phrase-like container – annotated to show its internal division structure –, containing divisions in various configurations – with rests at the beginning, at the end, with prolated rests, no rests at all, and so forth:

```
>>> parseable = r'''
... \new Voice {
...   {
...     { \time 4/4 r4 c4 }
...     { \times 2/3 { c4 r8 } r4 }
```

```

...
{ \time 4/4 r4 c4. r8 }
...
{ r4 \break }
...
{ \time 3/4 r4 }
...
{ c8 c4. }
...
\time 2/3 { \time 2/4 r4 c4 c4 }
...
{ \time 4/4 c4. r8 }
...
{ r8 c4 r8 }
...
}
...
...
>>> unconsolidated_staff = Staff(parseable, context_name='RhythmicStaff')
>>> consort.annotate(unconsolidated_staff)
>>> show(unconsolidated_staff)

```

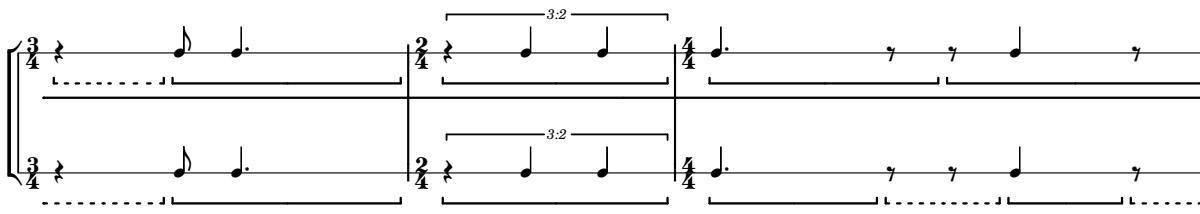
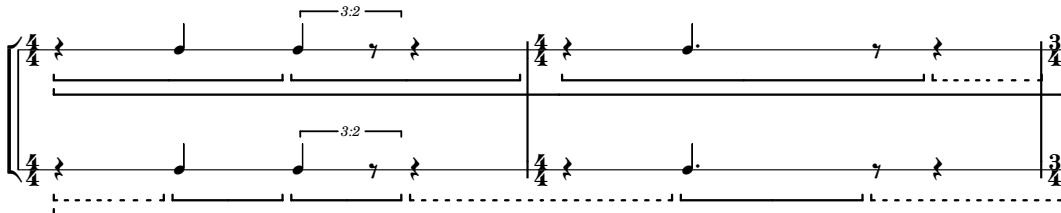


A clear comparison can be made by duplicating the original phrase, consolidating its rests, annotating it, and grouping both the original unconsolidated phrase and the consolidated into a staff group, with the original above and the altered below:

```

>>> consolidated_staff = Staff(parseable, context_name='RhythmicStaff')
>>> for voice in consolidated_staff:
...     for phrase in voice:
...         phrase = consort.SegmentMaker.consolidate_rests(phrase)
...
>>> consort.annotate(consolidated_staff)
>>> staff_group = StaffGroup([unconsolidated_staff, consolidated_staff])
>>> show(staff_group)

```



Note above how after rest consolidation the rests in the lower staff have been grouped together into their own divisions – annotated with dashed lines – while all other components – including rests within tuplets – remain in their original divisions. Now consider the following simple rhythm-maker which, when passed a division sequence of seven  $\frac{1}{4}$ -durations, generates a rhythm consisting of two groups of  $\frac{1}{4}$  notes delimited by rests:

```
>>> rhythm_maker=rhythmmakertools.NoteRhythmMaker(
...     output_masks=[
...         rhythmmakertools.BooleanPattern(
...             indices=[0],
...             period=3,
...         ),
...     ],
... )
>>> divisions = [Duration(1, 4)] * 7
>>> show(rhythm_maker, divisions=divisions)
```

$\frac{1}{4}$



The above rhythm-maker can be treated as the rhythm-maker for a music specifier annotating a performed timespan. Likewise, the divisions used above can be used as the division sequence for this same performed timespan. If this timespan's rhythm-maker would be called with its division list, the same rhythm as above would result:

```
>>> timespan = consort.PerformedTimespan(
...     divisions=divisions,
...     start_offset=0,
...     stop_offset=(7, 4),
...     musicSpecifier=consort.MusicSpecifier(
...         rhythm_maker=rhythm_maker,
...     ),
... )
>>> show(timespan)
```



The timespan can then be inscribed by calling the segment-maker's `inscribe_timespan()` method:

```
>>> inscribed_timespans = consort.SegmentMaker.inscribe_timespan(timespan)
>>> show(inscribed_timespans, range_=(0, (7, 4)))
```



```
>>> print(format(inscribed_timespans))
timespantools.TimespanInventory()
```

```

        [
            consort.tools.PerformedTimespan(
                start_offset=durationtools.Offset(1, 4),
                stop_offset=durationtools.Offset(3, 4),
                music(scoretools.Container(
                    "{ c'4 } { c'4 }"
                )),
                musicSpecifier=consort.tools.MusicSpecifier(
                    rhythmMaker=rhythmmakertools.NoteRhythmMaker(
                        output_masks=rhythmmakertools.BooleanPatternInventory(
                            (
                                rhythmmakertools.BooleanPattern(
                                    indices=(0,),
                                    period=3,
                                ),
                            )
                        ),
                    ),
                ),
                original_start_offset=durationtools.Offset(0, 1),
                original_stop_offset=durationtools.Offset(7, 4),
            ),
            consort.tools.PerformedTimespan(
                start_offset=durationtools.Offset(1, 1),
                stop_offset=durationtools.Offset(3, 2),
                music(scoretools.Container(
                    "{ c'4 } { c'4 }"
                )),
                musicSpecifier=consort.tools.MusicSpecifier(
                    rhythmMaker=rhythmmakertools.NoteRhythmMaker(
                        output_masks=rhythmmakertools.BooleanPatternInventory(
                            (
                                rhythmmakertools.BooleanPattern(
                                    indices=(0,),
                                    period=3,
                                ),
                            )
                        ),
                    ),
                ),
                original_start_offset=durationtools.Offset(0, 1),
                original_stop_offset=durationtools.Offset(7, 4),
            ),
        ]
    )

```

After inscription – including rest consolidation –, the original performed timespan has been split into two new performed timespans, whose start and stop offsets outline only those portions of the generated rhythm which do not contain rests. Their music attributes have likewise been populated with only the non-silent portions of that rhythm. During inscription, the segment-maker replaces each performed timespan in the maquette against which

inscription was called with the result of that inscription process. Additionally, the segment-maker performs some simple post-processing on each inscribed timespan’s music, attaching both a beam spanner to the leaves of the phrase, as well as the music specifier used to specify the phrase’s music to the phrase itself:

```
>>> music = inscribed_timespans[0].music
>>> indicator = inspect_(music).get_indicator(consort.MusicSpecifier)
>>> print(format(indicator))
consort.tools.MusicSpecifier(
    rhythm_maker=rhythmmakertools.NoteRhythmMaker(
        output_masks=rhythmmakertools.BooleanPatternInventory(
            (
                rhythmmakertools.BooleanPattern(
                    indices=(0,),
                    period=3,
                ),
            )
        ),
    ),
)
```

Attaching the music specifier directly to the performed timespan’s phrase container allows each leaf in the segment-maker’s score to locate the music specifier which specified, without needing to resort to referencing the maquette.

#### 4.3.6 METER PRUNING

After the timespan pruning outlined in subsection 4.3.4, and the possibility of gaps introduced due to rest consolidation as outlined in subsection 4.3.5, the overall stop offset of the maquette – not the stop offset derived from the segment-maker’s desired duration – may have shifted earlier. Depending on the degree of shift, timespans in the maquette may no longer occur during one or more of the implicit timespans of the previously fitted meters. Segment-makers may be configured to discard these silences via their `discard_final_silence` property, progressively removing meters from the end of the list of fitted meters until one overlaps at least one performed timespan in the maquette.

#### 4.3.7 POPULATING DEPENDENT TIMESPANS

The previous few passages, from subsection 4.3.1 through subsection 4.3.6, describe the process of populating a segment-makers’s timespan maquette with the products of its *independent* music settings – those music settings whose timespan makers are independent, notably flooded and talea timespan-makers. With the maquette partially populated, those music settings with dependent timespan-makers – timespan-makers which generate timespans based on the contents of a preexisting timespan inventory – may finally be called to provide their contributions.

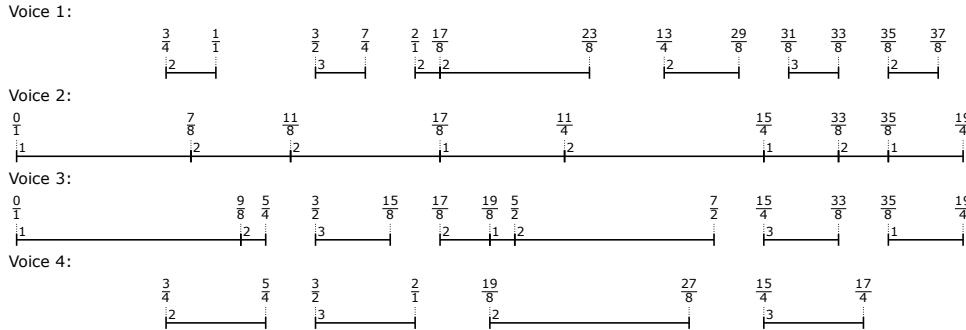
Dependent population proceeds almost identically to independent population with a few notable differences. For one, dependent population dispenses with meter finding entirely. Consort treats meter as entirely dependent upon independent timespans, as dependent timespans – in practice – are generally used for keyboard pedaling voices, and should therefore have little bearing on the overall metrical structure. And while maquette resolving, as described in subsection 4.3.2, results in a demultiplexed timespan inventory mapping – a dictionary of voice-names to timespan inventories –, independent timespan population completes by *re-multiplexing* that mapping – effectively flattening – back into a single timespan inventory. This flattening prepares the correct input for dependent timespan population, as dependent timespans require a single pre-populated timespan inventory as input. As there are no more passes of timespans to add after dependent timespan population – discounting the population of silent timespans, as described in subsection 4.3.8 – dependent timespan population completes with its maquette still demultiplexed.

#### 4.3.8 POPULATING SILENT TIMESPANS

With the maquette finally populated by inscribed performed timespans, properly split, resolved and consolidated, the segment-maker can fill in the remaining gaps between phrases in each voice. This is accomplished by creating rest-inscribed timespans for each gap. As there are no more layers to add to the maquette, the segment-maker can populate, split and inscribe timespans for each of these gaps in a single pass. For each voice in the segment-maker’s still-empty score, the segment-maker retrieves all timespans – if any – associated with that voice and subtracts each of them in turn from a single silent timespan the length of the entire segment, as determined by the first and last meter boundaries. Any remaining shards from that segment-length silent timespan represent gaps between phrases in that voice. If the maquette contains no performed timespans associated with that voice, the segment-length silent timespan remains unaltered. The segment-maker then splits the remaining silent timespan shards at every intersecting meter boundary, as described in subsection 4.3.4, guaranteeing that the resulting silent timespans do not cross bar-lines. Once split, the segment-maker partitions the silent timespans, and iterates over the partitioned groups. For each group of contiguous silent timespans, it generates a phrase of music containing only rests using a fully-masked note rhythm-maker – as described in subsection 4.3.5 –, attaches a dummy music-specifier to the phrase and instantiates a performed timespan annotated with that phrase, adding it to the current voice’s timespan inventory in the demultiplexed maquette.

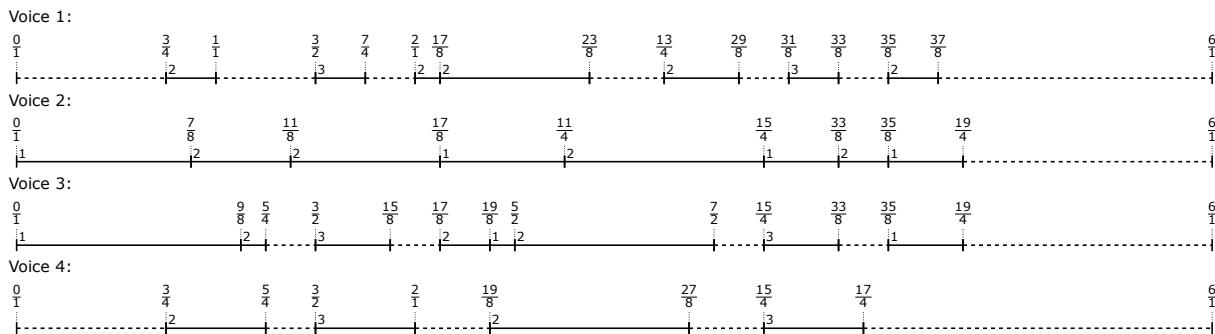
Recall the demultiplexed maquette created earlier in subsection 4.3.2:

```
| >>> show(demultiplexed_maquette, range_=(0, 6))
```



We can simulate the process of populating silent timespans in the following code, creating silent timespans – purely for visualization purposes – rather than the rest-inscribed timespans actually created during silent timespan population:

```
>>> segment_duration = Duration(24, 4)
>>> for voice_name, timespan_inventory in demultiplexed_maquette.items():
...     silent_timespans = timespantools.TimespanInventory([
...         consort.SilentTimespan(
...             start_offset=0,
...             stop_offset=segment_duration,
...             voice_name=voice_name,
...         ),
...     ])
...     for timespan in timespan_inventory:
...         silent_timespans += timespan
...     timespan_inventory.extend(silent_timespans)
...     timespan_inventory.sort()
...
...
>>> show(demultiplexed_maquette, range_=(0, 6))
```



The above assumes a segment duration of 6 instead of the maquette's initial duration of  $2\frac{1}{4}$ , and therefore pads out the end of each timespan inventory to that stop offset with silence.

#### 4.3.9 REWRITING METERS, REVISITED

Once its maquette is completely populated, Consort's segment-maker performs meter rewriting. This proceeds in a more elaborate manner than the meter rewriting process as outlined in section 3.7 and subsection 3.9.4, and

involves a number of notable differences.

For reasons of computational efficiency, Consort rewrites the meters in each phrase of music annotating each performed timespan in the maquette before those phrases have even been inserted into the segment-maker's score. Meter rewriting involves potentially many alterations to the contents of containers due to fusing and splitting, as well as many duration and offset lookups. Anytime a component is replaced or its duration changed, the cached offsets of components located in the score tree after the changed component as well as the durations of its parents are invalidated. They must be recomputed and re-cached on the next offset lookup performed on any component in the score tree. Delaying inserting each inscribed timespan's music into the segment-maker's score guarantees that that music's score depth remains shallow, and therefore limits the complexity of offset calculation during rewriting.

When beginning the meter rewriting process, the segment-maker converts its fitted meters into a timespan collection – Consort's optimized timespan inventory class – containing timespans annotated with those fitted meters, one per timespan.

```
>>> meters = metertools.MeterInventory([(3, 4), (2, 4), (6, 8), (5, 16)])
>>> meter_timespans = consort.SegmentMaker.meters_to_timespans(meters)
>>> print(format(meter_timespans))
consort.tools.TimespanCollection(
    [
        timespantools.AnnotatedTimespan(
            start_offset=durationtools.Offset(0, 1),
            stop_offset=durationtools.Offset(3, 4),
            annotation=metertools.Meter(
                '(3/4 (1/4 1/4 1/4))'
            ),
        ),
        timespantools.AnnotatedTimespan(
            start_offset=durationtools.Offset(3, 4),
            stop_offset=durationtools.Offset(5, 4),
            annotation=metertools.Meter(
                '(2/4 (1/4 1/4))'
            ),
        ),
        timespantools.AnnotatedTimespan(
            start_offset=durationtools.Offset(5, 4),
            stop_offset=durationtools.Offset(2, 1),
            annotation=metertools.Meter(
                '(6/8 ((3/8 (1/8 1/8 1/8)) (3/8 (1/8 1/8 1/8))))'
            ),
        ),
        timespantools.AnnotatedTimespan(
            start_offset=durationtools.Offset(2, 1),
            stop_offset=durationtools.Offset(37, 16),
            annotation=metertools.Meter(
                '(5/16 ((3/16 (1/16 1/16 1/16)) (2/16 (1/16 1/16))))'
            ),
        )
    ]
)
```

```

        ),
        ),
    ]
)

```

Representing meters as timespans provides two important benefits. First, meters intersecting a given division within a phrase can be efficiently located using the search methods implemented on `TimespanCollection`. Second, the time relation methods implemented on `Timespan` can be used to test if a given division's timespan is congruent – that is, possesses an identical start and stop offset – to a meter's timespan. Divisions containing solely rests which are also congruent to a meter timespan can be rewritten with full-bar rests.

The segment-maker then proceeds through each demultiplexed timespan inventory in the maquette, iterating over each timespan, then over each division in that performed timespan's music. Timespans whose rhythm-maker forbids meter rewriting – via the `forbid_meter_rewriting` flag on an optional `DurationSpellingSpecifier` – are skipped over.<sup>3</sup> In order to determine which meter governs a division, that division's timespan must be retrieved and then translated before it can be used to search the inventory of meter timespans. Because each phrase of music annotating each performed timespan has not yet been inserted into the score, they all express their start offset as 0. Likewise, each phrase's child divisions express their start offsets relative to their parent phrase's 0 start offset. Translating each division's timespan relative to the start offset of the performed timespan annotated by that division's phrase provides a useful search target for the meter timespan inventory. The translated division timespan represents the timespan that division *would* occupy if its phrase, and all other phrases, had already been inserted into the appropriate voice in the score. Intersecting meters can then be found through a simple search and retranslated relative to the current performed timespan's start offset, giving their appropriate location within the not-yet-inserted phrase. The following operations outline the translation and search process:

```

>>> inscribed_timespan = consort.PerformedTimespan(
...     start_offset=(5, 4),
...     stop_offset=(9, 5),
...     music=Container("{ c'4 }{ c'2 }{ c'4 }"),
... )
>>> division = inscribed_timespan.music[1]
>>> division_timespan = inspect_(division).get_timespan()
>>> print(format(division_timespan))
timespantools.Timespan(
    start_offset=durationtools.Offset(1, 4),

```

---

<sup>3</sup>It may be undesirable to rewrite a rhythm's meter in certain situations, particularly if a composer is trying to avoid the introduction of ties or dots for whatever reason.

```

stop_offset=durationtools.Offset(3, 4),
)

>>> translation = inscribed_timespan.start_offset
>>> division_timespan = division_timespan.translate(translation)
>>> print(format(division_timespan))
timespanTools.Timespan(
    start_offset=durationtools.Offset(3, 2),
    stop_offset=durationtools.Offset(2, 1),
)

>>> meter_timespan = meter_timespans.find_timeSpans_intersecting_timeSpan(
...     division_timespan)[0]
>>> print(format(meter_timespan))
timeSpanTools.AnnotatedTimespan(
    start_offset=durationtools.Offset(5, 4),
    stop_offset=durationtools.Offset(2, 1),
    annotation=meterTools.Meter(
        '(6/8 ((3/8 (1/8 1/8 1/8)) (3/8 (1/8 1/8 1/8))))',
        ),
)
)

>>> translation = -1 * division_timespan.start_offset
>>> meter_timespan = meter_timespan.translate(translation)
>>> print(format(meter_timespan))
timeSpanTools.AnnotatedTimespan(
    start_offset=durationtools.Offset(-1, 4),
    stop_offset=durationtools.Offset(1, 2),
    annotation=meterTools.Meter(
        '(6/8 ((3/8 (1/8 1/8 1/8)) (3/8 (1/8 1/8 1/8))))',
        ),
)
)

```

With the appropriate meters selected, rewriting continues very much as described in subsection 3.9.4. Tuplets are rewritten solely with respect for the pre-prolated contents durations, and unprolated containers are rewritten with respect for their intersecting meter, with an initial offsets applied to the meter rewriting process if they happen to start later than their meter. Additionally, Consort's meter rewriting tests silent meters – those whose leaves consist entirely of rests – for congruency with the current meter. Any division consisting solely of rests which also begins and ends at the start and stop offsets of a meter's timespan can be rewritten instead as a single full-bar rest. The segment-maker also attaches a `StaffLinesSpanner` to the silent division, which collapses the staff down to a single line, giving the score a fragmented appearance. Finally, Consort's segment-maker performs a logical-tie cleanup pass, fusing all 2-length logical ties consisting of matched pairs of  $\frac{1}{16}$  or  $\frac{1}{32}$  notes. This takes care of some possible artifacts of heavily subdivided meter-rewriting, and makes the final rhythmic output generally more readable.<sup>4</sup>

---

<sup>4</sup>Special thanks to Rei Nakamura for the suggestion.

### 4.3.10 POPULATING THE SCORE

After meter-rewriting, the segment-maker can finally populate its score. To do so, it iterates through its timespan maquette and still-unpopulated score in parallel, extracting the inscribed music from each performed timespan in the maquette and inserting those phrases into the score in the appropriate voice. With the segment-maker's score populated, rhythmic interpretation ends and non-rhythmic interpretation may begin.

## 4.4 NON-RHYTHMIC INTERPRETATION

Non-rhythmic interpretation involves the process of progressively embellishing the contents of the rhythmically-interpreted score while preserving the score hierarchy and attack-point structure. During non-rhythmic interpretation, the segment-maker may attach grace notes, attach indicators and spanners, change the pitches of notes or even replace those notes with chords.

### 4.4.1 SCORE TRAVERSAL

Recall the earlier discussion in subsection 2.6.2 regarding score iteration techniques. Of the various possible types of iteration, Consort's handlers primarily make use of two during non-rhythmic interpretation. The first – *attack-point* or *time-wise logical tie* iteration – iterates through all logical ties in the score in *time order* according to their start offset in the score, regardless of their vertical position within the score. Logical ties with identical start offsets – those appearing at simultaneities across voices – are then sorted by their *score order*. This results in an iteration which moves first forward in time and then from the top of the score to the bottom. The second technique – *phrase-wise* iteration – locates each voice context in the score, then iterates through the top-level containers in that voice. These top-level containers are the same containers reference by each performed timespan's `music` property, and represent each phrase in the maquette. Attack-point iteration helps maintain the continuity of some process across multiple voices in time, while phrase-wise iteration returns entire phrases and therefore allows processes to treat those phrases and all of the components located in the subtree rooted at that phrase as a single group.

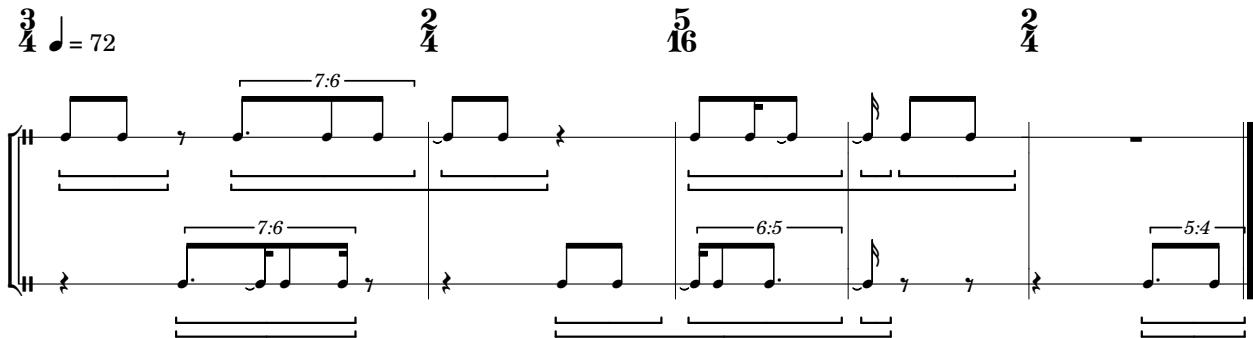
Consider the following two voice score, produced by a segment-maker and annotated to show each division and phrase:

```
>>> musicSpecifier = consort.MusicSpecifier(  
...     attachmentHandler=consort.AttachmentHandler(),  
...     rhythmMaker=rhythmmakertools.TaleaRhythmMaker(  
...         extraCounts_per_division=(0, 1),  
...         talea=rhythmmakertools.Talea([2, 3, 2, 4], 16),
```

```

...     ),
...
... )
>>> timespan_maker = consort.TaleaTimespanMaker(
...     initial_silence_talea=rhythmmakertools.Talea([0, 1], 4),
...     playing_groupings=(1, 2, 2, 1, 2),
...     playing_talea=rhythmmakertools.Talea([2, 3], 8),
...     silence_talea=rhythmmakertools.Talea([1, 2, 3, 4], 8),
... )
>>> music_setting = consort.MusicSetting(
...     timespan_maker=timespan_maker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segment_maker = consort.SegmentMaker(
...     desired_duration_in_seconds=8,
...     discard_final_silence=True,
...     permitted_time_signatures=[(2, 4), (5, 16), (3, 4)],
...     score_template=templatetools.GroupedRhythmicStavesScoreTemplate(
...         staff_count=2,
...         with_clefs=True,
...     ),
...     settings=[music_setting],
...     tempo=indicatortools.Tempo((1, 4), 72),
... )
>>> illustration, metadata = segment_maker(annotate=True, verbose=False)
>>> show(illustration)

```



At the beginning of non-rhythmic interpretation, the segment-maker constructs an *attack-point map*: a mapping of pitched logical ties against *attack-point signatures*. Consort's `AttackPointSignature` class caches information about the location of each pitched logical tie in the segment, including its normalized position within its division, phrase and segment and its indices within the same. The segment-maker constructs the attack-point map by iterating over all note and chords in its score in time-wise order. It selects the logical tie for each iterated leaf, skipping those for which the leaf is not the logical tie's head, and constructs an attack-point signature for the logical tie, storing both in an ordered dictionary. The segment-maker caches its attack-point map on its instance, allowing it to be referenced by its own methods, as well as examined after interpretation completes.

```

>>> attack_point_map = segment_maker.attack_point_map
>>> all_pairs = list(attack_point_map.items())
>>> first_logical_tie, first_attack_point_signature = all_pairs[0]
>>> print(first_logical_tie)
LogicalTie(Note("c'8"),)

>>> print(format(first_attack_point_signature))
consort.tools.AttackPointSignature(
    division_index=0,
    division_position=durationtools.Multiplier(0, 1),
    logical_tie_index=0,
    phrase_position=durationtools.Multiplier(0, 1),
    segment_position=durationtools.Multiplier(0, 1),
    total_divisions_in_phrase=1,
    total_logical_ties_in_division=2,
)

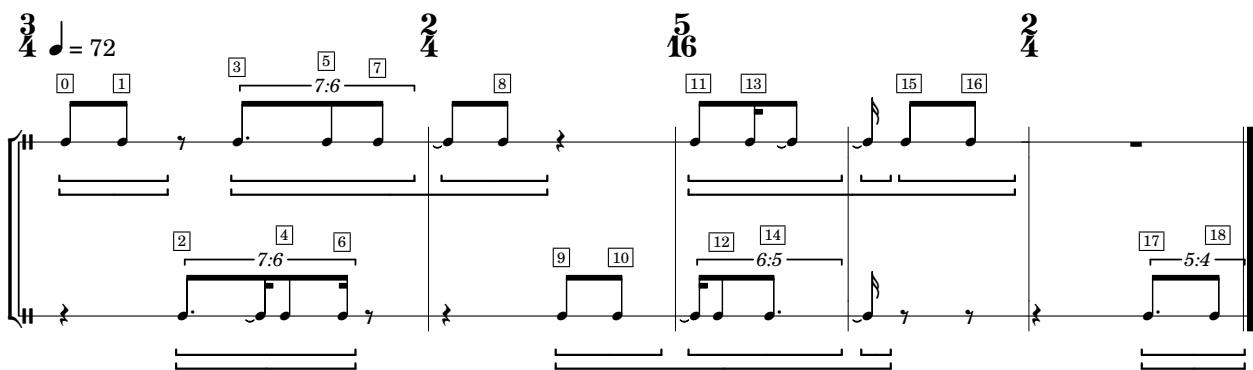
```

Because the attack-point map was constructed in time-wise order, each cached logical tie can be visited in time order when iterating over it. The following example iterates over the attack-point map, retrieving each key/value pair – comprised of a logical-tie selection and an attack-point signature – while enumerating them – producing the index of that pair, e.g. first, second, third, etc. – and attaches some markup to the head of that logical tie comprised of its index formatted within a padded box:

```

>>> for index, key_value_pair in enumerate(attack_point_map.items()):
...     logical_tie, attack_point_signature = key_value_pair
...     markup = Markup(index, Up)
...     markup = markup.smaller().pad_around(0.25).box()
...     attach(markup, logical_tie.head)
...
>>> show(illustration)

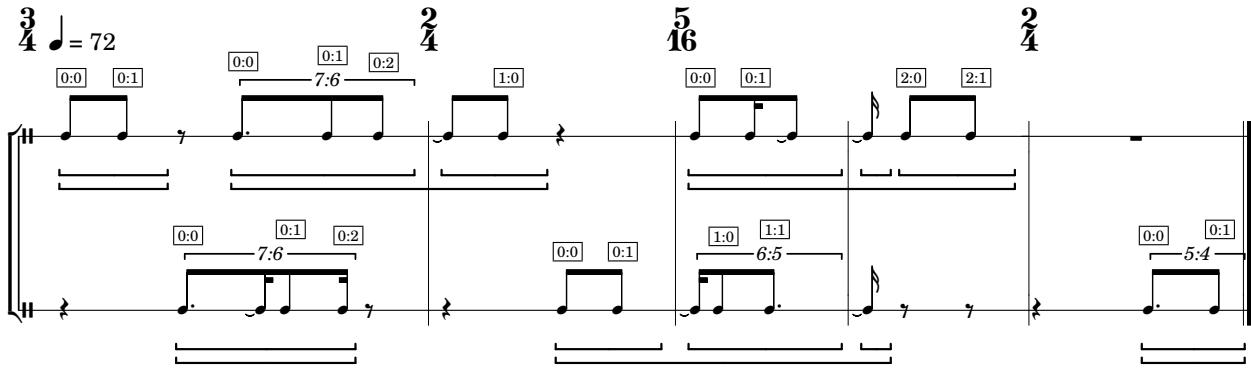
```



Attack-point signatures cache each logical-tie's *logical tie index* and *division index*. A logical tie's logical tie index gives its index within the list of all logical ties starting within its division. A logical tie's division index give the index of the division within which it starts within the phrase itself. The following iteration shows the division index and

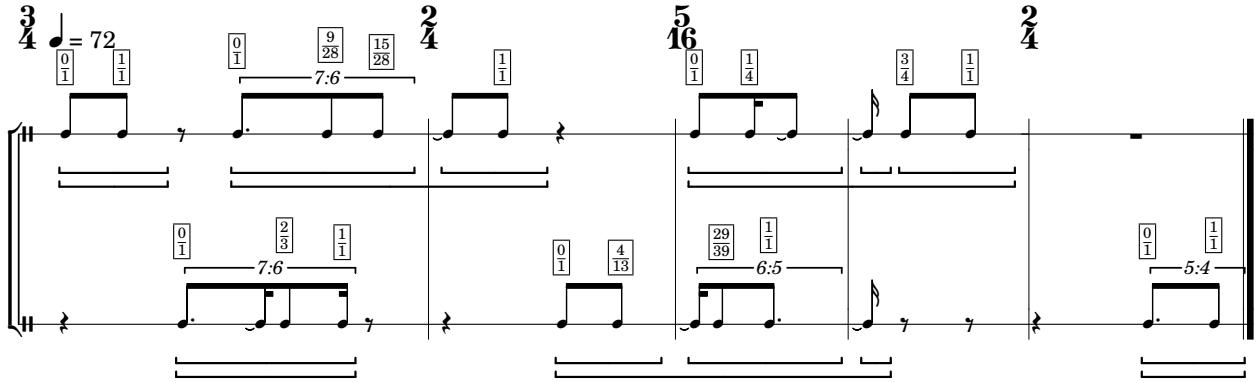
logical tie index of each attack-point, boxed and separated by a colon. The annotation brackets clarify the divisions within each phrase:

```
>>> for logical_tie, attack_point_signature in attack_point_map.items():
...     for markup in inspect_(logical_tie.head).get_markup():
...         detached = detach(markup, logical_tie.head)
...         string = '{}:{}{}'.format(
...             attack_point_signature.division_index,
...             attack_point_signature.logical_tie_index,
...             )
...         markup = Markup(string, Up)
...         markup = markup.smaller().pad_around(0.25).box()
...         attach(markup, logical_tie.head)
...
>>> show(illustration)
```



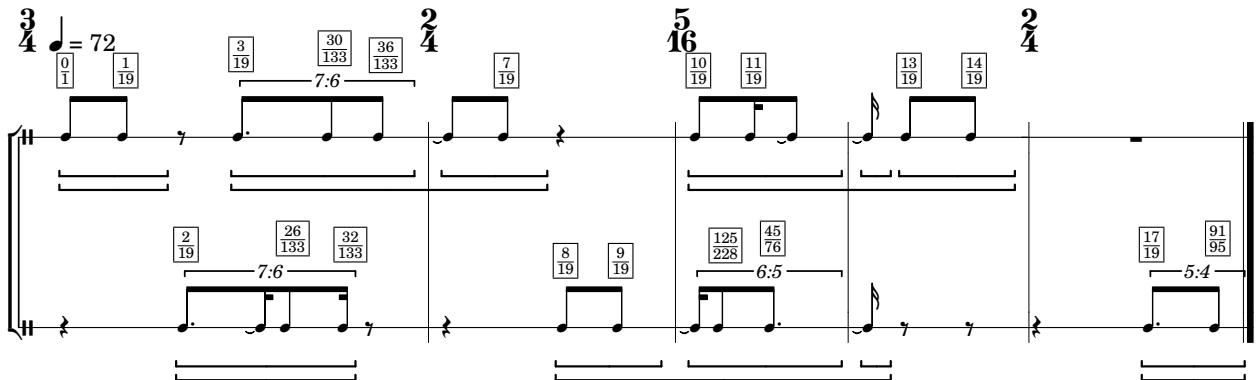
Attack-point signatures maintain the position of each logical tie's start offset normalized between 0 and 1 within its division, as well as its phrase. The following iteration show each logical tie's phrase position as a fraction, after detaching the previously attached markup:

```
>>> for logical_tie, attack_point_signature in attack_point_map.items():
...     for markup in inspect_(logical_tie.head).get_markup():
...         detached = detach(markup, logical_tie.head)
...         phrase_position = attack_point_signature.phrase_position
...         markup = Markup.fraction(phrase_position)
...         markup = Markup(markup, Up)
...         markup = markup.smaller().pad_around(0.25).box()
...         attach(markup, logical_tie.head)
...
>>> show(illustration)
```



Attack-point signatures also maintain the position of each logical tie's start offset in the context of the entire segment's timespan:

```
>>> for logical_tie, attack_point_signature in attack_point_map.items():
...     for markup in inspect_(logical_tie.head).get_markup():
...         detached = detach(markup, logical_tie.head)
...         segment_position = attack_point_signature.segment_position
...         markup = Markup.fraction(segment_position)
...         markup = Markup(markup, Up)
...         markup = markup.smaller().pad_around(0.25).box()
...         attach(markup, logical_tie.head)
...
>>> show(illustration)
```



Consort's phrase-wise iteration can be demonstrated by building an iterator function. The following function iterates through its score argument by voice, then iterates over the top-level containers within that voice. For each iterated container, it checks if that container's *effective* music specifier is equivalent to the "silent" music specifier which Consort attaches to its silent phrases. If the music specifiers match, the phrase must be silent, and the function *continues* past it. If the music specifiers don't match, the phrase must have some significant musical content, and therefore the function yields that phrase.

```

>>> def iterate_phrasewise(score):
...     prototype = consort.MusicSpecifier
...     silent_music_specifier = consort.MusicSpecifier()
...     for voice in iterate(score).by_class(Voice):
...         for phrase in voice:
...             music_specifier = inspect_(phrase).get_effective(prototype)
...             if music_specifier == silent_music_specifier:
...                 continue
...             yield phrase
...

```

Before running the iterator to attach the phrase-index markup, the previously attached markup must be removed:

```

>>> for logical_tie in attack_point_map.keys():
...     for markup in inspect_(logical_tie.head).get_markup():
...         detached = detach(markup, logical_tie.head)

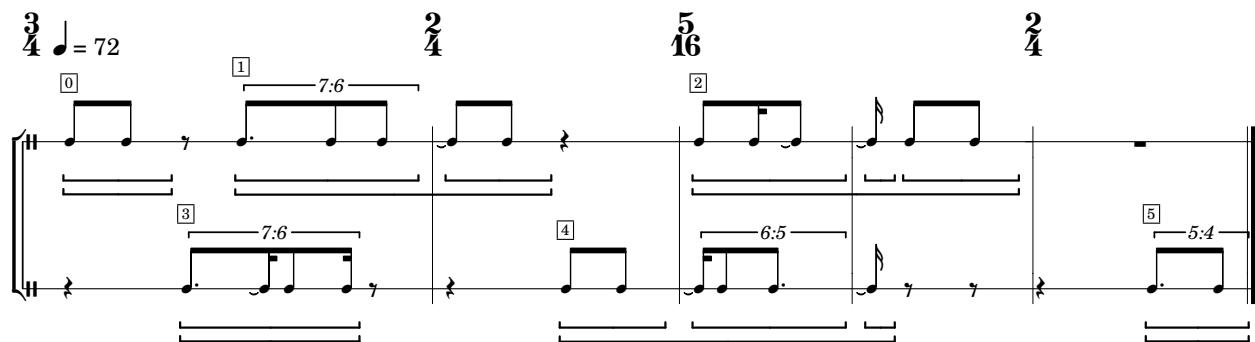
```

By chaining a call to the above-created `iterate_phrasewise()` function with Python's built-in `enumerate()`, phrases can be extracted from the score along with their index. The result simply counts each non-silent phrase left-to-right in each voice, from the top of the score to the bottom:

```

>>> for index, phrase in enumerate(iterate_phrasewise(segment_maker.score)):
...     first_leaf = phrase.select_leaves()[0]
...     markup = Markup(index, Up)
...     markup = markup.smaller().pad_around(0.25).box()
...     attach(markup, first_leaf)
...
>>> show(illustration)

```

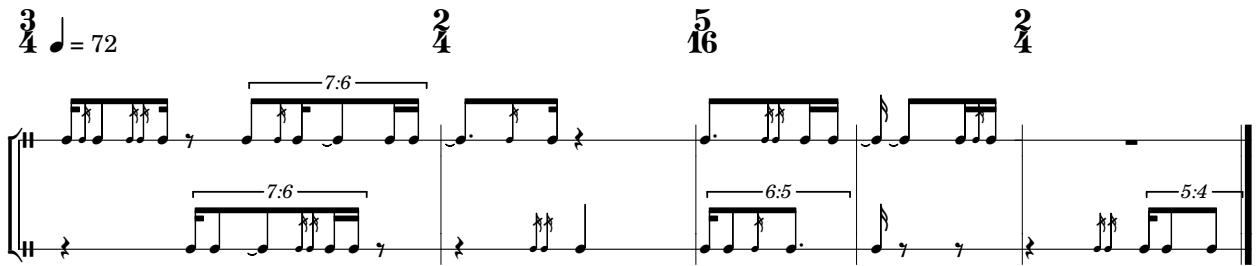


Some of the handlers used during non-rhythmic interpretation rely on information about each pitched logical tie in the score. Both grace- and pitch-handlers iterate through the score in time order. While grace-handlers do not make use of information stored within each logical tie's associated attack-point signature – in the current implementation of Consort only pitch-handlers rely on attack-point signatures –, simply caching the time-order of all logical ties within the segment saves multiple complex iteration procedures.

#### 4.4.2 GRACE-HANDLERS

Consort's segment-maker can be instructed to attach grace notes to various parts of the score by defining a *grace-handler* on a music-specifier. Grace-handlers attach *grace containers* to logical ties within a score in a patterned way, according to a cyclic sequence of counts, as well as collection of flags which restrict what leaves may be selected for attachment. Abjad implements grace notes as normal leaves – notes, chords and rests – within special components which act both as Abjad Container classes as well as attachable indicators. That is to say, grace notes must be placed within one of these grace containers which is then attached to another leaf in the score much like any other indicator such as a dynamic indication or articulation. Grace-handlers traverse the score by logical tie in time-order, iterating over the previously cached ordered dictionary of attack-points which was generated at the beginning of non-rhythmic interpretation. Consider the following music specifier, whose grace-handler places grace notes with a cyclic count of 1, 2, 0, 0, 0.

```
>>> musicSpecifier = consort.MusicSpecifier(
...     graceHandler=consort.GraceHandler(
...         counts=(1, 2, 0, 0, 0),
...         ),
...     rhythmMaker=rhythmmakertools.TaleaRhythmMaker(
...         extraCounts_per_division=(0, 1),
...         talea=rhythmmakertools.Talea([1, 2, 3, 1, 4], 16),
...         ),
...     )
>>> timespanMaker = consort.TaleaTimespanMaker(
...     initialSilenceTalea=rhythmmakertools.Talea([0, 1], 4),
...     playingGroupings=(1, 2, 2),
...     playingTalea=rhythmmakertools.Talea([2, 3], 8),
...     silenceTalea=rhythmmakertools.Talea([1, 2, 3, 4], 8),
...     )
>>> musicSetting = consort.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
...     )
>>> segmentMaker = consort.SegmentMaker(
...     desiredDuration_in_seconds=8,
...     discardFinalSilence=True,
...     permittedTimeSignatures=[(2, 4), (5, 16), (3, 4)],
...     scoreTemplate=templatetools.GroupedRhythmicStavesScoreTemplate(
...         staffCount=2,
...         withClefs=True,
...         ),
...     settings=[musicSetting],
...     tempo=indicatortools.Tempo((1, 4), 72),
...     )
>>> show(segmentMaker, verbose=False)
```



Note above that the first note in the score by time-wise ordering – the initial  $\frac{1}{16}$  note in the upper staff – does not receive the expected single grace note. The grace note pattern starts instead on the  $\frac{1}{8}$  note that follows. For purely practical reasons, Consort's grace-handler will *not* put grace notes before the first leaf in any voice in a score. LilyPond's grace spacing algorithm does not operate well with its strict proportional spacing algorithm. When spaced strictly proportionally, grace notes which are positioned before the beat – via LilyPond's \grace, \appoggiatura or \acciaccatura commands – may end up colliding with other glyphs in the staff. Consort works around this by using LilyPond's \afterGrace command, which places grace notes after the note to which they attach. This positioning allows grace notes to avoid most collisions. Unfortunately, this also means that grace notes must always appear after a note or rest, making it impossible to start a segment with grace notes.

#### 4.4.3 PITCH-HANDLERS

Consort's *pitch-handlers* apply pitches to logical ties within a score in a patterned way, through the use of various cyclic sequences applied to each logical tie in time-wise order, much like grace-handlers. Consider the following short segment, containing two voices of  $\frac{1}{16}$  note rhythms interspersed with rests. Without any pitch-handler specified, both voices default to creating notes pitched at middle-C:

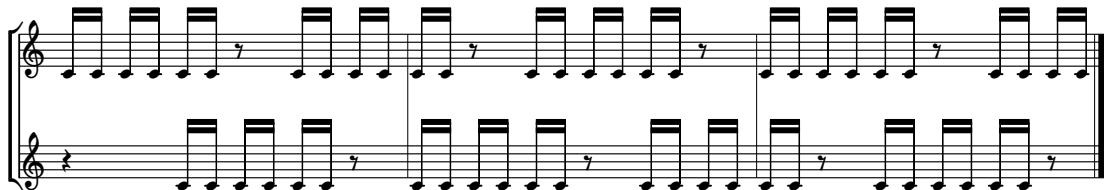
```
>>> segment_maker = consort.SegmentMaker(
...     desired_duration_in_seconds=9,
...     #omit_stylesheets=True,
...     permitted_time_signatures=[(3, 4)],
...     score_template=templatetools.GroupedStavesScoreTemplate(
...         staff_count=2,
...     ),
...     tempo=indicatortools.Tempo((1, 4), 60),
... )
>>> musicSpecifier = consort.MusicSpecifier(
...     rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
...         talea=rhythmmakertools.Talea([1], 16),
...     ),
... )
>>> timespan_maker = consort.TaleaTimespanMaker(
```

```

...
    initial_silence_talea=rhythmmakertools.Talea([0, 1], 4),
...
    playing_talea=rhythmmakertools.Talea([1], 8),
...
    playing_groupings=[3],
...
    silence_talea=rhythmmakertools.Talea([1], 8),
...
)
>>> segment_maker.add_setting(
...
    timespan_maker=timespan_maker,
...
    v1=music_specifier,
...
    v2=music_specifier,
...
)
>>> show(segment_maker, verbose=False)

```

$\frac{3}{4}$   $\text{♩} = 60$



By reconfiguring the above music specifier with an `AbsolutePitchHandler` – a concrete subclass of `Consort`'s abstract `PitchHandler` – different pitches can be applied to each note in the segment. The following pitch-handler paints a C-major scale across all of the logical ties in the resulting segment. Note how the G4, A4, B4 and C5 of the applied C major scale alternate between the voices according to both their time-wise position and their score order, as discussed in subsection 4.4.1:

```

>>> music_specifier = new(
...
    music_specifier,
...
    pitch_handler=consort.AbsolutePitchHandler(
...
        pitchSpecifier="c' d' e' f' g' a' b' c''",
...
    ),
...
)
>>> music_setting = consortium.MusicSetting(
...
    timespan_maker=timespan_maker,
...
    v1=music_specifier,
...
    v2=music_specifier,
...
)
>>> segment_maker = new(segment_maker, settings=[music_setting])
>>> show(segment_maker, verbose=False)

```

$\frac{3}{4}$   $\text{♩} = 60$



Pitch-handlers apply their pitch patterns in time-wise order regardless of rhythmic texture:

```
>>> musicSpecifier = new(
...     musicSpecifier,
...     rhythmMaker__talea__counts=[1, 2, 3, 4],
... )
>>> musicSetting = consort.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> show(segmentMaker, verbose=False)
```

$\frac{3}{4}$   $\text{♩} = 60$



As discussed in subsection 4.3.5, pitch-handlers, like rhythm-makers, maintain a seed value keyed to the music specifier in which they are defined, allowing them to maintain their pattern even in fragmentary or interrupted musical textures. Note how, when interrupted by phrases defined by a different music specifier, the previously defined texture continues its pitch application pattern after each interruption or silence exactly where it left off. For example, the A4  $\frac{1}{16}$  note in the lower voice in measure one is continued by the B4  $\frac{1}{8}$  note in the upper voice at the beginning of measure two, and followed by the C5  $\frac{1}{8}$  note directly below in the lower voice. The C-major pattern continues despite the intrusion of the triplet texture defined by the other\_music\_specifier:

```
>>> otherMusicSpecifier = consort.MusicSpecifier(
...     pitchHandler=consort.AbsolutePitchHandler(pitchSpecifier='g f# e f'),
...     rhythmMaker=rhythmmakertools.EvenDivisionRhythmMaker(
...         denominators=[8],
...         extraCounts_per_division=(1,),
...     ),
... )
>>> otherMusicSetting = consort.MusicSetting(
...     timespanMaker=consort.TaleaTimespanMaker(
...         initialSilenceTalea=rhythmmakertools.Talea([1], 2),
...         silenceTalea=rhythmmakertools.Talea([1], 2),
...     ),
...     v1=otherMusicSpecifier,
...     v2=otherMusicSpecifier,
... )
>>> segmentMaker = new(
```

```

...
    segment_maker,
...
    settings=[music_setting, other_music_setting],
...
)
>>> show(segment_maker, verbose=False)

```

Pitch-handlers may define explicit *pitch specifiers*, which behave somewhat analogously to the ratio-parts expressions discussed in subsection 4.3.1, describing which pitches or pitch-classes are to be used in which sections of a segment, as partitioned by a ratio:

```

>>> pitchSpecifier = consort.PitchSpecifier(
...     pitch_segments=(
...         "c' e' g'",
...         "fs' g' a'",
...         "b d'",
...         ),
...     ratio=(1, 2, 3),
...     )
>>> pitch_choice_timepans = consort.PitchHandler.get_pitch_choice_timepans(
...     pitchSpecifier=pitchSpecifier,
...     duration=segment_maker.desired_duration,
...     )
>>> print(format(pitch_choice_timepans))
consort.tools.TimespanCollection(
[
    timespantools.AnnotatedTimespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(3, 8),
        annotation=datastructuretools.CyclicTuple(
            [
                pitchtools.NamedPitch("c'"),
                pitchtools.NamedPitch("e'"),
                pitchtools.NamedPitch("g'"),
            ]
        ),
    ),
    timespantools.AnnotatedTimespan(
        start_offset=durationtools.Offset(3, 8),
        stop_offset=durationtools.Offset(9, 8),
        annotation=datastructuretools.CyclicTuple(
            [

```

```

        pitchtools.NamedPitch("fs'"),
        pitchtools.NamedPitch("g'"),
        pitchtools.NamedPitch("a'"),
    ],
),
),
timespantools.AnnotatedTimespan(
    start_offset=durationtools.Offset(9, 8),
    stop_offset=durationtools.Offset(9, 4),
    annotation=datastructuretools.CyclicTuple(
        [
            pitchtools.NamedPitch('b'),
            pitchtools.NamedPitch("d"),
        ]
),
),
),
]
)

```

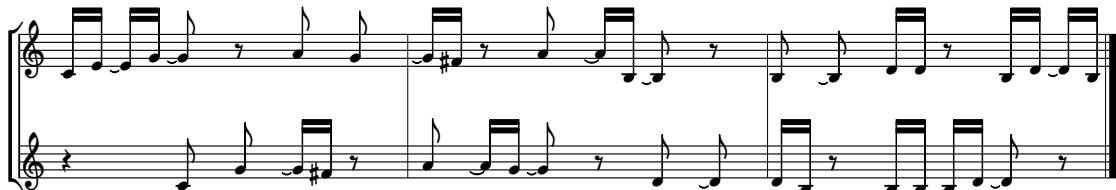
When used to reconfigure the previously-defined music specifier's pitch-handler, each of the pitch specifier's pitch segments appears in only one portion of the resulting score, as defined by the pitch-choice timespan inventory:

```

>>> musicSpecifier = new(
...     musicSpecifier,
...     pitchHandler__pitchSpecifier=pitchSpecifier,
... )
>>> musicSetting = consort.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> show(segmentMaker, verbose=False)

```

$\frac{3}{4}$   $\text{♩} = 60$



Likewise, a pitch-handler may define a pitch-operation specifier, which behaves similarly to pitch specifiers. Pitch-operation specifiers combine a sequence of pitch-operations – transposition, inversion, retrogression, rotation, etc. – with a ratio defining where those operations should appear during the course of a segment. Pitch-operation specifiers may be used alongside pitch specifiers, and with differing ratios. The following pitch-operation specifier

partitions the segment into three unequal parts, applying an operation to the first  $\frac{1}{4}$ , no operation to the middle  $\frac{1}{2}$ , and another operation to the final  $\frac{1}{4}$  of the segment:

```
>>> musicSpecifier = new(
...     musicSpecifier,
...     pitchHandler__pitchOperationSpecifier=consort.PitchOperationSpecifier(
...         pitchOperations=(
...             pitchtools.PitchOperation((
...                 pitchtools.Inversion(),
...             )),
...             None,
...             pitchtools.PitchOperation((
...                 pitchtools.Rotation(-1),
...                 pitchtools.Transposition(-1),
...             ))
...         ),
...         ratio=(1, 2, 1),
...     ),
... )
>>> musicSetting = consortium.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> show(segmentMaker, verbose=False)
```

$\frac{3}{4}$   $\text{♩} = 60$

Pitch-handlers may also be configured to forbid immediate pitch repetitions within the same voice via the `forbid_repetitions` flag, assuming the current logical tie has more than one pitch available via its `pitchChoices`:

```
>>> musicSpecifier = new(
...     musicSpecifier,
...     pitchHandler__forbidRepetitions=True,
... )
>>> musicSetting = consortium.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> show(segmentMaker, verbose=False)
```

$\frac{3}{4}$   $\text{♩} = 60$



Chords may be applied in a patterned way to the logical ties a pitch-handler iterates over by defining a sequence of *logical tie expressions* – classes which, when called on a logical tie, modify or replace each of the leaves in that logical tie, leaving the tie itself in place. Consort provides a few subclasses of logical tie expression, such as the ChordExpression, HarmonicExpression and KeyClusterExpression. When called on a logical tie, chord expression replace each note in the tie with a chord whose pitches may be specified either absolutely or in terms of intervals relative to the original pitch. Note that Python’s None, like in the previously defined pitch-operation specifier, acts as a “no-op”, making no change at all:

```
>>> musicSpecifier = new(
...     musicSpecifier,
...     pitchHandler__logical_tie_expressions=(
...         consort.ChordExpression(chord_expr=(-2, 0, 2)),
...         consort.ChordExpression(chord_expr=(-7, 0, 7)),
...         None,
...     ),
... )
>>> musicSetting = consort.MusicSetting(
...     timespan_maker=timespan_maker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> show(segmentMaker, verbose=False)
```

$\frac{3}{4}$   $\text{♩} = 60$



All of the above pattern definitions – pitch specifiers, pitch-operation specifiers and sequence of logical tie expressions – continue to apply in order even over an interrupted or otherwise fragmentary texture:

```

>>> segment_maker = new(
...     segment_maker,
...     settings=[music_setting, other_music_setting],
... )
>>> show(segment_maker, verbose=False)

```

$\frac{3}{4}$   $\text{♩} = 60$

Now, consider again the  $\frac{1}{16}$  note music specifier from earlier in this section, whose pitch-handler cycled through a C-major scale. Pitch-handlers may be configured to use the same pitch for each division, or even phrase, rather than choosing a new pitch for each encountered logical tie. This behavior is defined by the pitch-handler's *application rate*, which is quite analogous to the music specifier sequence application rate discussed in subsection 4.1.3. The following pitch handler applies the same pitch from its pitch specifier to each logical tie within each division. Both the divisions and phrases have been annotated for clarity:

```

>>> musicSpecifier = consort.MusicSpecifier(
...     pitch_handler=consort.AbsolutePitchHandler(
...         pitch_application_rate='division',
...         pitchSpecifier="c' d' e' f' g' a' b' c''',
...     ),
...     rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(denominators=[16]),
... )
>>> musicSetting = consort.MusicSetting(
...     timespan_maker=timespan_maker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segment_maker = new(segment_maker, settings=[musicSetting])
>>> lilypond_file, metadata = segment_maker(verbose=False)
>>> consort.annotate(lilypond_file.score)
>>> show(lilypond_file)

```

$\frac{3}{4}$   $\text{♩} = 60$



Per-division and per-phrase pitch application rates rely on the attack-point signature objects collected at the beginning of non-rhythmic interpretation to determine if any encountered logical tie is the first of its division or phrase. Pitch-handlers also require more complex seed-tracking logic – maintained by a dedicated `SeedSession` class –, as each encountered logical tie may not require choosing a new pitch, but could still require choosing a new logical tie expression. The above music specifier can be re-configured to apply the same pitch to every logical tie in a phrase:

```
>>> musicSpecifier = new(
...     musicSpecifier,
...     pitchHandler__pitchApplicationRate='phrase',
... )
>>> musicSetting = consort.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> lilypondFile, metadata = segmentMaker(annotate=True, verbose=False)
>>> show(lilypondFile)
```

$\frac{3}{4}$   $\text{♩} = 60$



Additionally, a pattern of interval deviations may be applied on top of any chosen pitches, regardless of the pitch-handler's application rate. This provides a mechanism for intermittently varying an otherwise stable melodic contour:

```

>>> musicSpecifier = new(
...     musicSpecifier,
...     pitchHandler__deviations=(0, 0, '-m2', '+m2'),
... )
>>> musicSetting = consort.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> lilypondFile, metadata = segmentMaker(annotate=True, verbose=False)
>>> show(lilypondFile)

```

Other pitch-handlers are possible. For example, Consort’s `PitchClassPitchHandler` first applies pitch-classes to each logical tie, then an octavation according to a *register specifier* which acts similarly to a break-point function, shifting the pitch-class up and down by octaves according to the shape of the specifier and the logical tie’s position within its division, phrase and segment. Pitch-handlers based on vertical sonorities, spectral information, or various other techniques could also be implemented. As it stands, I have so far only implemented two for the pieces I have written with Consort, finding them sufficient for now.

#### 4.4.4 ATTACHMENT-HANDLERS

Attachment-handlers manage the process of attaching indicators and spanners to selections of components within a score. Unlike grace-handlers and pitch-handlers, they iterate over the score by voice and phrase, rather than by logical-tie time-wise. Phrase-wise iteration allows attachment-handlers to create highly-contextualized attachments, considering each component in a phrase in terms of each other component. Attachment-handlers aggregate *attachment expressions*, objects pairing a *component selector* – as described in subsection 2.6.3 – and an iterable of attachments – indicators and spanners – or component expressions. Selectors chain together a series of *callbacks* which progressively refine a selection as each callback processes it. Much like the other handler classes described already, an attachment-expression’s attachments sequence cycles, allowing different groups of attachments to be attached

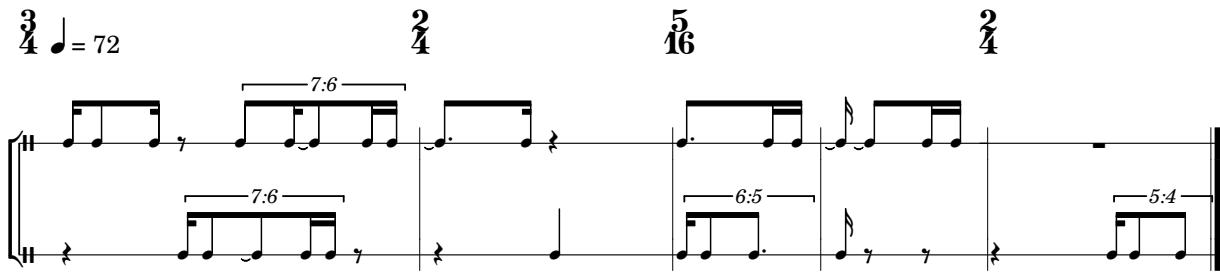
each time the attachment-expression is called with a different seed value. Attachment-handlers associate their attachment expressions with underscore-delimited string keys. This mechanism, nearly identical to that employed by music settings for associating music specifiers with voice-name abbreviations, allows attachment expressions – which may have an arbitrary number of such associations – to be reconfigured through templating to add new attachment expressions or overwrite or nullify specific existing expressions.<sup>5</sup>

Consider the following two-staff segment containing phrases interspersed by rests and a surface-rhythm with a variety of tuplets and ties:

```
>>> musicSpecifier = consort.MusicSpecifier(
...     attachmentHandler=consort.AttachmentHandler(),
...     rhythmMaker=rhythmmakertools.TaleaRhythmMaker(
...         extraCounts_per_division=(0, 1),
...         talea=rhythmmakertools.Talea([1, 2, 3, 1, 4], 16),
...     ),
... )
>>> timespanMaker = consort.TaleaTimespanMaker(
...     initialSilenceTalea=rhythmmakertools.Talea([0, 1], 4),
...     playingGroupings=(1, 2, 2),
...     playingTalea=rhythmmakertools.Talea([2, 3], 8),
...     silenceTalea=rhythmmakertools.Talea([1, 2, 3, 4], 8),
... )
>>> musicSetting = consort.MusicSetting(
...     timespanMaker=timespanMaker,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = consort.SegmentMaker(
...     desiredDuration_in_seconds=8,
...     discardFinalSilence=True,
...     permittedTimeSignatures=[(2, 4), (5, 16), (3, 4)],
...     scoreTemplate=templatetools.GroupedRhythmicStavesScoreTemplate(
...         staffCount=2,
...         withClefs=True,
...     ),
...     settings=[musicSetting],
...     tempo=indicatortools.Tempo((1, 4), 72),
... )
>>> show(segmentMaker, verbose=False)
```

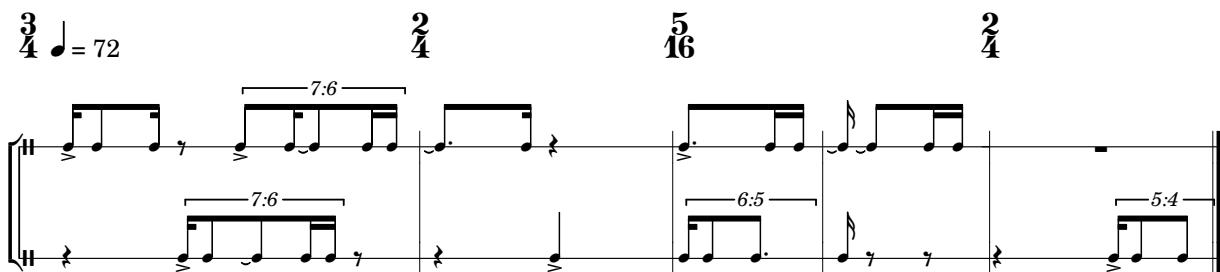
---

<sup>5</sup>The principle employed by both music settings and attachment handlers is crucial: named references beat positional references for ease of reconfiguration.



Accents can be attached to the first leaf of each phrase by configuring the music specifier's attachment-handler with an attachment expression whose attachments are simply an instance of an `Abjad Articulation` and whose selector has been configured to first select all leaves and then select the first item in that selection of leaves: the first leaf:

```
>>> attachment_expression = consort.AttachmentExpression(
...     attachments=Articulation('accent'),
...     selector=selectortools.Selector().by_leaves()[0],
... )
>>> music_specifier = new(
...     music_specifier,
...     attachment_handler__accents=attachment_expression,
... )
>>> music_setting = new(
...     music_setting,
...     v1=music_specifier,
...     v2=music_specifier,
... )
>>> segment_maker = new(segment_maker, settings=[music_setting])
>>> show(segment_maker, verbose=False)
```



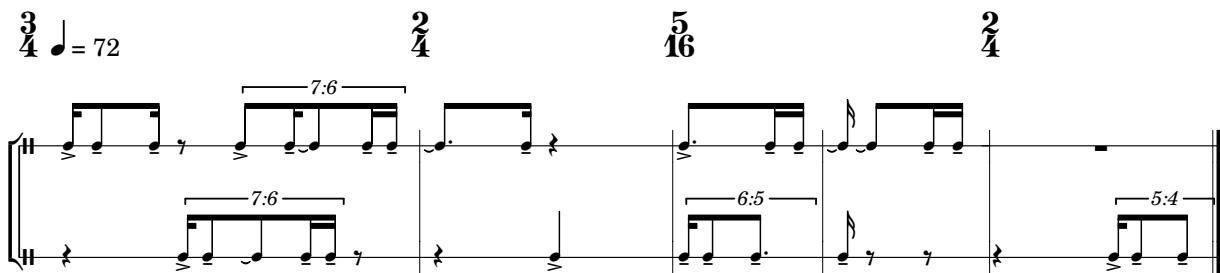
Tenuto articulations can be attached to the head of every logical tie in each phrase – except the very first – with a more complex selector. The following tenuto-attaching attachment-expression's selector first selects all leaves in each phrase, then selects all pitched logical ties, then selects all but the first of those logical ties – rather than all but the first leaf in *each* of those logical ties – and finally selects the first leaf in each selected logical tie:

```
>>> music_specifier = new(
...     music_specifier,
...     attachment_handler__tenuti=consort.AttachmentExpression(
...         attachments=Articulation('tenuto'),
...         selector=selectortools.Selector()
```

```

...
    .by_leaves()
...
    .by_logical_tie(pitched=True)
...
    .get_slice(start=1, apply_to_each=False)
[0]
...
),
...
)
>>> music_setting = new(
...
    music_setting,
...
    v1=musicSpecifier,
...
    v2=musicSpecifier,
...
)
>>> segment_maker = new(segment_maker, settings=[music_setting])
>>> show(segment_maker, verbose=False)

```

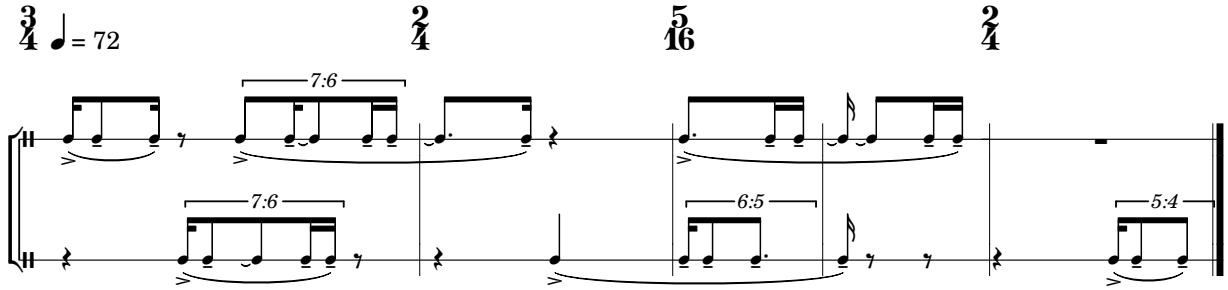


Attaching a spanner to the entirety of a phrase is a much simpler process. When configuring an attachment-handler with a new key, if the value of that key is not already an attachment expression, the attachment-handler creates a new attachment expression and sets the value as the attachment expression's attachment list. Likewise, an attachment expression which has no selector defined uses a default selector when called, which simply selects whatever the attachment expression was called against. In the following reconfiguration, a slur is attached to the entirety of each phrase, using the configuration coercion described above:

```

>>> musicSpecifier = new(
...
    musicSpecifier,
...
    attachmentHandler__slurs=Slur()
...
)
>>> music_setting = new(
...
    music_setting,
...
    v1=musicSpecifier,
...
    v2=musicSpecifier,
...
)
>>> segment_maker = new(segment_maker, settings=[music_setting])
>>> show(segment_maker, verbose=False)

```



Attachment-handlers and their attachment expressions can also delegate to objects specifically designed for attaching indicators and spanners, much like the logical tie expressions outlined in subsection 4.4.3. For example, Consort's *dynamic expression* attaches dynamic indications, potentially spanned by hairpins, to the first leaf in each division of a phrase, and to the last leaf of the phrase as well, taking care of special cases such as divisions with insufficient numbers of leaves. Dynamic expressions encapsulate their own selection logic which is generally too difficult to implement simply through chaining component selector callbacks. Consider the locations of the dynamics in the following annotated segment:

```
>>> musicSpecifier = new(
...     musicSpecifier,
...     attachmentHandler__dynamics=consort.DynamicExpression(['f', 'p'])
... )
>>> musicSetting = new(
...     musicSetting,
...     v1=musicSpecifier,
...     v2=musicSpecifier,
... )
>>> segmentMaker = new(segmentMaker, settings=[musicSetting])
>>> lilypondFile, metadata = segmentMaker(annotate=True, verbose=False)
>>> show(lilypondFile)
```

Note above how each division in each phrase begins with a dynamic, and the last division in each phrase – including those which are the *only* division – contain two dynamics, with the exception of those that only contain a single

logical tie.

#### 4.4.5 EXPRESSIVE ATTACHMENTS

Complex attachment-based formatting scenarios can also be constructed without relying on component expressions such as `DynamicExpression`. One technique used extensively in my scores, especially *Invisible Cities (ii): Armilla*, is to attach various idiomatic indicators as non-formatting annotations to components throughout each phrase, and then to attach a specially-designed spanner to the entirety of each phrase which knows how to inspect the leaves it covers for the previously attached annotations. For example, Consort's `StringContactSpanner` can inspect its leaves for `StringContactPoint` indicators and then construct markup for each contact point – sul ponticello, ordinario, and so forth – bridged by arrows when the contact points change, and parenthesizing the contact point markup when cautionary. Likewise, the `BowContactSpanner` can inspect its leaves for `BowContactPoint` indicators – which indicate the current position along the hair of the bow – as well as `BowMotionTechnique` indicators – which describe various motion qualities like *jeté* and tremolo – replacing the note heads of the staff with fractions for the current bow position and adding up- or down-bow markup as necessary. When combined with the appropriate typographic overrides, complex graphic notation results:

```
>>> staff = Staff()
>>> staff.extend(r"c'4. c'8 \times 2/3 { c'4 c'4 c'4 }")

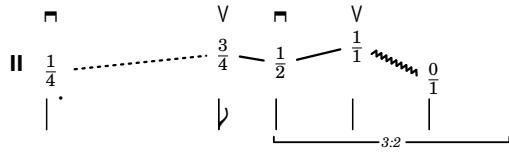
>>> leaves = staff.select_leaves()
>>> attach(indicatortools.BowMotionTechnique('jeté'), leaves[0])
>>> attach(indicatortools.BowContactPoint((1, 4)), leaves[0])
>>> attach(indicatortools.BowContactPoint((3, 4)), leaves[1])
>>> attach(indicatortools.BowContactPoint((1, 2)), leaves[2])
>>> attach(indicatortools.BowMotionTechnique('circular'), leaves[3])
>>> attach(indicatortools.BowContactPoint((1, 1)), leaves[3])
>>> attach(indicatortools.BowContactPoint((0, 1)), leaves[4])

>>> attach(Clef('percussion'), staff)
>>> override(staff).bar_line.transparent = True
>>> override(staff).dots.staff_position = -8
>>> override(staff).flag.Y_offset = -8.5
>>> override(staff).glissando.bound_details_left_padding = 1.5
>>> override(staff).glissando.bound_details_right_padding = 1.5
>>> override(staff).glissando.thickness = 2
>>> override(staff).script.staff_padding = 3
>>> override(staff).staff_symbol.transparent = True
>>> override(staff).stem.direction = Down
>>> override(staff).stem.length = 8
>>> override(staff).stem.stem_begin_position = -9
>>> override(staff).time_signature.stencil = False
```

```

>>> attach(spanertools.BowContactSpanner(), leaves)
>>> show(staff)

```



Notation created in this way treats spanners as typographic post-processors. The positioning of the various bowing indicators – the bow motion technique indicators, the bow contact point indicators – constitutes the compositional act, and the bookkeeping inherent to drawing the correctly formatted lines between them simply constitutes one part of the automated typesetting process.

#### 4.4.6 POST-PROCESSING

With grace-, pitch- and attachment-handling completed, non-rhythmic interpretation proceeds to its final step – post-processing –, during which various typographic adjustments are made to the score.

First, Consort’s segment-maker creates a *floating time-signature context* – described in subsection 3.9.5 – and inserts it as the first child of the score, assuming the score template did not already provide for one. The segment-maker then populates its time-signature context with measures containing typographic spacer-skips, creating the appearance of time signature indications floating above each bar-line at the top of the score. The segment-maker attaches other objects to this context as well, such as the its tempo indication and any configured rehearsal mark or segment name, e.g. “Interlude...”. Segments can also be configured with repeat signs or final bar-line indications. Consort’s segment-maker attaches such indicators to the appropriate leaves in the score during post-processing. Likewise, if a segment has been configured as the final segment, any colophon information – the locations and dates of composition – will be added beneath the last measure of the score.

More complex post-processing may also be carried out. For reasons related solely to how LilyPond handles various typographic constructs during typesetting, it may prove necessary to copy certain voices in the score, maintaining all rhythmic information therein, but changing their context name and filtering out various spanners and indicators so that only a specific subset of typographic commands remain. In doing so, it is possible to isolate typographic effects to each voice performed “simultaneously” in a staff. For example, my score *Invisible Cities (ii): Armilla* employs this voice-copying technique to create its bowing-staff typography. Up until the post-processing step of non-rhythmic interpretation, *Armilla*’s bowing voices contain a rhythm with many indicators and spanners attached – bow contact points, string contact points, bow motion techniques, bow contact spanners, string contact

spanners, and so forth. Bow contact spanners and string contact spanners create conflicting typographic overrides when formatted, so it is necessary to duplicate the bowing voice multiple times, maintaining only the relevant indicators and spanners in each duplicate. By subclassing Consort’s built-in segment-maker *Armilla* can extend its score post-processing with an additional step consisting of these voice copying operations.

Last, the segment-maker wraps the score inside a `LilyPondFile` instance – Abjad’s object model for LilyPond input files – and configures the LilyPond file with any necessary stylesheet file includes. By subclassing Consort’s segment-maker, individual score packages can teach their segment-maker subclasses to automatically locate their stylesheet files relative to the package where that segment-maker subclass was defined. Such stylesheets should include context definitions for the `TimeSignatureContext` as well as any other custom contexts defined for the score. As a final test, the segment-maker runs Abjad’s battery of component well-formedness checks against the score. These checks look for score-structure errors such as components appearing in multiple places in the score, ties which join notes with differing pitches, overlapping glissandi, or components with null parent references. If all tests pass, the segment-maker returns the LilyPond file object. Interpretation is complete.

## 4.5 PERSISTENCE & VISUALIZATION

Once interpreted, a segment-maker’s illustration may be persisted to disk as LilyPond syntax for inclusion in other LilyPond files, rendered as a PDF for viewing, or even serialized for other purposes. Composers study the results of interpretation, make changes to each segment’s specification, and re-interpret as necessary, a large-scale re-enactment of interactive programming’s pervasive *read-eval-print* loop paradigm.

This page intentionally left blank.

# 5

## Composition as software development



THROUGHOUT THE PREVIOUS CHAPTERS, I have presented concepts, techniques, classes and functions for modeling notation and composition in code, accompanied by numerous visualizations, all executed in the context of an interactive Python interpreter session. In light of that, it's important to understand that when composing computationally, I rarely work with the interactive console. Interactive sessions demonstrate simple techniques well, and allow for inspection of live code objects, but are prone to typing errors – consider the difficulty of typing dozens of lines perfectly – and ultimately discourage an iterative workflow.

Iteration is crucial for composing large-scale works, and requires that code and other assets be preserved throughout each cycle of definition and visualization so that they can be progressively revised, amended and otherwise re-configured. Code must be saved to disk as modules in order to be executed or imported into other code. Likewise, code should be encapsulated for both manageability and legibility, preferably organized into multiple files articu-

lating their structure or purpose. Of course, once code is arranged as files on disk, a whole variety of additional questions emerge: How should that code be organized? In how many files? How much and which code in each each file? Where should the files be kept? Does the code create other assets? Where are these stored? How are they named? How is the code executed? How is the project structure maintained? How are changes to the project saved, and how does one compare one version to another? And how is the code in the project tested to guarantee that it works at all, let alone creates the musical result one expects and desires?

The answers to such questions – and there often are clear answers – revolve the practicalities of managing not simply a musical endeavor, but a software project: code and directory structuring, development tools and workflows, version control and testing. All of these are discussed, in greater or lesser detail, in the following sections.

## 5.1 SCORE DIRECTORY LAYOUT

When working computationally, a structured, standardized approach to arranging one’s compositional workspace clarifies both the way one thinks about the act of composing, as well as the more menial workflows of document preparation which are inseparable from score-based composition. Standardization also allows one to reuse tools designed to manage one score package on another. Each of the *Invisible Cities* scores included in part II of this dissertation are implemented as Python packages extending both Abjad’s model of notation and Consort’s model of composition. Likewise, each *Invisible Cities* score package is structured into a nearly identical arrangement of eight top-level directories, each with a clearly delineated purpose and substructure.

```
ersilia/
├── __init__.py ..... The score package Python initializer.
├── build/ ..... LilyPond and LaTeX files for building document targets.
├── distribution/ ..... Finished PDFs for performers and conductors.
├── etc/ ..... Notes, to-do lists and plans.
├── makers/ ..... Customized segment-makers, score templates and other classes.
├── materials/ ..... Materials used to configure segment-makers.
├── segments/ ..... Configured segment-makers and their illustrations.
├── stylesheets/ ..... LilyPond stylesheets.
└── test/ ..... The score package test suite.
```

FIGURE 5.1: *Ersilia*’s top-level directory layout.

The top-level directories named in figure 5.1 house specific collections of assets used during composition or document preparation. Note the presence of an `__init__.py` file. This signals to Python that this directory represents an importable Python package. Assuming Python is aware of the location of the score, it and any further subpackages

within it can be imported:

```
| >>> import ersilia
```

Any subdirectories of `ersilia` also containing `__init__.py` files can be imported into the namespace of `ersilia`, allowing for the structured organization of any code assets employed by the score. Consider the various kinds of objects discussed in earlier chapters. Consort very broadly groups the objects used during composition into *makers*, *materials* and *segments*. Makers comprise any classes used during composition, such as segment-maker, pitch-handler or music specifier classes, but not their instances. Materials comprise instances of classes representing *out-of-time* musical constructs, such as pitch segments or music specifiers which have not yet been deployed along the timeline of the score. Segments comprise configured instances of the score's segment-maker class which maquette together previously-defined materials *in-time* into a musical chronology. Each of these three categories of objects has its own directory in the directory structure outlined here: `makers/`, `materials/` and `segments/`. Additionally, materials and segments may be illustrable. Any associated illustration assets for those code objects is necessarily grouped together with its originating Python modules.

The document preparation process also involves a number of different types of assets and tasks. Any LilyPond-typeset PDF created during the composition of a score, such as a material or segment illustration, or even the final score itself likely requires a corresponding score-specific stylesheet containing the typographic overrides and LilyPond context definitions pertinent to that score. These stylesheets reside in the top-level `stylesheets/` directory. Scores composed with Abjad and Consort and typeset with LilyPond and LaTeX involve potentially many source files: LilyPond sources for the musical content, yet more LilyPond sources for concatenating and styling that content, LaTeX sources for cover pages, prefaces and performance notes, and still more LaTeX sources for concatenating the PDFs created while typesetting various other sources. These source files, in their various stages of typesetting, occupy the `build/` directory. Finally, the finished documents, ready to be delivered to ensembles, occupy the `distribution/` directory.

### 5.1.1 MAKERS

Each score's `makers/` directory houses classes specific to that score, with each class stored on disk in its own Python module of the same name. In the case of my three *Invisible Cities* scores, these makers always comprise subclasses of Consort's `SegmentMaker` and `ScoreTemplate` classes. The segment-maker subclasses effectively pre-load common information about the score, such as what time signatures to permit by default or what the end-of-score markup

should look like. They may also define a considerable amount of additional score post-processing, as in the case of *Armilla*, where many passes of voice copying were required to create the typography for the bowing staves. A score’s score template class necessarily defines the score’s instrumentation and context hierarchy, giving the order and grouping of each performer’s staff. As none of the scores in this dissertation make use of common instrumentations, each requires that a completely new score template be defined.

Any other class definitions required by the score package should also be placed into the `makers/` directory. For example, *Plague Water*, whose directory structure is effectively identical to the *Invisible Cities* series but whose composition model both predated and heavily informed the development of Consort, houses nearly thirty separate class definitions in its `makers/` directory.

### 5.1.2 MATERIAL PACKAGES

Materials represent *out-of-time* musical objects, such as pitch collections, rhythm-makers, and even fully configured music specifiers, which may appear at some point in the time-line of a piece. They are implemented as importable Python packages, grouped flatly into the score package’s top-level `materials/` directory, itself a Python package due to the presence of an initializer file. Like the `makers` modules, material definition module define only a single object.

```
materials/
├── __init__.py ..... The subpackage initializer.
├── abbreviations/ .. A material package.
├── dense_timespan_maker/ .. Another material package.
├── guitar_agitato_music_specifier/ .. Yet another material package.
├── guitar_strummed_music_specifier/
├── guitar_tremolo_music_specifier/
├── guitar_undulation_tremolo_music_specifier/
├── percussion_bamboo_windchimes_music_specifier/
├── percussion_crotales_flash_music_specifier/
├── percussion_crotales_interruption_music_specifier/
└── ... ..... Many more material packages.
```

FIGURE 5.2: Overview of *Ersilia*’s `materials/` directory.

Each material package contains at the least its own initializer as well as a `definition.py` file consisting of Python code which defines or configures that material. Crucially, the actual code object expressed in that `definition.py` should be named after the material package itself. This simplifies the Python import process a great deal. Simple importing utilities can then be written to iterate over every directory within the `materials/` directory, producing corresponding Python import commands of the form `from my_material.definition import my_material`. Automatically importing the objects defined in each material package into the `materials` namespace allows those objects to

be easily referenced from within each segment definition:

```
>>> materials_names = dir(ersilia.materials)
>>> materials_names = [x for x in materials_names if not x.startswith('_')]
>>> print(materials_names)
['abbreviations', 'dense_timespan_maker', 'guitar_agitato_music_specifier',
'guitar_continuo_music_specifier', 'guitar_pointillist_harmonics_music_specifier',
'guitar_strummed_music_specifier', 'guitar_tremolo_music_specifier',
'guitar_undulation_tremolo_music_specifier', 'percussion_bamboo_windchimes_music_specifier',
'percussion_crotales_flash_music_specifier', 'percussion_crotales_interruption_music_specifier',
'percussion_low_pedal_music_specifier', 'percussion_marimba_agitato_music_specifier',
'percussion_marimba_ostinato_music_specifier', 'percussion_marimba_tremolo_music_specifier',
'percussion_snare_interruption_music_specifier', 'percussion_temple_block_fanfare_music_specifier',
'percussion_tom_fanfare_music_specifier', 'permitted_time_signatures', 'piano_agitato_music_specifier',
'piano_arm_cluster_music_specifier', 'piano_glissando_music_specifier',
'piano_palm_cluster_music_specifier', 'piano_pedals_music_setting', 'piano_pointillist_music_specifier',
'piano_string_glissando_music_specifier', 'piano_tremolo_music_specifier', 'pitch_pipe_music_specifier',
'saxophone_agitato_music_specifier', 'shaker_decelerando_music_specifier',
'shaker_spasmodic_music_specifier', 'shaker_tremolo_music_specifier', 'sparse_timespan_maker',
'string_agitato_music_specifier', 'string_legato_music_specifier', 'string_low_pedal_music_specifier',
'string_ostinato_music_specifier', 'string_overpressure_music_specifier',
'string_pointillist_music_specifier', 'string_tremolo_music_specifier', 'sustained_timespan_maker',
'tutti_timespan_maker', 'wind_agitato_music_specifier', 'wind_continuo_music_specifier',
'wind_low_pedal_music_specifier', 'wind_ostinato_music_specifier', 'wind_pointillist_music_specifier',
'wind_tremolo_music_specifier']
```

```
guitar_tremolo_music_specifier/
├── __init__.py ..... The material package's Python initializer.
├── definition.py ..... The material's definition file.
└── illustration.ly ..... The material illustration's LilyPond source.
└── illustration.pdf ..... The material's rendered illustration.
```

FIGURE 5.3: Directory listing of *Ersilia*'s `guitar_tremolo_music_specifier` material package.

Materials – depending on their type – may be illustrable, as described in section 2.1. Any generated illustration assets – comprising a LilyPond input file and its resulting PDF – are stored in the material package alongside their originating definition module. The means by which these illustration files come to reside there is elaborated on in subsection 5.2.2.

### 5.1.3 SEGMENT PACKAGES

Segments, like materials, are implemented as Python packages, grouped together into the top-level `segments`/ directory. Each segment package can be imported, has an obligatory Python initializer, as well as a definition module containing the definition of that segment's segment-maker. Because Consort's segment-maker class is illustrable, each segment package eventually houses that segment's illustration LilyPond source and output PDF.

```

segments/
├── __init__.py ..... The subpackage initializer.
├── __metadata__.py ..... Metadata about the order of segments in the score.
└── chemish/ ..... A segment package.
    ├── __init__.py ..... The segment package's Python initializer.
    ├── __metadata__.py ..... Auto-generated metadata about this score segment.
    ├── definition.py ..... The segment's definition file, containing a configured segment-maker.
    ├── illustration.ly ..... The segment illustration's LilyPond source.
    └── illustration.pdf ..... The segment's rendered illustration.
        cut_1/ ..... Another segment package.
        └── ...
        komokome/ ..... Another segment package.
        └── ...
        cut_2/ ..... Another segment package.
        └── ...
        sort/ ..... Yet another segment package.
        └── ...

```

FIGURE 5.4: Overview of *Ersilia*'s `segments/` directory.

Each segment package may also house a `__metadata__.py` module, which stores information about that segment such as the number of measures it contains, the final time signature and tempo, and so forth. This information is generated automatically during the segment-maker's interpretation, and allows other segment-makers in other segment packages to draw conclusions about their own segment's context in the full score without requiring the re-interpretation of any other segments. Like the illustration sources, these are discussed in more depth in subsection 5.2.2.

A `__metadata__.py` module sibling to each segment package simply defines the order of segments. For example, the order of segments in *Ersilia* is `komokome`, `cut_1`, `sort`, `cut_2`, then `chemish`. Such an order must be declared explicitly as it cannot be ascertained from the lexical ordering of the names of the segment packages. The `segments/`-local metadata module allows for such an explicit ordering.

#### 5.1.4 THE `stylesheets/` DIRECTORY

The `stylesheets/` directory consists of LilyPond files containing typographic overrides, context definitions, document header markup, page layout configuration and Scheme function definitions. The file `stylesheet.ily` represents the primary stylesheet for the entire score, and contains most of the typographic customization. However, LilyPond stylesheets can *cascade*. Multiple stylesheets can be included into the same master score file with definitions in subsequently included stylesheets overriding those in earlier ones. Likewise, stylesheets – because they are simply LilyPond files – can be included directly into one another.

For example, in *Ersilia*, the master stylesheet file `stylesheet.ily` directly includes Scheme definitions from the file `scheme.ily`. That master stylesheet is included into the interpreted LilyPond source of every segment. However, segments beyond the first also include the stylesheet file `nonfirst-segment.ily` which suppresses the appearance of titles and other headers at the top of those segment illustrations, giving the impression when viewing those segments' illustrations alone that one has jumped into the middle of the full score.

```
stylesheets/
└── nonfirst-segment.ily ..... Style information for segments after the first segment.
└── parts-landscape.ily ..... Style information for landscape-orientation parts.
└── parts-portrait.ily ..... Style information for portrait-orientation parts.
└── scheme.ily ..... LilyPond Scheme commands to be included in the primary stylesheet.
└── stylesheet.ily ..... The primary stylesheet.
```

FIGURE 5.5: Directory listing of *Ersilia*'s `stylesheets/` directory.

Because LilyPond supports the inclusion of one LilyPond source file into another, it is possible to separate the musical content of a work from the settings defining its typographic presentation or page layout. This is the approach I have taken, out of necessity, when building scores with Abjad and Consort. It is certainly possible to define every one of the typographic overrides found in `stylesheet.ily` in Abjad via its top-level `override()` and `set_()` functions. However, typesetting the illustration source of a segment in LilyPond generally takes less time than for that segment to be interpreted in Python. By separating out typographic overwrites from the musical “content” it becomes easier to iterate over refining those overrides by simply adjusting the stylesheet by hand and recompiling the already-interpreted segment illustration sources.

### 5.1.5 THE `build/` DIRECTORY

The `build/` directory, outlined in figure 5.6, holds files pertinent to building scores and parts, along with any component documents, including front and back covers, prefaces or performance notes. The contents of the `build/` directory are organized into *document targets* and *assets*. Each document target subdirectory consists of source files for producing scores and parts in a particular format, such as tabloid or A4 paper, or for a particular performance or language translation. Assets include any LilyPond and LaTeX file consisting of textual or musical content to be included into a document target. For example, segment illustration LilyPond sources are housed in `build/segments/` while LaTeX includes – containing blocks of prose to be flowed into a preface – are housed in `build/assets/`. Source files contained within document target subdirectories can generally be typeset successfully in order to produce either a finished document or a component of a finished document, such as a cover page. Asset source files, in

contrast, must always be included or otherwise combined with other source files in order to produce valid typeset-table input.

```

build/
└── 11x17-landscape/ ..... A document build target directory.
    └── ...
    └── 11x17-portrait/ ..... Another document build target directory.
        └── ...
    └── assets/ ..... LaTeX files to be included into each preface layout.
        ├── calvino.tex
        ├── instrumentation.tex
        ├── leguin.tex
        └── performance-notes.tex
    └── legal-landscape/ ..... Another document build target directory.
        └── ...
    └── legal-portrait/ ..... Yet another document build target directory.
        └── ...
    └── parts.ily ..... A LilyPond include file for generating parts.
    └── segments/ ..... Segment illustration LilyPond sources.
        ├── chemish.ily
        ├── cut-1.ily
        ├── cut-2.ily
        ├── komokome.ily
        └── sort.ily
    └── segments.ily ..... A LilyPond include file giving the order of the segments to concatenate.

```

FIGURE 5.6: Overview of *Ersilia*'s build/ directory.

The build/ directory also contains two includable files sibling to the document target, assets/ and segments/ subdirectories. The first file, parts.ily, defines LilyPond commands for outputting parts. This file is discussed further in subsection 5.2.5. The second file, segments.ily, defines in LilyPond syntax the order in which the segment illustration sources, collected in the segments/ subdirectory of the build/ directory, are to be concatenated.

```

11x17-landscape/ ..... A build target directory.
└── Makefile ..... A Makefile for GNU make affords various build tasks.
└── back-cover.pdf ..... PDF output of the back cover LaTeX source.
└── back-cover.tex ..... LaTeX source for the back cover.
└── front-cover.pdf ..... PDF output of the front cover LaTeX source.
└── front-cover.tex ..... LaTeX source for the back cover.
└── music.ly ..... LilyPond source for the concatenated score segments.
└── music.pdf ..... PDF output for the concatenated score segments LilyPond source.
└── parts.ly ..... LilyPond source for generating individual parts PDFs.
└── preface.pdf ..... PDF output for the preface LaTeX source.
└── preface.tex ..... LaTeX source for the preface.
└── score.pdf ..... PDF output of the complete score LaTeX source.
└── score.tex ..... LaTeX source for the complete score.

```

FIGURE 5.7: Directory listing of a document build target in *Ersilia*.

Each document target directory consists of a similar collection of files: LaTeX sources for the front cover, back

cover, and preface, LilyPond sources for the musical content of the score itself and parts, and a master LaTeX source which combines covers, the preface and the music into a single score PDF. The LilyPond `music.ily` contains \include statements which pull in the `segments.ily` file from the build/ directory as well as the score's main LilyPond stylesheet from the top-level `stylesheets/` directory. A Makefile, which defines build commands for the GNU make command-line utility, can also help automate various document target tasks, such as recompiling all LaTeX documents in the correct order to produce a finished score.

### 5.1.6 THE etc/ AND distribution/ DIRECTORIES

The `etc/` and `distribution/` directories are perhaps the simplest. The former holds any notes, plans or to-do lists pertinent to the compositional process of the score while the latter collects completed scores and parts for each document target from the build/ directory. While note-keeping can and does happen on paper, away from the keyboard, designating a space within the score package for such miscellaneous documents allows those notes and plans to be tracked by the score package's *version control system*, discussed in subsection 5.3.2.

### 5.1.7 THE test/ DIRECTORY

Finally, the `test/` directory contains Python modules defining parameterized tests which verify the stability of each material and segment package. The material tests simply attempt to import and evaluate the storage format of each object defined in a material package. The classes of these objects generally have much more extensive test suites already defined either in Abjad or Consort, so simply instantiating and formatting them here suffices.

```
test/
└── test_materials.py ..... Parameterized tests for validating integrity of each material package.
    └── test_segments.py ..... Parameterized tests for validating integrity of each segment package.
```

FIGURE 5.8: Parameterized tests.

Segment tests attempt to both interpret each segment's segment-maker, and also typeset the resulting LilyPond input file. Note that these tests do not attempt to guarantee that a particular segment produces some score exactly matching a target, but simply that the segment-maker manages to interpret without failure, and that LilyPond manages to typeset the resulting source without error.

**TODO:** Discuss segment tests as system tests.

### 5.1.8 PYTHON PACKAGING

Each score package should also be properly *packaged* according to Python standards so that it can be installed on other systems. This might strike composers as an unmotivated suggestion. Why should one structure their private score such that it can be used by others? Making a score installable in this way affords a number of conveniences related to testing. Installable scores can be tested in *virtual environments* – a common Python practice – by automated test runners like `tox`<sup>1</sup> or run on remote continuous-integration testing services such as Travis-CI.<sup>2</sup>

```
ersilia/
├── .git/..... The Git repository history.
├── .gitignore..... File patterns to be ignored by the Git version control system.
├── .travis.yml..... The Travis-CI build configuration script.
├── README.md. A MarkDown text file containing introductory information about the score package.
├── ersilia/..... The score package itself.
├── requirements.txt..... Dependency information, for use when installing on Travis-CI.
├── setup.cfg..... Python packaging configuration.
├── setup.py..... The Python package installation script.
└── tox.ini..... Configuration for the tox automated testing tool.
```

FIGURE 5.9: Overview of *Ersilia*'s Python packaging assets.

For those who wish to make both their scores and code completely public – as I have – installation affordances are simply necessary for letting others explore the code as quickly and easily as possible.

## 5.2 DOCUMENT PREPARATION IN DETAIL

As with working with Abjad in the simplest case, the work flow of composing and preparing scores with Consort continues to revolve around a cycle of defining musical structures in text, illustrating that music visually, then refining the textual definitions. Unlike creating simple musical examples at the command-line, as demonstrated throughout this dissertation – especially in chapter 2 –, managing a large-scale score requires considerably more tools and many more file-system assets. The following sections discuss some of the additional complexities involved in organizing and typesetting a large-scale score computationally.

### 5.2.1 BUILD TOOLS

**TODO:** What is the point of this? Warning: magic ahead.

<sup>1</sup><https://pypi.python.org/pypi/tox>

<sup>2</sup><https://travis-ci.org/>

I make use of a variety of custom build tools for managing assets within a score package which will not be discussed here in any detail as they are still rather provisional. These tools simplify various tasks such as creating new segment and material packages, executing segment definition modules in order to illustrate the contained segment-makers, comparing new illustration LilyPond sources and PDF outputs against previously rendered ones, and collecting segment LilyPond sources from each segment package into the build directory for document preparation. Simply put, build tools for score packages streamline the problems of moving files and folders into their appropriate locations, executing Python modules, persisting code objects to disk and cleaning up after any transient files.

### 5.2.2 ILLUSTRATING & PERSISTING SEGMENTS

All of the examples of object illustration demonstrated throughout this dissertation involve illustrating objects in a live Python interpreter session. However, in order to construct a score made of potentially many concatenated segment illustrations, those illustrations must be persisted to disk. While persistence can certainly also be handled in a live interpreter session, by-hand illustration and persistence are both error-prone and tedious. One way to simplify the task of illustrating segments and persisting their illustrations to disk is by adding executable code to the end of each segment definition module. When run by Python as a script, those segment definition modules can be instructed to illustrate their segment makers and persist the resulting illustration and segment metadata to disk in the same directory as the segment definition module. Consider the following trivial segment definition module, complete with an executable suite at its end:

```
import os
import consort
from abjad import persist

segment_maker = consort.SegmentMaker(
    desired_duration_in_seconds=4,
)

if __name__ == '__main__':
    illustration, metadata = segment_maker()
    directory_path = os.path.dirname(os.path.abspath(__file__))
    illustration_pdf_file_path = os.path.join(directory_path, 'illustration.pdf')
    metadata_file_path = os.path.join(directory_path, '__metadata__.py')
    persist(illustration).as_pdf(
        pdf_file_path=illustration_pdf_file_path,
        candidacy=True,
    )
    persist(metadata).as_module(
        module_file_path=metadata_file_path,
```

```
    object_name='metadata',
)
```

When run by Python as a script – via a command like `python my_segment_definition.py` – rather than imported, the code in the segment definition module executes in the module namespace `__main__`. The conditional `if __name__ == '__main__'`: guarantees then that the suite beneath that conditional executes if and only if the current module name is `__main__`, again because the segment definition module was run as a script. This crucially prevents illustration code from running when the definition module is simply imported rather than executed. However, if the suite underneath the conditional *does* execute, it first calls the segment-maker defined earlier in that module. That segment-maker then returns both an illustration and a segment metadata dictionary. Next, file paths relative to the segment module's file path are determined in order to persist the just-created illustration and metadata. The global variable `__file__` in any Python code module gives the location of that module on the file system, from which that module's directory can be determined. Finally, calls to Abjad's top-level `persist()` function against the illustration and metadata expose persistence agent instances which afford persisting each object to disk as a LilyPond file and Python module respectively. The `candidacy=True` keyword argument to `persist{illustration}.as_pdf(...)` checks whether a PDF already exists and only overwrites if the new would differ.

As mentioned in subsection 5.2.1, I make use of custom build tools for my scores which afford a more elaborate version of the illustration and persistence task outlined above. One such elaboration passes the metadata for the previous segment – if such metadata exists – to the current segment-maker's `__call__()` method. Recall that the previous segment can be determined by consulting the `__metadata__.py` module sibling to the segment packages, as outlined in subsection 5.1.3. A modified version of the current segment-maker's metadata is also passed as an argument at call-time, and includes both a count of the total number of segments and the current segment's index within those segments. Such metadata allows a segment-maker to automatically determine if it is the first or last of all segments, as well as to take into account any pertinent settings effective at the end of the previous segment, such as the previous segment's ending tempo or time signature.

Note too that persisting material illustrations can be handled in a nearly identical fashion. And unlike segments, they do not need to even optionally consult or persist metadata.

### 5.2.3 COLLECTING AND CONCATENATING SEGMENT ILLUSTRATIONS

The model of composition afforded by Consort assumes scores consist of multiple segments, each of which is persisted as a LilyPond score context based on an identical score template. This assumption relies on LilyPond's ability

to concatenate like-named contexts. Consider the following LilyPond expression:

```
{\context Score = "The Score" <<
    \context Staff = "Staff A" { c'1 d'1 }
    \context Staff = "Staff B" { c'1 b1 }
>>
\context Score = "The Score" <<
    \context Staff = "Staff A" { e'1 f'1 }
    \context Staff = "Staff B" { a1 g1 }
>>
\context Score = "The Score" <<
    \context Staff = "Staff A" { g'1 a'1 }
    \context Staff = "Staff B" { f1 e1 }
>>
}
```

The above example contains three like-named scores – each named “The Score” – grouped by a pair of outer braces into a single music expression. Because the scores have identical names and the staff contexts within them are also identically named – “Staff A” and “Staff B” –, LilyPond concatenates each like-named context in each score together.

The resulting music expression is equivalent to the following:

```
\context Score = "The Score" <<
    \context Staff = "Staff A" { c'1 d'1 e'1 f'1 g'1 a'1 }
    \context Staff = "Staff B" { c'1 b1 a1 g1 f1 e1 }
>>
```

An identical context concatenation process is used to fuse the scores defined in each segment’s illustration file into a single music expression. First however, those segment illustrations must be collected into the `segments/` directory within the `build/` directory and their sources massaged to permit concatenation.

Each segment illustration, as it exists in its segment package, represents a complete, fully-typesettable LilyPond file, consisting of a LilyPond version statement, pitch-name language command, various stylesheet include commands, and that segment’s score’s context block wrapped within a `\score` block:

```
\version "2.19.17"
\language "english"

#(ly:set-option 'relative-includes #t)

\include "../../stylesheets/stylesheet.ily"

\score {
    \context Score = "The Score" <<
        ...
    >>
}
```

Such an input cannot simply be included along with the other segment illustration files to create a concatenated score. In fact, a construction like the following, with \score blocks nested within other \score blocks, is considered a syntax error by LilyPond:

```
\version "2.19.17"

\score {
    {
        \version "2.19.17"
        \score {
            \context Score = "The Score" << c'1 >>
        }
        \version "2.19.17"
        \score {
            \context Score = "The Score" << c'1 >>
        }
        \version "2.19.17"
        \score {
            \context Score = "The Score" << c'1 >>
        }
    }
}
```

Instead, some simple string processing must be applied against the illustration file to trim out all content besides the inner score \context block, resulting in a much thinner but includable construction.

```
\context Score = "The Score" <<
...
>>
```

This trimming removes unnecessary header and styling information from each collected segment illustration and allows LilyPond to concatenate the score contexts together into a single score structure.

Segment collection then involves copying each segment illustration LilyPond source from its segment package into the segments/ subdirectory of the score package's build/ directory, naming that copied illustration source after its originating segment package to differentiate it from the other collected segment sources, and finally trimming that source as shown above. As with segment illustration automation, I make use of some provisional build tools which simplify this process. However, a naive approach to the task of collecting and trimming segment illustration could also be implemented in a few dozen lines of Python code. With the segment illustrations collected into the build/ directory, the build/ directory's segments.ily LilyPond include file can be updated based on the segment order encoded in the segments/ directory's \_\_metadata\_\_.py module. For example, *Ersilia*'s segments.ily include file ultimately looks like this:

```
{
  \include "segments/komokome.ly"
  \include "segments/cut-1.ly"
  \include "segments/sort.ly"
  \include "segments/cut-2.ly"
  \include "segments/chemish.ly"
}
```

Finally, the updated `segments.ily` file can be included directly into a document target's `music.ly` LilyPond source file, to create the complete musical contents in the appropriate layout for that document target's paper output format. The LilyPond source for a document target's musical content, formatted for legal-size paper in portrait orientation, might then simply look like the following:

```
\version "2.19.17"
\language "english"

#(ly:set-option 'relative-includes #t)
\include "../../stylesheets/stylesheet.ily"
#(set-default-paper-size "legal" 'portrait)
#(set-global-staff-size 11)

\score {
  \include "../segments.ily"
}
```

## 5.2.4 ORGANIZING AND TYPESETTING LATEX ASSETS

**TODO:** Improve motivation in opening paragraph.

In the spirit of LilyPond's automated musical typesetting, I have chosen to rely on LilyPond's spiritual predecessor, LaTeX, for handling all purely-textual or otherwise utilitarian typesetting in the documents I produce. LaTeX, like LilyPond, takes a textual source file as input, consisting of a variety of commands which describe the structure and content of a document, and produces a typeset target, generally as a PDF. Each document target of course requires more content than simply the musical meat of the score. Title pages, covers, prefaces and performance notes also require typesetting and should be handled as simply as possible while maintaining identical content, output formats, fonts and spacing.

The most complex LaTeX task in my document preparation process involves formatting identical textual content for different paper sizes and orientations, as with each document target's preface information. A preface containing multiple sections, each potentially containing nested lists of instruments or diagrams, may not format equally well in all desired paper dimensions. For example, the preface to *Armilla* contains six different sections

of widely divergent lengths, but fits well in portrait orientations of tabloid and legal paper when split into two columns. However, when formatted on letter paper, or in landscape orientation, sections of prose no longer work in the two column layout cleanly and need to be individually resized or flowed into additional columns. Formatting such a preface properly in both landscape and portrait orientations, in tabloid, letter and legal paper sizes requires non-trivial rearrangement and must be adjusted by hand. However, care must be taken not to duplicate content across different LaTeX source files. Duplicated prose inevitably goes out of sync, creating a tremendous cognitive burden on the composer as they prepare their documents.

This problem – needing to provide manually-tweaked alternate layouts for prose while discouraging by-hand copying – can be solved by separating out each section of text in the preface into separate includable LaTeX files, stored in the assets/ subdirectory of the build/ directory, and then including those files back into each alternative preface layout file. Such a workflow, like the use of distinct stylesheet and score files in LilyPond, separates content from layout.

Consider the preface to *Ersilia*, which contains four separate sections: two quotes, from Italo Calvino and Ursula K. Le Guin, and two passages on instrumentation and performance practice. Each of these sections is stored as a separate LaTeX file in the assets/ subdirectory of *Ersilia*'s build/ directory. When preparing prefaces for the tabloid landscape edition and the legal portrait edition, each collection of prose assets is flowed into different text grids via the `textpos` LaTeX package. For example, the tabloid landscape uses four separate columns, each of varying widths:

```
\begin{document}

\begin{textblock}{31}(0, 0)
    \center\huge\textbf{PREFACE}
\end{textblock}

\begin{textblock}{6}(0, 2)
    \subimport{../assets/calvino}{}
\end{textblock}

\begin{textblock}{7}(7, 2)
    \subimport{../assets/leguin}{}
\end{textblock}

\begin{textblock}{4}(15, 2)
    \subimport{../assets/instrumentation}{}
\end{textblock}

\begin{textblock}{11}(20, 2)
    \subimport{../assets/performance-notes}{}
\end{textblock}
```

```
\end{document}
```

The legal portrait version uses two columns, with the quotes on the left side and the performance instructions on the right:

```
\begin{document}

\begin{textblock}{23}(0, 0)
    \center\huge\textbf{PREFACE}
\end{textblock}

\begin{textblock}{11}(0, 2)
    \subimport{../assets/calvino}{}
    \subimport{../assets/leguin}{}
\end{textblock}

\begin{textblock}{11}(12, 2)
    \subimport{../assets/instrumentation}{}
    \subimport{../assets/performance-notes}{}
\end{textblock}

\end{document}
```

In both cases, none of the actual content of the prose appears in the LaTeX files defining the layout of the prose.

This guarantees that any edits or corrections to the prose will be reflected equally in all layouts of the preface.

LaTeX can also be used to join documents together. Each document target contains a master `score.tex` source file which contains commands for combining all other PDF components of the score together into a single PDF via LaTeX, obviating the need to use any other PDF-merging tool. For example, the 11x17 portrait `score.tex` for *Ersilia* looks like this:

```
\documentclass{article}
\usepackage[papersize={11in, 17in}]{geometry}
\usepackage{pdfpages}
\begin{document}

\includepdf[pages=-]{ersilia-11x17-portrait-front-cover.pdf}
\includepdf[pages=-]{ersilia-11x17-portrait-preface.pdf}
\includepdf[pages=-]{ersilia-11x17-portrait-music.pdf}
\includepdf[pages=-]{ersilia-11x17-portrait-back-cover.pdf}

\end{document}
```

### 5.2.5 PART EXTRACTION

When working with LilyPond and Abjad, part extraction relies on two fairly simple LilyPond mechanisms: *tags* and *book blocks*. In LilyPond, any music expression can be labeled with a tag, a string or symbol which identifies that music expression. Later, music expressions can be filtered to either remove or solely preserve any expression with a given tag. Consider the following LilyPond pseudo-code, consisting of a score containing three staves and a time signature context:

```
\keepWithTag #'(time B)
\new Score <<
  \tag #'time \new TimeSignatureContext = "Time Signature Context" { ... }
  \tag #'A \new Staff = "Staff A" { ... }
  \tag #'B \new Staff = "Staff B" { ... }
  \tag #'C \new Staff = "Staff C" { ... }
>>
```

Each context contained by the Score context – the three staves and the time signature context – is tagged via a LilyPond `\tag` command, associating that context's music expression with the tag's symbol. The command preceding the score itself – `\keepWithTag #'(time B)` – indicates that the score should filter out any tagged music expression which do not belong to the list of expressions `#'(time B)`. That is, the score should omit the two staves tagged A and C, effectively producing a score structured like so:

```
\new Score <<
  \new TimeSignatureContext = "Time Signature Context" { ... }
  \new Staff = "Staff B" { ... }
>>
```

This tagging technique is used in every score developed with Consort. Each score's score template includes tag commands labeling both the time signature context – which must appear both in the full score and all parts as it contains time signature, tempo and rehearsal mark information – as well as all inner contexts necessary for individual performers. By constructing the appropriate `\keepWithTag` commands, music expressions representing each performer's part can be constructed easily. Note the use of `\tag` commands throughout the structure of *Armilla*'s score:

```
\context Score = "Armilla Score" <<
  \tag #'time
  \context TimeSignatureContext = "Time Signature Context" {
  }
  \tag #'viola-1
```

```

\context StringPerformerGroup = "Viola 1 Performer Group" \with {
    instrumentName = \markup {
        \hcenter-in
        #10
        "Viola 1"
    }
    shortInstrumentName = \markup {
        \hcenter-in
        #10
        "Va. 1"
    }
} <<
\context BowingStaff = "Viola 1 Bowing Staff" {
    \clef "percussion"
    \context Voice = "Viola 1 Bowing Voice" {
    }
}
\context FingeringStaff = "Viola 1 Fingering Staff" {
    \clef "alto"
    \context Voice = "Viola 1 Fingering Voice" {
    }
}
>>
\tag #'viola-2
\context StringPerformerGroup = "Viola 2 Performer Group" \with {
    instrumentName = \markup {
        \hcenter-in
        #10
        "Viola 2"
    }
    shortInstrumentName = \markup {
        \hcenter-in
        #10
        "Va. 2"
    }
} <<
\context BowingStaff = "Viola 2 Bowing Staff" {
    \clef "percussion"
    \context Voice = "Viola 2 Bowing Voice" {
    }
}
\context FingeringStaff = "Viola 2 Fingering Staff" {
    \clef "alto"
    \context Voice = "Viola 2 Fingering Voice" {
    }
}
>>
>>

```

The time signature context receives its own tag, and the staff groups wrapping each performer's bowing and fingering staves are also tagged appropriately.

While tagging allows for extracting parts as music expressions, it does not yet result in actual documents for each part. LilyPond's \book block structure, combined with the \bookOutputSuffix command, provide a concise mechanism for generating multiple output PDFs from a single LilyPond input file. As demonstrated throughout this document, LilyPond files are structured into blocks: context blocks, score blocks, header and paper blocks, and so forth. The highest level block is a \book block. Somewhat like parts in LaTeX, book blocks separate content from one another by page breaks. Score blocks contained in separate book blocks are guaranteed to never appear on the same page together. Furthermore, by specifying a book output suffix within each book block, LilyPond will not simply separate that book block's content by page breaks but will actually output a wholly separate PDF, whose filename is suffixed with that book's output suffix.

Consider this excerpt from *Ersilia*'s `parts.ily` file:

```
\book {
    \bookOutputSuffix "cello"
    \score {
        \keepWithTag #'(time cello)
        \include "../segments.ily"
    }
}

\book {
    \bookOutputSuffix "clarinet"
    \score {
        \keepWithTag #'(time clarinet)
        \include "../segments.ily"
    }
}

\book {
    \bookOutputSuffix "contrabass"
    \score {
        \keepWithTag #'(time contrabass)
        \include "../segments.ily"
    }
}
```

This file contains one book block per instrument. Each book block specifies an output suffix pertinent to a specific performer in the ensemble. Each book block also contains a score block whose contents consist entirely of an include statement – pointing at `segments.ily`, which concatenates all score segments into a single music expression – wrapped in a `\keepWithTag` command which filters out everything except that performer's musical content and the global time signature context.

With these techniques in mind, a complete parts-extraction file would look like the following, from *Ersilia*:

```

\version "2.19.17"
\language "english"

#(ly:set-option 'relative-includes #t)
\include "../../stylesheets/stylesheet.ily"
#(set-default-paper-size "11x17" 'landscape)
\include "../../stylesheets/parts-landscape.ily"
\include "../parts.ily"

```

Beyond the initial LilyPond boilerplate of specifying a LilyPond version and pitch-name input language, this part extraction file simply consists of an include for the global stylesheet, a page-layout command, an include for a stylesheet for landscape parts and finally an include for the global parts definition file, `parts.ily`. When interpreted by LilyPond, the above will generate one 11x17 landscape PDF per book block defined in `parts.ily`.

Note that there are methods by which composers can generate parts in LilyPond. Most LilyPond users who work strictly with LilyPond, writing “by hand”, would likely places each of the instrumental parts in the score into a separate file or variable. Those parts would then be combined into either the full score or a part for a single player as necessary. Because Consort produces a single score, complete with all parts joined into a single expression, filtering must be used to “strip” the score down to the desired musical elements.

## 5.3 PROJECT MAINTENANCE

When one composes with code, one necessarily acts as a software developer. Therefore, scores undertaken in this fashion benefit not only from the techniques laid out earlier in this chapter, but also from those practiced daily by developers working in disciplines beyond music. Such universal techniques include version control and testing, both means of managing the stability and complexity of projects as they grow and change.

### 5.3.1 AUTOMATED REGRESSION TESTING

Regression testing<sup>6</sup> examines the stability and correctness of a software system during the course of development, allowing the software’s authors to verify that changes and revisions to that system have not introduced errors or unexpected behavior. Rather than testing the system manually, software authors typically write *automated regression testing batteries*: collections of tests implemented as functions or classes which can be run automatically by a testing tool – a *test runner* –, returning the results to the authors as a report. Testing philosophies and practices in the open source community are now very diverse and sophisticated, therefore a full discussion of software testing is beyond the scope of this document. Nevertheless, I believe that testing is crucial to the development of any software system and therefore to any score or composition model implemented in code.

Both Abjad, Consort and the various scores implemented with these packages make extensive use of tests. Such tests take a variety of forms: *documentation tests*, *unit tests*, *system tests* and *parameterized tests*. Documentation testing, or “doctesting” in Python parlance, verifies that the code examples in the documentation strings accompanying classes, methods and functions are correct. Unit testing examines small fragments of code, such as individual functions, class initializers or methods, passing in a variety of input and examining the output for correctness. System testing verifies that an entire software system functions as expected. In the context of Consort, auditing the illustration produced by a fully configured segment-maker would constitute a system test as such an operation touches upon nearly every class defined in Consort’s library. Additionally, parameterized tests provide a means of applying a single test against a variety of input. This is used extensively in Abjad and Consort to guarantee that every class in each system can be instantiated, is fully documented, can be represented as a string, can be hashed and so forth. Rather than write over 900 hundred separate tests – one for each class in Abjad – all classes can be collected in a list and passed to a single parameterized test function, which then runs itself against each class as though that were separate test.

The three scores I implemented with Consort each contain two parameterized tests in their top-level test/ directory. One test verifies that the objects defined in each material definitions are valid and contain no errors. The second parameterized test illustrates each segment definition, failing not only if the segment-maker is unable to interpret itself but also if LilyPond fails to typeset the resulting illustration source. These per-score tests serve a number of functions. They allow me to ascertain that the logic implemented in the score itself, in Consort, in Abjad and in any dependency of these projects continues to interoperate – at least non-catastrophically. They also allow me verify that the LilyPond output produced by the score’s segment-makers – along with any stylesheet information defined in the score package – is still valid LilyPond source code, and has not been deprecated by newer versions of LilyPond.

All of these types of testing combined act as a kind bulwark against backsliding both during the development of a project, in the moment, and afterward when maintaining the longevity of prior work against obsolescence.

### 5.3.2 VERSION CONTROL

Version control systems<sup>11</sup> – or *VCS* – record changes to sets of files, tracking additions, deletions, name changes and content modifications, generally on a line-by-line basis. Changes are grouped together into bundles called *commits*, labeled with a timestamp and *commit message* elaborating on the purpose or contents of the changes. The graph of

commits made against a project as recorded by the project’s VCS, where each commit is connected to some previous parent commit, constitute the *history* of that project. While anecdotally uncommon amongst composers, virtually all software developers make use of version control to greater or lesser degrees, for a wide variety of reasons.

Version control provides the most compact and legible way of archiving multiple versions of a project. When making changes, instead of duplicating the directory containing one’s work, giving the copied directory a name like “MyProject-year-month-day” or “MyProject-version-3” and then editing the contents of that copy, one simply works in the same directory as always, committing each change to the version control system’s history. At any point and for any reason, any previously committed version of the project can be recalled. Reverting to an earlier version of the project does not destroy later versions which, because they have also been recorded in the project’s history, can be returned to any point. Most version control systems store their history in hidden files or directories within the project. For example, every project I work uses the VCS git<sup>3</sup>, originally authored by Linus Torvalds, the chief architect of the Linux kernel. Git stores its history in a hidden directory named .git in the root of the versioned project, as indicated in subsection 5.1.8.

Many version control systems support *tagging*, the labeling of a commit in the version history as particularly important. Software projects often tag commits intended for release to the public with version numbers like 1.0, 2.5 or even v1.8.5-rc3. Likewise, important commits in the history of a score’s development might be tagged similarly. For example, the version of the project when the score was first sent to an ensemble might be tagged world-premier. Tagging allows the composer to continue revising the score while still always being able to refer back to milestones in the score’s history.

Version control systems make it easier to understand how and why a project was changed. Not only does each commit have a commit message, allowing the author to provide some explanation of their actions, but any two commits can be compared against one another as a “*diff*”, a description of the line-by-line differences between each differing file. Reading the diff between two commits is often more illuminating than reading the descriptions and intentions in the commits’ commit messages. Diffs provide a clear description of how textual content in a project has changed, making them especially useful when working in code or when composing textual input to automated typesetting programs like LilyPond and LaTeX. Consider how difficult such a version comparison task would be when comparing the contents of two project directories by hand – probably impossible, but certainly very time-consuming and error-prone.

---

<sup>3</sup><http://git-scm.com/>

Finally, version control affords collaboration and experimentation. Collaboration here crucially also includes collaboration with oneself. Many version control systems, including git, support *branches* which allow the revision history of the project to split into parallel time-lines rather than follow a single linear path. One can create branches whenever one desires, for example when attempting to solve a problem in more than one way where each attempt is isolated from one another but still maintained by the project's version history. Branches can also be merged into one another, converging parallel version histories back into a linear history. Such flexibility gives authors the freedom to embark on radical or incremental revisions and reorganizations of their projects without fear of confusion or lost work.

# 6

## Conclusion

The previous chapters have discussed a computational model of music composition, implemented in the Python library Consort, and the model of notation which it extends, Abjad. The various open-source systems – L<sup>A</sup>T<sub>E</sub>X, LilyPond, Python, etc. – which interoperate to make these twin computational models possible have also been demonstrated, and some standard solutions for establishing a document preparation workflow which streamlines and accelerates a cycle of score visualization through automated typesetting has been proposed.

As described in chapter 2, Abjad’s model of notation treats musical score as a hierarchy consisting of containers – staves, voices, measures and tuplets – and leaves – notes, rests and chords –, to which indicators – clefs, dynamics, etc. – and spanners – slurs, beams, glissandi, hairpins, and so forth – can be attached. Abjad’s model is clear and explicit whenever possible. Those objects comprising a score which a composer might wish to create – what

we might call the semantic content of the score<sup>1</sup> – are all represented by classes in Abjad, each with a well-defined interface exposing only those properties and methods pertinent to that class. Abjad’s notation model strives for composition-process agnosticism<sup>2</sup>, allowing composers to work directly with the elemental notation objects rather than obligating them to rely on opinionated or idiosyncratic mechanisms. Abjad provides a variety of models of musical time, discussed at length in chapter 3, such as timespans and metrical hierarchies. These time models permit alternative means of constructing, coordinating and transforming musical structures than those provided by simply working with score trees directly. Timespans, especially, afford the sketching of dense polyphonic phrasing structures, and have been foundational to my working process for years, explicitly since *Aurora* and certainly with intention, although not name, for many years prior to that. Although I and the other Abjad developers have found timespans to be incredibly utilitarian, and certainly one of the most fundamental tools in our toolkit for talking about time in score, I initially<sup>3</sup> developed them as an affordance for structuring large-scale orchestral works. All of these aspects, combined with Abjad’s tools for iterating over, selecting, and inspecting score components provides a strong foundation for others to implement their own personal models of composition: how one goes about organizing notation into a musical work.

For my part, Consort constitutes such a model of composition: a collection of high-level abstractions for organizing the elements of notation. Consort divides the process of composition into two stages – specification and interpretation – and proposes – but does not enforce – that scores be structured as a series of segments.<sup>4</sup> Specification entails the configuration of a segment-maker – the object responsible for coordinating the creation of a segment of score – with music settings. These “settings” bundle a timespan-maker, whose responsibility it is for determining when and in which voices some material should appear, with music specifiers, objects aggregating

---

<sup>1</sup>As opposed to those objects which are necessary or implicit, such as staff lines, bar lines, measure numbers, etc. Of course, for some composers, staff lines can and do represent semantic musical content. However, when creating input for LilyPond, I would argue that staff lines are generally simply implicit.

<sup>2</sup>Agnosticism here stretches only so far as being agnostic of all compositional processes so long as they revolve around Western common practice notation.

<sup>3</sup>Timespans as a compositional tool in Abjad began, in spirit, with the `timeintervaltools` subpackage, my first large contribution to Abjad, authored around 2010, which introduced a timespan-like class called `TimeInterval` and a `TimeIntervalTree` for containing them. These classes were named after the “interval-tree” data structure, often used for modeling scheduling conflicts, as it provides a highly-optimized search algorithm for finding overlap between one time interval and other time intervals or offsets. Trevor Bača later introduced a much more generalized `timespantools` subpackage and nominative `Timespan` class, into which I merged some of the more idiosyncratic `timeintervaltools` functionality, such as timespan explosion.

<sup>4</sup>Segmentation acts both as a practical aid for typesetting, allowing smaller portions of the score to be visualized anew, and as a cognitive aid to the composer, by constraining the scope of detail they must confront during specification to a more manageable amount.

together the various makers and handlers defining a musical material. Once configured, the segment-maker may be interpreted, evaluating each of its music settings to generate a maquette – an annotated timespan structure describing the location of musical materials in the score, but not yet their notation – as well as a governing sequence of meters. That maquette is then progressively interpreted into notation, with each timespan’s annotating music specifier contributing rhythm, pitch and other typographic information to the resulting segment of score.

## 6.1 CONCERNS & IMPLICATIONS

As described in section 4.2, Consort’s specification and interpretation process treats the act of composition analogously to the act of *compilation* in software. Like a compiler, Consort’s segment-maker parses a high-level description of music – its music settings, timespan-makers and music specifiers – into an intermediate representation – the timespan maquette, itself perhaps poetically akin to computing’s notion of an *abstract syntax tree*<sup>5</sup> –, and finally converts that intermediate representation into “low-level” notational primitives. In this way, Consort privileges composition with the procedural, or the general, over the specific. It is much more difficult – although not impossible – to change a single pitch at one moment in time than it is to change all of the pitches in an entire score. This has tremendous practical implications when working with such a highly-procedural system and, from experience, can be rather problematic. For example, because pitches are often “painted” onto the score in time-wise order across different voices, adding or removing a single attack-point can shift the pitches painted onto all subsequent attack-points. This expressive “entanglement” makes revising music already in the hands of performers treacherous, and I’m certainly guilty of raising some eyebrows from time to time. But the ability to describe and perform precise, mass transformations on musical materials – even if occasionally unintentionally – is one of, if not *the* driving motivation behind Consort. Segments may be stretched, while preserving their overall internal phrase structure. The rhythm-makers inscribing a subset of a maquette can be swapped for other rhythm-makers, yielding wholly different surface textures. Runs of notes and chords occupying weighted pitch centers can be selected and octavated en masse. Such transformations are afforded by computation. And from a computational perspective, one can consider Consort as a system which treats scores as enormous composite expressions, comprising the notational sum of the interpretation algorithm applied against each specification:

$$\sum_{i=1}^n \text{Interpretation}(\text{Specification}_i) \quad (6.1)$$

---

<sup>5</sup>I do not want to overstretch this metaphor, though. I doubt it would hold up to vigorous inspection.

In considering Consort as revolving around *score-as-expression*, it's also worth noting that randomness – random number generators, noise functions, coin flipping, or any other such variant – plays no part in this discussion. Every segment-maker, every rhythm-maker, pitch-handler or other procedural mechanism both in Consort's ecosystem and Abjad's, is completely deterministic. This decision is primarily pragmatic. Each package's testing regime is considerably simpler without randomness, and the results produced by each system remain stable across multiple runs. But there's another realisation at work here. I would argue that randomness is often a proxy for richness, detail or creative “touch”. Artists generally rely on randomness as a tool not for ideological or conceptual reasons – although some certainly do –, but simply because it affords the rapid production of material and variations on that material. In effect, a labor-saving device, and one which I relied on myself for a number of years in my acoustic music, for example in *Aurora* (chapter 7), as well as in all of my electronic music to this day. But there are other, less surprising means of production. Any sufficiently complex, but finite, fixed pattern of values is liable to be indistinguishable to a listener from a random sequence. This is compounded when multiple sequences of different lengths interact, as is the case in the talea timespan-makers described in subsection 3.4.2, as well as most rhythm-makers. In fact, such sequences need not be particularly long. Less than ten integers in a talea's count sequence is often sufficient, so long as it combines with other patterns, such as the prolation-inducing `extra_counts_per_division` keyword. Fixed patterns also offer something that random sequences do not: the ability to both appear random, and to appear memorable. I suppose this realisation is rather trivial, but it took me a number of years to fully come to terms with it as a working philosophy and, in practice, the sequences I use for phrasing, rhythm, pitches and anything else have become shorter and shorter. In retrospect, it's worth interrogating whether randomness is necessary or desirable at all.

Consort's origin as a software library and the composition model it implements contains a number of other implicit assumptions. Amongst these – and I can only speak for myself – is that composition is most strongly situated in the act of specification. When I compose, I specify what will be in the score, and I specify where and how it will come to be there – that is, by what processes. This description is either a little vague or over-obvious, but consider that each of the three *Invisible Cities* scores, in chapter 9, chapter 10 and chapter 11, rely on only superficially different segment-makers. Virtually all of their differentiation lies in the specification of their segments, not in the process by which they are interpreted. Interpretation then becomes almost like an instrument performing these different specifications, a recapitulation of so much of my electronic composing, where custom synthesizers and audio processing networks perform different configurations over and over, auditioning for me the materials

I will later maquette into the final work. In both cases, the acoustic and the electronic, I am also acting – quite pointedly – as the author of these “instruments” and so the algorithm used to perform a given specification is certainly not exterior to composition, just positioned differently: less specific, more general, like a compositional voice or fingerprint rather than a particular performance. Crucially, these algorithms persist from one score to the next, just as I re-use and re-combine the synthesizers I created for one electronic piece in another. The scores created with Consort are an extreme example of this re-use, undertaken specifically to investigate its practicality.

Abstraction, encapsulation, inheritance, all foundational principles in computing, point at re-use. They act to conserve labor, to persist the work done on one day in formalizing a process so that it might be reapplied on another day with little additional effort. In effect, they act as a kind of accelerant in the development and extension of further computational systems. And computing of course allows us to do things *fast*. But I believe it’s important to be cautious, even deeply suspicious of this. The same computing research, built over cumulative generations often in the service of finance and war, powering the so-called information age we find ourselves in today – its prizes being efficiency, productivity, accuracy, connectivity – also allows me to create new musical material faster and in greater quantity than I can ever hope to read through. This is a false economy. Unlike with electronic music, my attention simply cannot keep up with an endless stream of score. Speed has its place – and I am no more immune to its siren call than anyone else – but I hope to firmly position the benefits of computation, of library-writing elsewhere. The same qualities which afford speed often also afford structure, and ultimately extensibility. Consort and Abjad’s existence as libraries in a network of other libraries, their “library-ness”, their “open-sourceness” – even their testing regimes – act as bulwarks against the loss of knowledge encoded in them, and encourage others to interrogate, critique and extend their structure, and the music-making world they propose.

Finally, it’s important to reiterate a comment from the introduction: when working within a formal framework, one only has computational, programmatic control over what that framework describes. That which cannot be named, cannot truly be touched, and if it can be named, but cannot be described with clarity, it can only be grasped weakly, or even incorrectly. This is both cautionary to those who work in formal systems, and perhaps a panacea for those who eschew them. There is much work to be done to bring more names and descriptions into the “light,” but many things simply cannot be described specifically enough for a creature as stupid as a computer to understand.

## 6.2 FUTURE WORK

In no way do I consider this project finished. Nor do I think a project like this – both the modeling of notation and composition – can ever be complete. There is, in my opinion, no single universal methodology to composition, nor should there ever be. And there is still quite a lot of work to do to solve an entire array of practical problems, let alone the compositional ones I often wait months or years to approach. Having devoted so much effort to large- and small-scale time structures, I need to turn my attention toward harmony and orchestration as constraints and coordinating forces. Convincing piano music remains a bugaboo. Multi-staff writing, with voices crossing between upper and lower staves and back again is well supported by LilyPond, which was designed with the spacing concerns of dense Romantic music as a foremost priority, but which requires considerable hand-adjustment. Without careful rhythmic and pitch control to account for collisions, procedurally-generated staff-changing music in a multi-voice texture quickly becomes a mess. Likewise, the use of dependent timespan-makers to create pedaling voices based on other voices, as outlined in subsection 3.4.3, sometimes produces satisfactory results, but more often doesn’t. The dependent timespan-maker is unable to truly account for the events it reacts against, and creates pedaling changes at timespan boundaries rather than at meaningful musical moments. Of course, what constitutes a meaningful musical moment is difficult to say, and I’m simply not sure yet that I *can* say. And in light of my cautions about naming and describing concepts computationally, I’m not sure I ever will.

Any additional affordances for idiomatic instrumental writing would be a great help: models of string instrument fingerings, and catalogues of woodwind trills and multiphonics, as well as the relative dynamic ranges in different registers and with different techniques: all of the quantitative knowledge contained in the various orchestration manuals which composers make use of. Perhaps most crucially, mechanisms for specifying that literal music expressions – not procedures to be applied during interpretation, but fully complete excerpts of music – be placed into a segment-maker’s output wherever desired would greatly extend Consort’s expressivity. The same holds true for transformations on the interpreted music, such as shifting a phrase forward or backward along the timeline, or deleting specific moments.

## 6.3 PARTING WORDS

My intention with providing the complete sources to both my scores and working methods – as laid out in the appendices, and discussed throughout the preceding chapters – is not that others copy me, although they certainly

can if they like. This work is open-source, after all. I wouldn't be offended, but maybe a little disappointed. Why would someone who managed to put together the tools and knowledge to successfully interpret one of these scores also not take the time to place some personal stamp on their duplication by turning the knobs or mixing the potions differently, at least a little bit? It seems so unlikely to me, and I'm sure I'll be surprised if and when it ever happens.

Rather, my intentions are purely pedagogical. I hope this shared knowledge can be something like a lighthouse to those who come after me, rather than hiding it away to collect dust. And while the code presented here may become dusty, as most code does, the concepts and techniques – as separate from their concrete implementations – most likely won't. While I presented all of the work in this dissertation in the programming language Python and with LilyPond as its typesetting engine, these notional and compositional models could and certainly *should* be implemented in many other programming languages. A rich ecosystem of models would do wonders to keep stagnation at bay, to share knowledge amongst colleagues, and to reduce the barriers-to-entry for newcomers to this compositional modality.

I imagine myself, a composer ten years younger, searching for answers to many of the questions I've now made good progress on solving, questions which rarely even concern making art because there is still too much groundwork to lay. Had I a clear path then to follow, diverge from, reverse-engineer, or even wholly reject, maybe I would have produced more music by now. Or maybe not. I no longer have the same misgivings as I did when younger about splitting my creative energies between composition and engineering. If anything, they don't seem that different to me anymore.

BASTA.

This page intentionally left blank.

## **Part II**

## **Practice**

This page intentionally left blank.

# 7

## *Aurora (2011)*

A composition for string orchestra

Premièred by Ensemble Kaleidoskop  
on Friday July 1st, 2011  
in the Chamber Music Hall  
of the Philharmonie, Berlin

# Performance Notes

## 1 Instrumentation

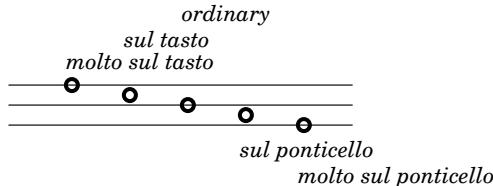
*mbrsi/aurora* has the following instrumentation:

- 12 violins
- 4 violas
- 4 cellos
- 2 contrabasses

## 2 Bowing

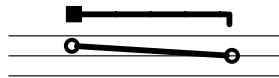
### 2.1 Bow Positions

Bow positions from *sul ponticello* to *sul tasto* are indicated in 3-line tablature fragments above the main staff:



### 2.2 Overpressure

Overpressure is indicated by a black box and bracket above the bowing-staff:



### 2.3 Circular Bowing / Ponticello-Tasto Tremoli

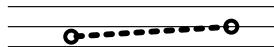
Zigzags on the bowing-staff indicate rapid circular bowing (essentially a tremolo from *sul tasto*

to *sul ponticello*):



### 2.4 Jete / Spiccato

Dotted lines on the bowing-staff indicate a *jéte* or similarly bounced bow:



## 3 Glissandi

### 3.1 Normal Glissandi

Two types of glissandi are prescribed. The first, with a straight line, is to be played as expected:



### 3.2 Oscillations

The second, with a zigzag-line, indicates a glissandi with a very, very wide vibrato, of at least a few semitones:



# AURORA

for Ensemble Kaleidoskop

Josiah Wolf Oberholzer (nj8f)

*J = 66*

Violin 01  
Violin 02  
Violin 03  
Violin 04  
Violin 05  
Violin 06  
Violin 07  
Violin 08  
Violin 09  
Violin 10  
Violin 11  
Violin 12  
Viola 01  
Viola 02  
Viola 03  
Viola 04  
Cello 01  
Cello 02  
Cello 03  
Cello 04  
Contrabass 01  
Contrabass 02

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Pizz.

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(7)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

*(II)*

Violin o1 *fppp*

Violin o2

Violin o3 *(I-II-III-IV)* *fppp* *p*

Violin o4

Violin o5

Violin o6

Violin o7 *(I-II-III-IV)* *fppp* *Pizz.*

Violin o8 *p*

Violin o9

Violin o10

Violin o11

Violin o12

Viola o1 *(I-II-III-IV)* *fppp* *p*

Viola o2

Viola o3

Viola o4 *(I-II-III-IV)* *fppp* *p*

Cello o1

Cello o2

Cello o3 *(I-II-III-IV)* *fppp* *p*

Cello o4 *(I-II-III-IV)* *fppp* *p*

Contrabass o1

Contrabass o2 *p*

(15)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(19)

Violin 01  
Violin 02  
Violin 03  
Violin 04  
Violin 05  
Violin 06  
Violin 07  
Violin 08  
Violin 09  
Violin 10  
Violin 11  
Violin 12  
Viola 01  
Viola 02  
Viola 03  
Viola 04  
Cello 01  
Cello 02  
Cello 03  
Cello 04  
Contrabass 01  
Contrabass 02

(23)

(28)

(33)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(7)

Violin or

Violin oz

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin n

Violin o12

Viola or

Viola oz

Viola o3

Viola o4

Cello or

Cello oz

Cello o3

Cello o4

Contrabass or

Contrabass oz

(1)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Pizz.

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(5)

Violin or

Violin o2

Violin o3 (IV-II-II-D)

Violin o4

Violin o5

Violin o6

Violin o7 (IV-II-II-D) (IV-II-II-IV)

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12 (I-II-II-IV)

Viola or

Viola o2

Viola o3

Viola o4 (IV-II-II-D) (II-II-III-IV)

Cello or

Cello o2

Cello o3 (I-II-III-IV) (IV-II-II-D)

Cello o4

Contrabass or

Contrabass o2

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin n

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

33

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

57

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(61)

Violin or

Violin o2

Violin o3 (IV-II-II-IV) (I-II-III-V)

Violin o4

Violin o5 (IV-II-II-IV) (I-II-III-V)

Violin o6 (IV-II-II-IV) (I-II-III-V)

Violin o7

Violin o8

Violin o9

Violin o10 (IV-II-II-IV) (I-II-III-V)

Violin o11

Violin o12 (IV-II-II-IV) (I-II-III-V) ppp

Viola or (I-II-III-V)

Viola o2 ffpp p

Viola o3

Viola o4 (I-II-III-V) p

Cello or (I-II-III-IV) ffpp

Cello o2 (I-II-III-IV)

Cello o3 (I-II-III-IV) ffpp p

Cello o4 (I-II-III-IV)

Contrabass or (I-II-III-IV) ffpp

Contrabass o2 (I-II-III-IV) mp

(65)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin n

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(69)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(2)

Violin o1

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola o1

Viola o2

Viola o3

Viola o4

Cello o1

Cello o2

Cello o3

Cello o4

Contrabass o1

Contrabass o2

(77)

Violin o1  
Violin o2  
Violin o3  
Violin o4  
Violin o5  
Violin o6  
Violin o7  
Violin o8  
Violin o9  
Violin o10  
Violin n  
Violin o12  
Viola o1  
Viola o2  
Viola o3  
Viola o4  
Cello o1  
Cello o2  
Cello o3  
Cello o4  
Contrabass o1  
Contrabass o2

(82)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(ff)

Violin o1

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7 (S-B-B-D-D)

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola o1

Viola o2

Viola o3

Viola o4 (D-B-B-D-D)

Cello o1 (D-B-B-D-D)

Cello o2

Cello o3

Cello o4 (D-B-B-D-D)

Contrabass o1

Contrabass o2

(9)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

*(g)*  
 Violin o1  
 Violin o2  
 Violin o3  
 Violin o4  
 Violin o5  
 Violin o6  
 Violin o7  
 Violin o8  
 Violin o9  
 Violin o10  
 Violin o11  
 Violin o12  
 Viola o1  
 Viola o2  
 Viola o3  
 Viola o4  
 Cello o1  
 Cello o2  
 Cello o3  
 Cello o4  
 Contrabass o1  
 Contrabass o2

(8)

Violin or

Violin o2

Violin o3

Violin o4 (IV-III-IV) Pizz.

Violin o5 (IV-III-IV) f

Violin o6 (IV-III-IV) ffpp nf

Violin o7 (IV-III-IV) Pizz.

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4 Pizz. (IV-III-IV) Pizz.

Cello or

Cello o2

Cello o3 (IV-III-IV)

Cello o4 ppp

Contrabass or

Contrabass o2

102

Violin o1

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola o1

Viola o2

Viola o3

Viola o4

Cello o1

Cello o2

Cello o3

Cello o4

Contrabass o1

Contrabass o2

(105)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

108

This page contains three systems of musical notation for a large string ensemble. The instrumentation includes multiple violins (12 parts), violas (4 parts), cellos (4 parts), and a double bass (2 parts). The music is divided into measures by vertical bar lines. Each measure contains several notes and rests, with specific dynamics and performance techniques indicated above the staff. The first system starts with a dynamic of *ppp*, followed by *p*, *mp*, *mf*, and *ff*. The second system begins with *ppp* and includes markings like *Pizz.*, *Sl.*, and *Dotted*. The third system features dynamics such as *f*, *ff*, *mp*, *mf*, and *ppp*. Various performance instructions are scattered throughout the score, including *(I-II-III-IV)*, *(IV-VI-VII)*, and *(VII-VIII-IX)*. Articulation marks like *pizz.*, *sl.*, and *dotted* are also present.

(III)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(115)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin n

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(119)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(122)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05 (IV-II-III-IV) (I-II-III-IV)

Violin 06 (IV-II-III-IV) (I-II-III-IV)

Violin 07 (IV-II-III-IV) (I-II-III-IV)

Violin 08 (IV-II-III-IV) (I-II-III-IV)

Violin 09 (IV-II-III-IV) (I-II-III-IV)

Violin 10

Violin 11

Violin 12 (IV-II-III-IV) (I-II-III-IV)

Viola 01

Viola 02 (IV-II-III-IV) (I-II-III-IV)

Viola 03 (IV-II-III-IV) (I-II-III-IV)

Viola 04 (IV-II-III-IV) (I-II-III-IV) Pizz.

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01 (IV-II-III-IV) (I-II-III-IV)

Contrabass 02 (IV-II-III-IV) (I-II-III-IV)

126

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(130)

Violin or

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin n

Violin 12

Viola or

Viola 02

Viola 03

Viola 04

Cello or

Cello 02

Cello 03

Cello 04

Contrabass or

Contrabass 02

(134)

138

This page from a musical score contains 21 staves of music for a large orchestra. The instruments listed on the left are: Violin 01, Violin 02, Violin 03, Violin 04, Violin 05, Violin 06, Violin 07, Violin 08, Violin 09, Violin 10, Violin 11, Violin 12, Viola 01, Viola 02, Viola 03, Viola 04, Cello 01, Cello 02, Cello 03, Cello 04, Contrabass 01, and Contrabass 02. The music is divided into measures by vertical bar lines. Each staff includes dynamic markings such as *p* (piano), *f* (forte), *pp* (pianissimo), *mf* (mezzo-forte), and *mp* (mezzo-piano). Articulation marks like *sf* (sforzando) and *sfz* (sforzando zappato) are also present. Performance instructions include tempo markings like *1.5*, *2.5*, and *3.5*, and specific labels like *(IV-II-III-IV)* and *(IV-III-II-I)*. The score is highly detailed, showing intricate patterns of notes and rests across all staves.

(ff)

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9 (I II III IV)

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(165)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(149)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(152)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(155)

159

Violin o1

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola o1

Viola o2

Viola o3

Viola o4

Cello o1

Cello o2

Cello o3

Cello o4

Contrabass o1

Contrabass o2

A detailed musical score page, numbered 163, featuring 25 staves of music for a string orchestra. The instruments listed on the left are: Violin 01, Violin 02, Violin 03, Violin 04, Violin 05, Violin 06, Violin 07, Violin 08, Violin 09, Violin 10, Violin 11, Violin 12, Viola 01, Viola 02, Viola 03, Viola 04, Cello 01, Cello 02, Cello 03, Cello 04, Contrabass 01, and Contrabass 02. The music consists of three systems of measures. Measure 1 starts with sustained notes and dynamic markings like  $p$ ,  $fpp$ , and  $ppp$ . Measure 2 continues with sustained notes and dynamic markings. Measure 3 concludes with sustained notes and dynamic markings. Various performance techniques are indicated throughout the score, such as slurs, grace notes, and dynamic markings like  $f$ ,  $mf$ ,  $p$ ,  $fpp$ , and  $ppp$ . Some staves have specific measure numbers written above them, such as '12-13-14' and '12-13-14'. Measure 12 is also labeled '12-13-14'.

167

Violin or

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola or

Viola o2

Viola o3

Viola o4

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(171)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

p

f

fpp

*z*

*IV III 4.0*

*z*

*IV 42-32 4*

(175)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

175

176

(IV-II-12-2) (IV-III-11-6)

(IV-III-11-5)

(IV-III-11-4)

(IV-III-11-3)

(IV-III-11-2)

(IV-III-11-1)

(179)

Violin or

Violin o3

Violin o5

Violin o6

Violin o7

Violin o8

Violin og

Violin o10

Violin n

Violin o12

Viola or

Viola o2

Viola o3

Viola o5

Cello or

Cello o2

Cello o3

Cello o4

Contrabass or

Contrabass o2

(183)

Violin o1

Violin o2

Violin o3

Violin o4

Violin o5

Violin o6

Violin o7

Violin o8

Violin o9

Violin o10

Violin o11

Violin o12

Viola o1

Viola o2

Viola o3

Viola o4

Cello o1

Cello o2

Cello o3

Cello o4

Contrabass o1

Contrabass o2

(186)

Violin or

Violin o2

Violin o3 (IV-II-III) (IV-V-I-D)

Violin o4 (IV-II-III) (IV-V-I-D)

Violin o5 (IV-II-III) (IV-V-I-D)

Violin o6

Violin o7

Violin o8 (IV-V-I-D)

Violin o9 (IV-V-I-D) (IV-VII-D)

Violin o10 (IV-VII-D)

Violin o11 (IV-VII-D) (IV-VI-D)

Violin o12

Viola or

Viola o2

Viola o3

Viola o4 (IV-VII-D)

Viola o5 (IV-VII-D) (IV-VI-D)

Cello or (IV-VII-D) (IV-VI-D)

Cello o2 (IV-VII-D)

Cello o3 (IV-VII-D) (IV-VI-D)

Cello o4

Contrabass or

Contrabass o2

190

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

(194)

Violin 01

Violin 02

Violin 03

Violin 04

Violin 05

Violin 06

Violin 07

Violin 08

Violin 09

Violin 10

Violin 11

Violin 12

Viola 01

Viola 02

Viola 03

Viola 04

Cello 01

Cello 02

Cello 03

Cello 04

Contrabass 01

Contrabass 02

This page intentionally left blank.

# 8

## *Plague Water (2014)*

A composition for baritone saxophone,  
electric guitar, piano and percussion

Premièred by Ensemble Nikel  
on Saturday April 5th, 2014  
in Paine Hall, Harvard University

# PREFACE

## 1

From Volodimir Pavliuchuk's *Cordial Waters*:

**No.1 Plague Water (1671, England)**

- 150 gm scabious (*Scabiosa sp.*)
- 150 gm pimpernel (*Anagallis arvensis*)
- 150 gm tormentil root (*Potentilla erecta*)
- 4 litres 5% malt extract wash  
(*strong beer as in the original*)

Macerate for 12 hours and then distil.

The recommended does is a spoonful every 4 hours.

**No.2 Plague Water (1677, England)**

- 100 gm rue
- 100 gm rosemary
- 100 gm sage
- 100 gm sorrel
- 100 gm celandine (*Chelidonium majus*)  
(*The leaves contain small amounts of toxic alkaloids which can be reduced greatly by drying the plant*)
- 100 gm mugwort (*Artemisia vulgaris*)
- 100 gm bramble (blackberry) tops
- 100 gm pimpernel (*Anagallis arvensis*)
- 100 gm dragons (*Dracunculus vulgaris*)
- 100 gm agrimony (*Agrimonia eupatoria*)
- 100 gm lemonbalm
- 100 gm angelica leaves
- 4 litres white wine  
(*substitute a 15% ABV sugar wash*)

Macerate for 5 days and then distil.

## 2

**Baritone Saxophone**

Bartok-pizzicato indications above noteheads indicate slap tongues.

**Electric Guitar**

The electric guitar should be treated with 6 different colors, via effects pedal(s). The color to be used is indicated at the beginning of each section of the score. Pedal colors may include any combination of distortion, reverb or short delay (less than a quarter second). A volume pedal should be placed last in the effect chain, to control overall dynamic.

**Piano**

Cross-shaped noteheads indicate glissandi on the tops of the keys, without depressing the keys, played with the flesh of the fingers, or fingernails. A flat or natural sign above the glissandi determines whether to play on the black or white keys.

**Percussion**

Instrumentation is somewhat up to the discretion of the performer, but should obey the following guidelines:

- 4 wooden shakers, bamboo wind-chimes, maracas, rainsticks, cabasa, caxixi etc. These could include metal timbres, but should be primarily wood. The order of the shakers is not important. Instruments with a longer decay, and a more granular sound quality, such as rainsticks and bamboo windchimes are preferred.
- 5 wood blocks, arranged from lowest to highest. The exact pitch is not important. These could also be temple blocks. The sound quality should be very dry.
- 3 large drums, including at least one proper bass drum, arranged from lowest to highest.

Percussion should be performed with bare hands. Wooden rings may be worn to increase the overall dynamic, especially on the wood blocks. Styrofoam blocks should be placed on the bass drums, to be used during the rehearsal marks indicated in the score (4, 14, 17a, 17b). Grace notes should always be played with the hands.

# PLAQUE WATER

*for Ensemble Nikel*

Josiah Wolf Oberholtzer (1984)

$\text{♩} = 64$

$\frac{2}{4}$  **I**  $\frac{2}{8}$

*baritone saxophone*

*electric guitar* [COLOR ONE]

*piano*

*4 wood shakers*

*5 wood blocks*

*3 bass drums*

Dynamic markings: ffz, ff, f, mp, 3.2, 5.3, 5.4, L.V.

$\text{J} = 80$

5      **2**      **8**      **16**      **3** [2]

9      **6**      **3**

baritone saxophone      electric guitar      piano      wood shakers      bass drums

12

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

$\text{♩} = 72$

6

$\frac{2}{4}$  [3A]

4r

*mp* — 4.3 — 3.2 — *f*

*ff* — 3.2 —

[COLOR THREE]

*p*

*p*

*p*

*p*

*p* — 4.3 —

*p* — 3.2 — *mp*

*p*

15

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

$\frac{4}{4}$

2

$\frac{2}{8}$

4r

3.2 — *ff* — 4.3 — 5.3 — *mf* — 4.3 — *ff* — 3.2 —

*p*

*p* — *pp* — 4.3 —

*p*

— L.V. —

*p*

*p* — L.V. —

*p*

*p*

*p* — 4.3 — *f*

*p* — 8.5 — *mp* —

*p*

mp — 12.11 —

19

**2**

**4**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

ffz 4.3 ffz

p 3.2 p

p L.V.

p L.V.

mp 12.7 f 6.5 ffz

mp 3.2 16.9 f

22

**8**

**2**

**8**

baritone  
saxophone

electric  
guitar

piano

wood  
shakers

wood  
blocks

bass  
drums



32

**2**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

33

**2**

**2**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

40

**baritone saxophone**

**electric guitar**

**piano**

**wood shakers**

**wood blocks**

**bass drums**

**43**

**8** **4**

**baritone saxophone**

**electric guitar**

**piano**

**wood shakers**

**wood blocks**

**bass drums**

45

**4**

**5**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

*air*

*Sva*

*Sva*

*pp*

*p*

6.5

6.5

3.2

4.3

48

**6**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

*air*

*Sva*

*Sva*

4.3

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

55      **8**      **16**      **8**

*baritone saxophone*  
*electric guitar*  
*piano*  
*wood shakers*  
*wood blocks*  
*bass drums*

**59**      **8**

J = 80      **3** **4** **5**

*baritone saxophone*  
*electric guitar*  
*piano*  
*wood shakers*  
*wood blocks*  
*bass drums*

62

**6**      **8**      **4**      **8**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

**J = 112**

66

**9**      **4**      **8 [6]**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

**COLOR FIVE**

70

$\text{♪} = 48$   
 $\frac{5}{16} \boxed{7}$

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

76       $\frac{7}{16}$        $\frac{2}{4}$        $\frac{3}{4}$        $\frac{5}{16}$

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

$\text{♩} = 80$

89  $\frac{3}{4}$  [8]  $\frac{16}{8}$   $\frac{3}{8}$   $\frac{3}{4}$

*baritone saxophone*:  $\text{mp}$   $f$  5.3  $\text{mp}$  6.5  $\text{mp}$

*electric guitar*: **COLOR TWO**  $\text{mp}$  5.3  $\text{mp}$  6.5

*piano*:  $\text{p}$  3.2  $\text{p}$  Sub 8.5  $\text{p}$

*wood shakers*:  $\text{p}$  4.3

*wood blocks*:  $\text{p}$  3.2  $\text{p}$  3.2  $\text{p}$

*bass drums*:  $\text{p}$

84  $\frac{3}{8}$   $\frac{6}{8}$   $\frac{3}{8}$

*baritone saxophone*:  $\text{mp}$  6.5  $\text{mp}$  3.2  $f$

*electric guitar*:  $\text{mp}$  6.5  $\text{mp}$  3.2  $f$

*piano*:  $\text{p}$  Sub 3.2  $\text{pp}$   $\text{p}$   $\text{mf}$

*wood shakers*:  $\text{p}$  Sub 3.2  $\text{pp}$   $\text{p}$

*wood blocks*:  $\text{p}$  8.7  $\text{p}$   $\text{pp}$

*bass drums*:  $\text{p}$   $\text{pp}$

87      **3**       **$\frac{2}{4}$  [9]**      **8**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

**322**

$\text{♪} = 48$

91      **2**      **8**      **3**      **16**      **5** **10 A**      **7** **16**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

97      **2**      **4**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

*i* 100 **I6**  $\frac{2}{4}$  **IOB**

$\text{♪} = 48$

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

**COLOR SIX**

3.2 4.3 15ma 3.2 15ma 3.2  
Sva 3.2 3.2 4.3  
L.V. 3.2 3.2 3.2

**STYROFOAM**

$f$  12.7  $\text{ff}$   $mp$   $mp$  4.3  $mp$   $mp$  3.5  $p$   $mp$

*i* 104 **2** **5** **16**  $\frac{2}{4}$  **IOC**

$\text{♪} = 48$

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

**COLOR SIX**

15ma 15ma 15ma 3.2 3.2 3.2  
Sva 3.2 3.2 3.2  
L.V. 3.2 3.2 3.2

**STYROFOAM**

10.9  $f$   $mp$   $f$  3.2  $\text{ff}$   $mp$  3.2

108      **3**      **2**      **16**  
*baritone saxophone*  
*electric guitar*  
*piano*  
*wood shakers*  
*wood blocks*  
*bass drums*

$\text{J} = 48$   
 112      **2**      **10D**      **8**      **2**  
*baritone saxophone*  
**COLOR SIX**  
*electric guitar*  
*piano*  
*wood shakers*  
*wood blocks*  
**STYROFOAM**  
*bass drums*

*116*      **5**      **16**      **5**      **2**      **2** [II]

*J = 64*

**baritone saxophone**  
*p*      *pp*

**electric guitar**  
*ppp*

**piano**  
*15ma*      *15ma*  
*pp*      *3.2*      *3.2*  
*8va*      *8va*  
*p*      *3.2*  
*8va*      *L.V.*

**wood shakers**  
*ppp*

**wood blocks**

**bass drums**  
*f*

*mp*      *3.2*      *6.5*      *ff*

*mp*      *4.3*

**COLOR ONE**

*g2*      *g2*  
*g2*      *5.3*  
*g2*      *5.4*  
*g2*

*mp*

*121*

**2**      **2**      **2**      **3**  
**8**      **4**      **8**      **16**

**baritone saxophone**  
*3.2*      *g2*      *g2*      *g2*  
*g2*      *g2*      *g2*      *f*

**electric guitar**  
*>*      *>*      *>*      *>*  
*g2*      *5.4*      *g2*      *g2*  
*g2*      *g2*      *g2*      *f*

**piano**  
*g2*      *5.4*  
*g2*      *5.3*  
*g2*      *5.3*      *5.4*  
*g2*      *5.4*  
*g2*      *5.3*  
*g2*      *5.3*      *5.4*  
*g2*      *5.3*  
*g2*      *5.3*

*8va*      *L.V.*  
*8va*      *8va*  
*8va*      *8va*      *L.V.*

**wood shakers**  
*mp*      *3.2*

**wood blocks**

**bass drums**  
*mp*

*mp*      *4.3*      *3.2*  
*mp*      *f*

$\text{♩} = 72$   
 127 **12A**

**baritone saxophone**:  $\frac{4}{4}$  (measures 1-2),  $\frac{2}{8}$  (measures 3-4),  $\frac{4}{4}$  (measures 5-6). Dynamics:  $\text{fz}$ ,  $\text{fz}$  (with 3.2),  $\text{f}$ ;  $\text{fz}$  (with 3.2),  $\text{f}$  (with 3.2). Articulation:  $\text{b}$  (measure 1),  $\text{b}$  (measure 2),  $\text{b}$  (measure 3),  $\text{b}$  (measure 4),  $\text{b}$  (measure 5),  $\text{b}$  (measure 6).

**electric guitar**:  $\frac{4}{4}$  (measures 1-2),  $\frac{2}{8}$  (measures 3-4),  $\frac{4}{4}$  (measures 5-6). Dynamics:  $\text{p}$  (with 3.2),  $\text{p}$  (with 3.2). Articulation:  $\text{b}$  (measure 1),  $\text{b}$  (measure 2),  $\text{b}$  (measure 3),  $\text{b}$  (measure 4),  $\text{b}$  (measure 5),  $\text{b}$  (measure 6).

**piano**:  $\frac{4}{4}$  (measures 1-2),  $\frac{2}{8}$  (measures 3-4),  $\frac{4}{4}$  (measures 5-6). Dynamics:  $\text{p}$  (with 4.3),  $\text{p}$  (with 3.2). Articulation:  $\text{b}$  (measure 1),  $\text{b}$  (measure 2),  $\text{b}$  (measure 3),  $\text{b}$  (measure 4),  $\text{b}$  (measure 5),  $\text{b}$  (measure 6).

**wood shakers**:  $\frac{4}{4}$  (measures 1-2),  $\frac{2}{8}$  (measures 3-4),  $\frac{4}{4}$  (measures 5-6). Dynamics:  $\text{L.V.}$  (measure 1),  $\text{L.V.}$  (measure 2),  $\text{L.V.}$  (measure 3),  $\text{L.V.}$  (measure 4),  $\text{L.V.}$  (measure 5),  $\text{L.V.}$  (measure 6).

**wood blocks**:  $\frac{4}{4}$  (measures 1-2),  $\frac{2}{8}$  (measures 3-4),  $\frac{4}{4}$  (measures 5-6). Dynamics:  $\text{mp}$  (measure 1),  $\text{mp}$  (measure 2),  $\text{mp}$  (measure 3),  $\text{mp}$  (measure 4),  $\text{mp}$  (measure 5),  $\text{mp}$  (measure 6).

**bass drums**:  $\frac{4}{4}$  (measures 1-2),  $\frac{2}{8}$  (measures 3-4),  $\frac{4}{4}$  (measures 5-6). Dynamics:  $\text{mp}$  (measure 1),  $\text{f}$  (measure 2),  $\text{mp}$  (measure 3),  $\text{mp}$  (measure 4),  $\text{mp}$  (measure 5),  $\text{mp}$  (measure 6).

**baritone saxophone**:  $\frac{2}{8}$  (measures 1-2),  $\frac{4}{4}$  (measures 3-4),  $\frac{2}{8}$  (measures 5-6),  $\frac{4}{4}$  (measures 7-8). Dynamics:  $\text{fz}$  (with 6.5),  $\text{fz}$  (with 4.3). Articulation:  $\text{b}$  (measure 1),  $\text{b}$  (measure 2),  $\text{b}$  (measure 3),  $\text{b}$  (measure 4),  $\text{b}$  (measure 5),  $\text{b}$  (measure 6),  $\text{b}$  (measure 7),  $\text{b}$  (measure 8).

**electric guitar**:  $\frac{2}{8}$  (measures 1-2),  $\frac{4}{4}$  (measures 3-4),  $\frac{2}{8}$  (measures 5-6),  $\frac{4}{4}$  (measures 7-8). Dynamics:  $\text{p}$  (measure 1),  $\text{p}$  (measure 2),  $\text{p}$  (measure 3),  $\text{p}$  (measure 4),  $\text{p}$  (measure 5),  $\text{p}$  (measure 6),  $\text{p}$  (measure 7),  $\text{p}$  (measure 8).

**piano**:  $\frac{2}{8}$  (measures 1-2),  $\frac{4}{4}$  (measures 3-4),  $\frac{2}{8}$  (measures 5-6),  $\frac{4}{4}$  (measures 7-8). Dynamics:  $\text{L.V.}$  (measure 1),  $\text{L.V.}$  (measure 2),  $\text{L.V.}$  (measure 3),  $\text{L.V.}$  (measure 4),  $\text{L.V.}$  (measure 5),  $\text{L.V.}$  (measure 6),  $\text{L.V.}$  (measure 7),  $\text{L.V.}$  (measure 8).

**wood shakers**:  $\frac{2}{8}$  (measures 1-2),  $\frac{4}{4}$  (measures 3-4),  $\frac{2}{8}$  (measures 5-6),  $\frac{4}{4}$  (measures 7-8). Dynamics:  $\text{L.V.}$  (measure 1),  $\text{L.V.}$  (measure 2),  $\text{L.V.}$  (measure 3),  $\text{L.V.}$  (measure 4),  $\text{L.V.}$  (measure 5),  $\text{L.V.}$  (measure 6),  $\text{L.V.}$  (measure 7),  $\text{L.V.}$  (measure 8).

**wood blocks**:  $\frac{2}{8}$  (measures 1-2),  $\frac{4}{4}$  (measures 3-4),  $\frac{2}{8}$  (measures 5-6),  $\frac{4}{4}$  (measures 7-8). Dynamics:  $\text{mp}$  (with 4.3),  $\text{mp}$  (measure 1),  $\text{mp}$  (measure 2),  $\text{mp}$  (measure 3),  $\text{mp}$  (measure 4),  $\text{mp}$  (measure 5),  $\text{mp}$  (measure 6),  $\text{mp}$  (measure 7),  $\text{mp}$  (measure 8).

**bass drums**:  $\frac{2}{8}$  (measures 1-2),  $\frac{4}{4}$  (measures 3-4),  $\frac{2}{8}$  (measures 5-6),  $\frac{4}{4}$  (measures 7-8). Dynamics:  $\text{mp}$  (with 6.5),  $\text{f}$  (measure 1),  $\text{mp}$  (measure 2),  $\text{mp}$  (measure 3),  $\text{mp}$  (measure 4),  $\text{mp}$  (measure 5),  $\text{mp}$  (measure 6),  $\text{fp}$  (measure 7),  $\text{fp}$  (measure 8).

135       $\frac{2}{8}$        $\frac{2}{4}$        $\frac{2}{8}$        $\frac{2}{4}$        $\frac{2}{8}$

*baritone saxophone*      *electric guitar*      *piano*      *wood shakers*      *wood blocks*      *bass drums*

$\text{♩} = 72$

141      **12B**       $\frac{2}{8}$        $\frac{2}{4}$

*baritone saxophone*      *electric guitar*      *piano*      *wood shakers*      *wood blocks*      *bass drums*

**COLOR THREE**

144

**2**      **4**      **4**      **8**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

*ff*      *4.3*      *ff*

*p*      *3.2*      *pp*      *4.3*

*p*

*3.2*

*p*      *L.V.*

*mp*      *4.3*

*mp*      *12:11*      *mp*      *3.2*

148

**2**      **8**      **2**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

*ff*      *4.3*      *ff*

*p*

*p*      *4.3*

*p*      *L.V.*

*p*      *4.3*

*f*      *6.5*

*mp*

*mp*

*mp*

*p*

153

**2**

**16**

**4**

**2**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

*L.V.* **ff** — *L.V.*

*L.V.* **ff** — *L.V.*

*L.V.* **ff** — *L.V.*

**16:11**

**f**

**p**

**J = 80**

158

**2**

**2**

**3** **[13]**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

*ff* — **5.4**

**ff** — **8.5**

**mp** — **5.4**

**f**

**5.3**

**[COLOR TWO]**

*ff* — **3.2**

**mp** — **4.3**

**f**

*ff* — **3.2**

**p**

*ff* — **3.2**

**p**

*ff* — **3.2**

**mp**

**f**

**ff** — **3.2**

**ff**

**ff** — **3.2**

**ff** — **5.4**

163

3 4 3

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

167

3

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

$J = 96$

170 **14**

*baritone saxophone*

*air*

(*sempre ppp*)

**COLOR FOUR**

*Sua-----*

*Sua-----*

(*sempre ppp*)

*piano*

(*sempre ppp*)

*wood shakers*

(*sempre ppp*)

*wood blocks*

(*sempre ppp*)

**STYROFOAM**

*bass drums*

(*sempre ppp*)

172

**5**

**6**

*baritone saxophone*

*air*

*Sua-----*

*Sua-----*

*piano*

*wood shakers*

6:5

*wood blocks*

7:6

*bass drums*

174

**8**

**16**

**8**

baritone saxophone

This musical score page shows six staves for different instruments. The top staff is for the baritone saxophone, which has a melodic line with some grace notes and dynamic markings like 'air' and '3.2'. The second staff is for the electric guitar, featuring eighth-note patterns with 'Sva' markings and time signatures '8.5' and '3.2'. The third staff is for the piano, with a continuous eighth-note pattern. The fourth staff is for wood shakers, showing a simple eighth-note pattern. The fifth staff is for wood blocks, with a single eighth-note stroke. The bottom staff is for bass drums, with two strokes indicated.

electric guitar

piano

wood shakers

wood blocks

bass drums

177

**8**

**8**

**33**

baritone saxophone

This musical score page shows the same six instruments as the previous page. The baritone saxophone has a sustained note with a grace note. The electric guitar has a melodic line with 'Sva' markings and time signatures '6.5' and '3.2'. The piano has a sustained note with a grace note. The wood shakers have a sustained note with a grace note. The wood blocks have a sustained note with a grace note. The bass drums have two sustained notes with grace notes.

electric guitar

piano

wood shakers

wood blocks

bass drums

180      8

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

7

182      8      16      7      5

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

186

**16**      **8**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

**8**

189

**8**      **3**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

**3**

192      **16**      **7**      **8**  
*baritone saxophone*  
  
*electric guitar*  
*piano*  
*wood shakers*  
*wood blocks*  
*bass drums*

$\text{♩} = 48$   
 195      **5**      **15**      **16**      **7**      **2**      **46**  
*baritone saxophone*  
  
*electric guitar*  
**COLOR SIX**  
*piano*  
*wood shakers*  
*wood blocks*  
*bass drums*  
**STYROFOAM**

$\text{♩} = 64$

200  $\frac{2}{4}$  **16**  $\frac{8}{4}$

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

**COLOR ONE**

$\frac{8}{4}$   $\frac{2}{4}$

$\text{♩} = 96$

205  $\frac{8}{4}$   $\frac{3}{4}$   $\frac{7}{4}$  **17-A**  $\frac{3}{4}$   $\frac{5}{4}$

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

**COLOR FOUR**

*air*

(*sempre ppp*)

9.8  $f$

(*sempre ppp*)

(*sempre ppp*)

*Sust.*

(*sempre ppp*)

- L.V.

(*sempre ppp*)

**STYROFOAM**

(*sempre ppp*)

210 8 16

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

213 8 16

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

217

**5**      **3**      **16**      **8**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

221

**3**      **8**      **16**      **8**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

225      3            8            16            8

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

$\text{♩} = 96$

229      6            8 [17B]            16

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

$\text{♩} = 96$

229      6            8 [17B]            16

232

**8**

**8**

**5**

**16**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

236

**8**

baritone saxophone

electric guitar

piano

wood shakers

wood blocks

bass drums

238

**8**

**3**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

This musical score page contains six staves. The top three staves are for 'baritone saxophone', 'electric guitar', and 'piano'. The bottom three staves are for 'wood shakers', 'wood blocks', and 'bass drums'. Measure 238 starts with a 3/8 time signature. The 'baritone saxophone' has a sustained note with a grace note and a dynamic of  $\frac{4}{3}$ . The 'electric guitar' has a sustained note with a grace note and a dynamic of  $\frac{4}{3}$ . The 'piano' has a sustained note with a grace note and a dynamic of  $\frac{3}{2}$ . Measures 239 and 240 continue with similar patterns. Measure 241 begins with a 4/8 time signature, indicated by a '4' above the staff and a '16' below it.

241

**4**

**3**

**16**

*baritone saxophone*

*electric guitar*

*piano*

*wood shakers*

*wood blocks*

*bass drums*

This continuation of the musical score follows the pattern established in measure 241. It includes the same six instruments: baritone saxophone, electric guitar, piano, wood shakers, wood blocks, and bass drums. The time signature changes from 4/8 to 3/16 in measure 241, which is explicitly marked with a '4' over the staff and a '16' under it. The instruments play sustained notes with grace notes and specific dynamics like  $\frac{4}{3}$  and  $\frac{3}{2}$ .

*Jamaica Plain*  
December 2013 - February 2014

# 9

## *Invisible Cities (i): Zaira (2014)*

A composition for eight players

Premièred by Ensemble Mosaik  
on Saturday October 4th, 2014  
in Paine Hall, Harvard University

# PREFACE

## 1

*In vain, great-hearted Kublai, shall I attempt to describe Zaira, city of high bastions. I could tell you how many steps make up the streets rising like stairways, and the degree of the arcades' curves, and what kind of zinc scales cover the roofs; but I already know this would be the same as telling you nothing. The city does not consist of this, but of relationships between the measurements of its space and the events of its past: the height of a lamppost and the distance from the ground of a hanged usurper's swaying feet; the line strung from the lamppost to the railing opposite and the festoons that decorate the course of the queen's nuptial procession; the height of that railing and the leap of the adulterer who climbed over it at dawn; the tilt of a guttering and a cat's progress along it as he slips into the same window; the firing range of a gunboat which has suddenly appeared beyond the cape and the bomb that destroys the guttering; the rips in the fish net and the three old men seated on the dock mending nets and telling each other for the hundredth time the story of the gunboat of the usurper, who some say was the queen's illegitimate son, abandoned in his swaddling clothes there on the dock.*

*As this wave from memories flows in, the city soaks it up like a sponge and expands. A description of Zaira as it is today should contain all of Zaira's past. The city, however, does not tell its past, but contains it like the lines of a hand, written in the corners of the streets, the gratings of the windows, the banisters of the steps, the antennae of the lightning rods, the poles of the flags, every segment marked in turn with scratches, indentations, scrolls.*

- Italo Calvino, *Invisible Cities*

## 2 Instrumentation

- Flute, with brazil nut shaker
- Oboe
- Clarinet in b-flat, with brazil nut shaker
- Percussion

Metals      tam-tam      low cymbal      middle cymbal      high cymbal      brake drum

Woods      bamboo windchimes      guero      tambourine

Drums      bass drum      kick drum      low tom      high tom

Mallets: hard sticks or bare hands, wire brushes, superballs

- Piano

Prepare the lowest and highest octaves with any combination of felt, tape or rubber to dampen and distort the timbre of the strings.

Guero passages should be played with a piece of hard paper or plastic, on the keys. The register of the motions is left to the performer.

- Violin, with brazil nut shaker
- Viola, with brazil nut shaker
- Cello

*Invisible Cities (i):*

# ZAIRA

for Ensemble Mosaik

Josiah Wolf Oberholtzer (1984)

$\text{J} = 72$

3  
A

Musical score for section A (measures 1-6). The score consists of eight staves:

- Flute
- Oboe
- Clarinet in B-flat
- Metals
- Woods
- Drums
- Piano
- Cello

The tempo is  $\text{J} = 72$ . The instrumentation includes Flute, Oboe, Clarinet in B-flat, Metals, Woods, Drums, Piano, and Cello.

$\text{J} = 48$

2  
B

3

2

[keyclick]

Musical score for section B (measures 7-12). The score consists of nine staves:

- Fl.
- Ob.
- Cl. in B-flat
- Metals
- Drums
- Pf.
- Vn.
- Va.
- Vc.

The tempo is  $\text{J} = 48$ . The instrumentation includes Flute, Oboe, Clarinet in B-flat, Metals, Drums, Piano, Violin, Viola, and Cello.

Performance instructions and dynamics include:

- Flute: 'keyclick' (multiple instances), 'brush' (multiple instances), '(L.V.)' (multiple instances).
- Oboe: 'keyclick' (multiple instances), 'pp' (multiple instances).
- Clarinet in B-flat: 'keyclick' (multiple instances), 'pp' (multiple instances), '5.4' (multiple instances).
- Metals: 'brush' (multiple instances), '(L.V.)' (multiple instances).
- Drums: 'brush' (multiple instances), 'pp' (multiple instances), '(L.V.)' (multiple instances).
- Piano: 'prepared' (multiple instances), '4r' (multiple instances), '7.6' (multiple instances), 'ppp' (multiple instances), '4.3' (multiple instances), 'f' (multiple instances).
- Violin: 'flautando' (multiple instances), '9.8' (multiple instances).
- Viola: 'flautando' (multiple instances), '9.8' (multiple instances).
- Cello: 'flautando' (multiple instances), '9.8' (multiple instances).

13       $\frac{4}{4}$

Fl. II 7  $\xrightarrow[3.2]{ppp}$   
Ob. II  $\xrightarrow[4.3]{}$   
Cl. in B-flat II 7  $\xrightarrow[p]{}$

Metals  $\xrightarrow[brush]{mp}$  (L.V.)  $\xrightarrow[p]{pp}$  (L.V.)  $\xrightarrow[brush]{mf}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[brush]{mf}$

Drums  $\xrightarrow[pp]{p}$  (L.V.)  $\xrightarrow[p]{mp}$  (L.V.)  $\xrightarrow[pp]{pp}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[mf]{p}$

Pf.  $\xrightarrow[f]{p}$   $\xrightarrow[5.4]{pp}$   $\xrightarrow[p]{pp}$   $\xrightarrow[5.3]{ppp}$   $\xrightarrow[5.4]{ppp}$   $\xrightarrow[7.6]{mf}$

Vn.  $\xrightarrow[5.4]{pp}$   $\xrightarrow[5.3]{pp}$   $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Va.  $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Vc.  $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

16       $\frac{3}{4}$

Fl. II 7  $\xrightarrow[3.2]{ppp}$   
Ob. II  $\xrightarrow[4.3]{}$   
Cl. in B-flat II 7  $\xrightarrow[p]{}$

Metals  $\xrightarrow[brush]{}$  (L.V.)  $\xrightarrow[p]{pp}$  (L.V.)  $\xrightarrow[brush]{}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[brush]{}$

Drums  $\xrightarrow[pp]{p}$  (L.V.)  $\xrightarrow[p]{mp}$  (L.V.)  $\xrightarrow[pp]{pp}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[mf]{p}$

Pf.  $\xrightarrow[prepared]{f}$   $\xrightarrow[5.4]{pp}$   $\xrightarrow[p]{pp}$   $\xrightarrow[5.3]{ppp}$   $\xrightarrow[5.4]{ppp}$   $\xrightarrow[7.6]{mf}$

Vn.  $\xrightarrow[5.4]{pp}$   $\xrightarrow[5.3]{pp}$   $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Va.  $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Vc.  $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

2       $\frac{4}{4}$

Fl. II 7  $\xrightarrow[p]{ppp}$   $\xrightarrow[3.2]{ppp}$   
Ob. II  $\xrightarrow[5.4]{ppp}$   $\xrightarrow[3.2]{ppp}$   
Cl. in B-flat II 7  $\xrightarrow[p]{ppp}$

Metals  $\xrightarrow[brush]{}$  (L.V.)  $\xrightarrow[p]{pp}$  (L.V.)  $\xrightarrow[brush]{}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[brush]{}$

Drums  $\xrightarrow[pp]{p}$  (L.V.)  $\xrightarrow[p]{pp}$  (L.V.)  $\xrightarrow[pp]{pp}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[mp]{p}$

Pf.  $\xrightarrow[prepared]{f}$   $\xrightarrow[5.4]{pp}$   $\xrightarrow[p]{pp}$   $\xrightarrow[6.5]{ppp}$   $\xrightarrow[pp]{pp}$   $\xrightarrow[p]{p}$

Vn.  $\xrightarrow[5.4]{pp}$   $\xrightarrow[5.3]{pp}$   $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Va.  $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Vc.  $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

3       $\frac{4}{4}$

Fl. II 7  $\xrightarrow[5.4]{ppp}$   $\xrightarrow[3.2]{ppp}$   
Ob. II  $\xrightarrow[7.6]{ppp}$   $\xrightarrow[5.4]{ppp}$   $\xrightarrow[3.2]{ppp}$   
Cl. in B-flat II 7  $\xrightarrow[p]{ppp}$

Metals  $\xrightarrow[brush]{}$  (L.V.)  $\xrightarrow[p]{pp}$  (L.V.)  $\xrightarrow[brush]{}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[brush]{}$

Drums  $\xrightarrow[pp]{p}$  (L.V.)  $\xrightarrow[p]{pp}$  (L.V.)  $\xrightarrow[pp]{pp}$  (L.V.)  $\xrightarrow[p]{ppp}$  (L.V.)  $\xrightarrow[mp]{p}$

Pf.  $\xrightarrow[prepared]{f}$   $\xrightarrow[5.4]{pp}$   $\xrightarrow[p]{pp}$   $\xrightarrow[6.5]{ppp}$   $\xrightarrow[pp]{pp}$   $\xrightarrow[p]{p}$

Vn.  $\xrightarrow[5.4]{pp}$   $\xrightarrow[5.3]{pp}$   $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Va.  $\xrightarrow[pp]{pp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

Vc.  $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[ppp]{ppp}$   $\xrightarrow[pp]{pp}$

27

**5** **16** **2** **4** **4** **3** **8** **5** **16**

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Va.

Vc.

$\text{J} = 72$

29

**2** **C** **3** **4** **2**

Fl.

Ob.

Cl. in B-flat

Metals

Woods

Drums

Pf.

Vn.

Va.

Vc.

34      5      3      3      3

Fl.      shaker  
Ob.      ram / slap 5A  
Cl. in B-flat      shaker  
Woods  
Drums      brush  
Pf.      fast  
Vn.      flautando 4.3  
Va.      shaker  
Vc.      col legno

35      8      3      3      3

Fl.      ram / slap 5A  
Ob.      mp  
Cl. in B-flat      7.5 p  
Woods      7.6  
Drums      L.V.  
Pf.      3.2  
Vn.      7.  
Va.      5A p  
Vc.      3.2 pp

36      16      4      4

Fl.      ram / slap 5A  
Ob.      mp  
Cl. in B-flat      7.5 p  
Woods      3.2 ff  
Drums      L.V.  
Pf.      fast  
Vn.      7.  
Va.      5A p  
Vc.      3.2 pp

37      8      3      3      3

Fl.      ram / slap 5A  
Ob.      mp  
Cl. in B-flat      7.5 p  
Woods      7.6  
Drums      L.V.  
Pf.      slow  
Vn.      7.  
Va.      5A p  
Vc.      3.2 pp

Musical score for orchestra and percussion, page 10, measures 11-12. The score includes parts for Flute (Fl.), Oboe (Ob.), Clarinet in B-flat (Cl. in B-flat), Woodwinds (Woods), Drums, Piccolo (Pf.), Violin (Vn.), Cello (Va.), and Bass (Vc.). The score features complex rhythmic patterns and dynamic markings such as *p*, *pp*, *fff*, and *ffff*. Various performance techniques are indicated, including *shaker*, *ram / slap*, *blautando*, and *guerra*. Measure 11 ends with a forte dynamic (*ffff*) and measure 12 begins with a piano dynamic (*p*). Measures 11 and 12 both conclude with dynamic markings like *ppp* and *pp*.

2 3  
 Fl. shaker  
 Ob. ram / slap  
 Cl. in B-flat ram / slap  
 Woods ram / slap  
 Drums (L.V.)  
 Pf. fast  
guero  
 Vn. slow  
guero  
 Va.  
 Vc.

4 3  
 Fl. ram / slap  
 Ob. ram / slap  
 Cl. in B-flat ram / slap  
 Woods (L.V.)  
 Drums (L.V.)  
 Pf. slow  
guero  
 Vn. shaker  
 Va.  
 Vc.

5 4  
 Fl. pp  
 Ob. pp  
 Cl. in B-flat pp  
 Metals pp  
 Woods fff  
 Drums brush  
 Pf. slow  
guero  
 Vn. overpressure  
 Va. shaker  
 Vc. overpressure

$\downarrow = 84$   
**16** 5 **D** 2 3 5

55 6 5 3 2 16

60 4 2 3 5 16

66 E 3 8 16

Fl. Ob. Cl. in B-flat Metals Woods Drums Pf. Vn. Va. Vc.

*ram/slap* *keyclick* 4.3 *p* *pp* *3.2* *pp*  
*p* *f* *p* *f=p* *keyclick* *pp*  
*mf* *p* *6.5* *pp*  
*ff* *mf* *7.5*  
*ff* *7.5*  
*prepared* *ppp* *prepared* *slow* *guero*  
*ppp* *p* *ppp*  
*ppp*

70 4 2 3

Fl. Ob. Cl. in B-flat Metals Drums Pf. Vn. Va. Vc.

*keyclick* *airtone* *ram/slap* 7.6 *airtone* *ram/slap* *keyclick* *ppp*  
*p* *ppp* *ppp* *mp* *mp* *ppp* *ppp* *ppp*  
*mf* *mf* *mf* *p* *f* *p* *mf* *mf*  
*keyclick* *airtone* *ram/slap* 7.6 *airtone* *ram/slap* *keyclick* *ppp*  
*pp* *ppp* *ppp* *mp* *p* *ppp* *ppp* *ppp*  
*brush* *brush* *brush* *brush* *brush* *brush* *brush* *brush*  
*f* *ff* *ff* *ff* *ff* *ff* *ff* *ff*  
*9.8* *f* *mf* *slow* *guero* *p* *pp* *6.5* *3.2*  
*brush* *brush* *brush* *brush* *brush* *brush* *brush* *brush*  
*p* *p* *p* *p* *p* *p* *p* *p*  
*overpressure* *overpressure* *overpressure* *overpressure* *overpressure* *overpressure* *overpressure* *overpressure*  
*p* *pp* *pp* *pp* *pp* *pp* *pp* *pp*

74

Fl. II 7- 7.5 keyclick  
Ob. II p PPP 5.4  
Cl. in B-flat II 7- 7.6 PPP  
Metals brush  
Drums brush fff  
Pf. ppp 6.5  
Vn. p overpressure  
Va. p overpressure  
Vc. p overpressure fff

75

Fl. 5.4 keyclick  
Ob. 6.5 4.3 keyclick 3.2 PPP  
Cl. in B-flat 7- 7.6 PPP  
Metals brush  
Drums brush f 3.2  
Pf. prepared p 7.6 ppp  
Vn. 4.3  
Va. 5.3  
Vc. 4.3

76

Fl. 5.4 keyclick ram/slap 5.4 airtone 6.5  
Ob. 6.5 4.3 keyclick 3.2 PPP  
Cl. in B-flat 7- 7.6 PPP  
Metals brush  
Drums brush f 3.2  
Pf. prepared p 7.6 ppp  
Vn. 4.3  
Va. 5.3  
Vc. 4.3

77

Fl. 5.4 keyclick ram/slap 5.4 airtone 6.5  
Ob. 6.5 4.3 keyclick 3.2 PPP  
Cl. in B-flat 7- 7.6 PPP  
Metals brush  
Drums brush f 3.2  
Pf. prepared p 7.6 ppp  
Vn. 4.3  
Va. 5.3  
Vc. 4.3

82

**A**

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Va.

keyclick 5.4 airtone ram/slap 6.5

keyclick 4.3 airtone pp

airtone 3.2 pp

airtone 3.2 pp

brush 3.2

brush

brush

fast guero 5.3

overpressure

overpressure

overpressure

86

**Fl.** *keyclick* 7.6 *ram / slap* *airtone* *ram / slap*

**Ob.** *ram / slap* 3.2 *keyclick* 3.2 *keyclick* 5.4 *ram / slap* *airtone* *ram / slap*

**Cl. in B-flat** *keyclick* 5.4 *ram / slap* 5.4 *ram / slap* *airtone* *ram / slap*

**Metals** 6.5 3.2 *brush* 9.7

**Drums** *ff* 8.7 *brush* 4.3

**Pf.** *prepared* 7.6 *pp* 4.3 *prepared* 5.4 *ppp* *prepared* 7.5 *p* *prepared* 6.5 *f* 8.7 *f*

**Vn.** *overpressure* 3.2 *overpressure* *overpressure* *overpressure*

**Va.** *overpressure* *overpressure* *overpressure* *overpressure*

**Vc.** *overpressure* 3.2 *overpressure* *overpressure* *overpressure*

91

3 8 3 8 4 4

Fl. *[ram/slap]* *[airtone]* *[ppp]* *[pp]* *[ram/slap]* *[keyclick]* *[5.4]* *[5.4]* *[5.4]* *[keyclick]*  
*[mf]* *[ppp]* *[mp]* *[p]* *[ppp]* *[mf]* *[mp]* *[p]* *[ppp]* *[p]*

Ob. *[3.2]* *[4.5]* *[airtone]* *[p]* *[pp]* *[keyclick]* *[5.3]* *[4.5]* *[keyclick]*  
*[ff]* *[pp]*

Cl. in B-flat *[airtone]* *[ppp]* *[ram/slap]* *[keyclick]* *[ppp]* *[mf]* *[ram/slap]* *[7.6]* *[keyclick]*  
*[ppp]* *[mp]* *[p]* *[ppp]* *[p]* *[ppp]* *[p]*

Metals *[fff]* *[mf]* *[brush]* *[3.2]* *[f]* *[4.3]* *[3.2]* *[brush]*  
*[mp]*

Drums *[fff]* *[mf]* *[3.2]* *[mf]* *[brush]* *[5.3]* *[mf]* *[5.4]* *[mf]* *[brush]*  
*[mf]*

Pf. *[mf] > o* *[ff] 6.5* *[p] > f* *[p] - f* *[f > p] 5.4* *[mf] > o* *[slow]* *[guar]* *[mp]*  
*[3.2]* *[5.4]* *[7.6]* *[ff]* *[3.2]* *[5.4]* *[ff]* *[f > p]*

Vn. *[p]* *[pp]* *[b (1/2)]* *[overpressure]* *[fff]*

Va. *[fff]* *[overpressure]*

Vc. *[fff]* *[overpressure]*

96 **2**

Fl. *keyclick* *4toss*  
*p* *ppp*

Ob. *airtone* *f* *p*

Cl. in B-flat *keyclick* *ram/slap* *6.5* *3.2*  
*p* *ppp* *4.3 ppp* *mp*

Metals *keyclick* *ram/slap* *4.3* *5.4 ppp*  
*p* *ppp* *6.5 p* *mp*

Drums *brush* *ff*  
*3.2* *3.2* *7.5*

PF. *prepared* *p* *f-p*  
*prepared* *pp* *slow* *guero*  
*p-f* *mp* *mf* *fast* *guero*  
*p* *pp* *pp* *p* *pp*

Vn. *4toss*  
*ppp*

Va. *pp*

Vc. *4toss* *ppp* *p* *3.2 mp*

*J = 48*

103 **3** [F1] **4**

Fl. *shaker* *ppp*

Ob. *keyclick* *7.5* *f* *p*

Cl. in B-flat *keyclick* *7.5* *4toss*  
*p-f* *ppp* *f*

Metals *brush* *fff p-f* *ppp*

Drums *brush* *fff* *p* *ppp*

PF. *fff* *ppp*

Vn. *shaker* *ppp* *4.3 ppp* *overpressure* *3.2*  
*p* *p* *ppp* *mp* *mp*

Va. *shaker* *ppp* *4.3 pp* *overpressure* *5.4* *shaker*  
*p* *p* *pp* *mp* *mp* *p*

Vc. *ppp*

*J = 48*

103 **3**

Fl. *shaker* *ppp*

Ob. *keyclick* *7.5* *p* *ppp*

Cl. in B-flat *keyclick* *7.5* *airtone* *p* *ppp* *keyclick*  
*keyclick* *airtone* *3.2*

Metals *brush* *fff p-f* *ppp*

Drums *brush* *fff* *p* *ppp*

PF. *fff* *ppp*

Vn. *shaker* *ppp* *4.3 ppp* *overpressure* *3.2*  
*p* *p* *ppp* *mp* *mp*

Va. *shaker* *ppp* *4.3 pp* *overpressure* *5.4* *shaker*  
*p* *p* *pp* *mp* *mp* *p*

Vc. *ppp*

107  $\frac{3}{4}$  3  $\frac{4}{4}$  4  $\frac{2}{4}$  5  $\frac{16}{16}$

Fl.  
Ob.  
Cl. in B-flat  
Metals  
Drums  
Pf.  
Vn.  
Va.

112  $\frac{6}{8}$  3  $\frac{4}{4}$  3  $\frac{8}{8}$  6  $\frac{8}{8}$

Fl.  
Ob.  
Cl. in B-flat  
Metals  
Drums  
Pf.  
Vn.  
Va.  
Vc.

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Vc.

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Va.

Vc.

126 **3** **4** **5** **16** **3** **8** **F2** **6** **8**  
 ♩ = 48

Fl.  
 Ob.  
 Cl. in B-flat  
 Metals  
 Drums  
 Pf.  
 Vn.  
 Va.  
 Vc.

131 **2** **4** **16** **4** **3** **8**

Fl.  
 Ob.  
 Cl. in B-flat  
 Metals  
 Drums  
 Pf.  
 Vn.  
 Va.  
 Vc.

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Va.

Vc.

*airtone*

*shaker*

*keyclick*

*brush*

*superball*

*superball*

*brush*

*shaker*

*keyclick*

*overpressure*

*overpressure*

*overpressure*

*overpressure*

*overpressure*

J = 96

G

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Va.

Vc.

*keyclick*

*overpressure*

*overpressure*

*overpressure*

*overpressure*

*overpressure*

145       $\frac{3}{4}$

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Va.

Vc.

5       $\frac{3}{4}$

3       $\frac{5}{4}$

5       $\frac{4}{4}$

150       $\frac{2}{4}$

Fl.

Ob.

Cl. in B-flat

Metals

Drums

Pf.

Vn.

Va.

Vc.

4       $\frac{4}{4}$

3       $\frac{3}{4}$

2       $\frac{2}{4}$

155

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *f* *p* 7.6 *mf* 7.0 *f* 4.3 *ff* 4.3 *mf* 3.2  
Pf. *p*  
Vn. *p*  
Va. *ppp*  
Vc.

3 4 3 4 3 5 16 3 5 16

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *f* *p* 7.6 *mf* 7.0 *f* 4.3 *ff* 4.3 *mf* 3.2  
Pf. *p*  
Vn. *p*  
Va. *ppp*  
Vc.

3 8 3 4 3 2 4 4 4 5 16

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *f* *p* 7.6 *mf* 7.0 *f* 4.3 *ff* 4.3 *mf* 3.2  
Pf. *p*  
Vn. *p*  
Va. *ppp*  
Vc.

3 8 3 5 16

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *f* *p* 7.6 *mf* 7.0 *f* 4.3 *ff* 4.3 *mf* 3.2  
Pf. *p*  
Vn. *p*  
Va. *ppp*  
Vc.

overpressure  
*fff*  
overpressure  
*fff*  
overpressure  
*fff*

161

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *ff* *mf* 4.3 *fff* *p* 3.2 *f* 7.6 *f* *p* *ff* 5.4  
Pf. *p* 5.4 *pp*  
Vn. *p*  
Va. *ppp*  
Vc. *p* 4.5 *f* *p*

3 4 3 8 2 4 4 4 5 16

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *ff* *mf* 4.3 *fff* *p* 3.2 *f* 7.6 *f* *p* *ff* 5.4  
Pf. *p* 5.4 *pp*  
Vn. *p*  
Va. *ppp*  
Vc. *p* 4.5 *f* *p*

3 8 2 4 4 4 4 5 16

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *ff* *mf* 4.3 *fff* *p* 3.2 *f* 7.6 *f* *p* *ff* 5.4  
Pf. *p* 5.4 *pp*  
Vn. *p*  
Va. *ppp*  
Vc. *p* 4.5 *f* *p*

3 8 2 4 4 4 4 5 16

Fl. *p*  
Ob. *p*  
Cl. in B-flat *p*  
Metals  
Drums *ff* *mf* 4.3 *fff* *p* 3.2 *f* 7.6 *f* *p* *ff* 5.4  
Pf. *p* 5.4 *pp*  
Vn. *p*  
Va. *ppp*  
Vc. *p* 4.5 *f* *p*

106       $\frac{2}{4}$       8      5       $\frac{5}{16}$        $\frac{5}{16}$  H       $\frac{4}{4}$

$\text{Fl.}$        $\text{Ob.}$        $\text{Cl. in B-flat}$        $\text{Metals}$        $\text{Woods}$        $\text{Drums}$        $\text{Pf.}$        $\text{Vn.}$        $\text{Va.}$        $\text{Vc.}$

107       $\frac{2}{4}$       3       $\frac{4}{4}$

$\text{Fl.}$        $\text{Ob.}$        $\text{Cl. in B-flat}$        $\text{Woods}$        $\text{Drums}$        $\text{Pf.}$        $\text{Vn.}$        $\text{Va.}$        $\text{Vc.}$

177      **5**      **16**      **4**      **2**      **3**      **5**      **16**      **3**

**Fl.** *shaker*  
**Ob.** *keyclick* 7.5 *pp* *ram/slap* 9.8 *ppp*  
**Cl. in B-flat** *shaker* 6.5 *pp* *ppp*  
**Woods** *brush* 4.3 *mf* *p* *mp* 4.3  
**Drums** *brush* *mp*  
**Pf.** *prepared* 7.6 *f* *p* *pp* *prepared* 7.6 *f* *p* *mf* *p*  
**Vn.** *shaker* 7.6 *p* *pp* *shaker* 7.6 *p* *pp* *pp*  
**Va.** *flautando* 3.2 *pp* *shaker* 5.4 *p* *pp* *overpressure* 5.4 *p* *pp*  
**Vc.** *pp* *trill* *pp* *trill* *pp*

*j = 48*      **fff**      **I**      **5**      **16**      **3**

**Fl.** *shaker* 5.4 *mp*  
**Ob.** *keyclick* *ppp*  
**Cl. in B-flat** *shaker* 5.4 *p* *ppp* *shaker* 4.3  
**Metals** *superball* *fff* *pp*  
**Woods** *brush* *mf* *mf*  
**Drums** *ffff* *pp*  
**Pf.** *fast guero* *mf* *slow guero* *mp* *ppp* *p*  
**Vn.** *flautando* 5.4 *p* *shaker* *p* *pp* *overpressure* *mf*  
**Va.** *flautando* *ppp* *col legno* *p* *shaker* 5.3 *p* *ppp* *overpressure* *pp*  
**Vc.** *ppp* *3.2* *ppp* *p* *ffff* *f* *pp* *overpressure* 5.4 *mf*

189       $\frac{3}{8}$        $\frac{6}{8}$        $\frac{2}{4}$        $\frac{6}{8}$        $\frac{3}{8}$

Fl.      Ob.      Cl. in B-flat      Metals      Woods      Pf.      Vn.      Va.      Vc.

190       $\frac{3}{4}$        $\frac{4}{4}$        $\frac{3}{8}$        $\frac{5}{16}$        $\frac{2}{4}$

Fl.      Ob.      Cl. in B-flat      Metals      Woods      Drums      Pf.      Vn.      Va.      Vc.

200  $\frac{4}{4}$

Fl. *airtone* *ppp*

Ob. *airtone* *p*

Cl. in B-flat *airtone* *ppp*

Metals *superball* *ppp*

Woods *brush* *pp*

Metals *superball* *p*

Pf. *p*

Vn. *overpressure* *mf*

Va. *shaker* *pp*

Vc. *overpressure* *3.2* *p*

*overpressure* *5.4* *mf*

*shaker* *ppp* *p*

206  $\frac{4}{4}$

Fl. *airtone* *p*

Ob. *airtone* *ppp*

Cl. in B-flat *airtone* *p*

Metals *brush* *ppp*

Woods *superball* *pp*

Drums *brush* *p*

Metals *superball* *p*

Pf. *ppp*

Vn. *overpressure* *s* *mp*

Va. *overpressure* *4.3* *mf*

Vc. *overpressure* *4.3* *mp*

*shaker* *pp*

Vn. *overpressure* *s* *mf*

Va. *overpressure* *4.3* *p*

Vc. *overpressure* *4.3* *mf*

*shaker* *pp*

*overpressure* *s* *mp*

*overpressure* *4.3* *mf*

210       $\frac{2}{4}$        $\frac{4}{4}$        $\frac{2}{4}$        $\frac{5}{8}$        $\frac{2}{4}$

Fl.      shaker  
Ob.      airtone  
Cl. in B-flat      shaker  
Metals      superball  
Woods  
Drums      4.3

Pf.  
Vn.      shaker  
Va.  
Vc.      cod legno, overpressure  
J = 84

215       $\frac{5}{16}$        $\frac{3}{8}$  [J]       $\frac{2}{4}$        $\frac{3}{8}$        $\frac{2}{4}$

Fl.      shaker  
Cl. in B-flat  
Metals      superball  
Drums      fff, superball  
Pf.      prepared  
Vn.      shaker  
Va.

222

$\text{J} = 96$   
3 [K]

Musical score for measures 222-223. The score includes parts for Flute (Fl.), Clarinet in B-flat (Cl. in B-flat), Metals, Drums, Piano (Pf.), Violin (Vn.), and Cello (Va.). Measure 222 consists of sustained notes. Measure 223 begins with sustained notes followed by a dynamic section for Metals, Drums, and Piano. The Metals part has a dynamic of  $fff$ , a duration of 4.5 measures, and a dynamic of  $ppp$ . The Drums part has a dynamic of  $fff$ . The Piano part includes prepared piano techniques:  $pp$ ,  $prepared$  (with a grace note),  $pp$ , and  $ffff$ . The Vn. and Va. parts also have sustained notes.

229

Musical score for measure 229. The score includes parts for Metals and Piano (Pf.). The Metals part consists of a continuous series of eighth-note strokes. The Pf. part has a dynamic of  $f$ .

237

Musical score for measure 237. The score includes parts for Metals and Piano (Pf.). The Metals part consists of a continuous series of eighth-note strokes. The Pf. part has a dynamic of  $f$ .

This page intentionally left blank.

# 10

## *Invisible Cities (ii): Armilla* (2015)

A composition for viola duet

Premièred by Elizabeth Weisser &  
John Pickford Richards  
on Saturday February 7th, 2015  
in Paine Hall, Harvard University

# PREFACE

## 1

*Whether Armilla is like this because it is unfinished or because it has been demolished, whether the cause is some enchantment or only a whim, I do not know. The fact remains that it has no walls, no ceilings, no floors; it has nothing that makes it seem a city except the water pipes that rise vertically where the houses should be and spread out horizontally where the floors should be: a forest of pipes that end in taps, showers, spouts, overflows. Against the sky a lavabo's white stands out, or a bathtub, or some other porcelain, like late fruit still hanging from the boughs. You would think that the plumbers had finished their job and gone away before the bricklayers arrived; or else their hydraulic systems, indestructable, had survived a catastrophe, an earthquake, or the corrosion of termites.*

*Abandoned before or after it was inhabited, Armilla cannot be called deserted. At any hour, raising your eyes among the pipes, you are likely to glimpse a young woman, or many young women, slender, not tall of stature, luxuriating in the bathtubs or arching their backs under the showers suspended in the void, washing or drying or perfuming themselves, or combing their long hair at a mirror. In the sun, the threads of water fanning from the showers glisten, the jets of the taps, the spurts, the splashes, the sponges' suds.*

*I have come to this explanation: the streams of water channeled in the pipes of Armilla have remained in the possession of nymphs and naiads. Accustomed to traveling along underground veins, they found it easy to enter the new aquatic realm, to burst from multiple fountains, to find new mirrors, new games, new ways of enjoying the water. Their invasion may have driven out the human beings, or Armilla may have been built by humans as a votive offering to win the favor of the nymphs, offended at the misuse of the waters. In any case, now they seem content, these maidens: in the morning you hear them singing.*

- Italo Calvino, *Invisible Cities*

## 2

*The dunes ran inland, low and grassy, for half a mile or so, and then there were lagoons, thick with sedge and saltreeds, and beyond those, low hills lay yellow-brown and empty out of sight. Beautiful and desolate was Selidor. Nowhere on it was there any mark of man, his work or habitation. There were no beasts to be seen, and the reed-filled lakes bore no flocks of gulls or wild geese or any bird.*

- Ursula Le Guin, *The Farthest Shore*

## 3 Bow contact points

The current position along the bow as it contacts the strings is indicated with fractions, where 0/1 indicates the frog, and 1/1 indicates the tip of the bow. Continuous bowing is shown by lines connecting bow contact fractions.

## 4 Dynamics

Dynamics are always in terms of effort, not effect. When bowing very quickly and with strong dynamic, the effect should be a traditional *forte*. Likewise, when bowing slowly and with a light dynamic, the effect should be a traditional *piano*. Slow bowing with strong dynamic should result in various colors of scratch, while fast bowing with light pressure should give various qualities of flautando (depending of course on where on the string the bow is contacting).

## 5 String contact points

**D.P.** Dietro ponticello: behind the bridge.

**M.S.P.** Molto sul ponticello

**S.P.** Sul ponticello

**Ord.** Ordinario

**S.T.** Sul tasto

**M.S.T.** Molto sul tasto

When bowing behind the bridge, the fingering staff switches to percussion clef. The behind-the-bridge string to bow on are given by the four spaces of the five-line-staff, with string *IV* being the lowest space and *I* the highest.

## 6 Other techniques

### Across-the-string tremoli

Indicated by traditional tremolo hashes on the bow tablature's rhythm staff

### Along-the-string tremoli

Indicated by zigzag bow tablature glissandi.

### Thrown bow

Indicated by dashed bow tablature glissandi

### Pizzicati

Indicated with a cross notehead in the tablature staff.

### Accents

Accents in the bow tablature staff indicate a sudden staccato increase in bow pressure.

Tremoli, both across- and along-the-string, should be very tight. When the two techniques appear simultaneously, the resulting motion is tightly circular bowing.

*Invisible Cities (i):*  
**ARMILLA**  
*(a botanical survey of the uninhabited southern isles)*  
 for Elizabeth Weisser and John Pickford Richards

Josiah Wolf Oberholtzer (1984)

$\text{♩} = 36$

**A** Far Sorr

*D.P.*

Viola 1

Viola 2

*ORD.* ..... *S.T.*

**3** **8** **8**

Va. 1

S.T.

Va. 2

*M.S.T.*

**7** **3** **2** **5**

Va. 1

Va. 2

*ORD.* ..... *S.T.*

11       $\frac{3}{8}$       S.T.

Va. 1

M.S.T. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{1}{4}$   $\frac{3}{8}$   $\frac{1}{2}$   $\frac{5}{8}$   $\frac{3}{4}$

Va. 2

M.S.T. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{1}{4}$   $\frac{3}{8}$   $\frac{1}{2}$   $\frac{5}{8}$   $\frac{3}{4}$

ORD. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{1}{4}$   $\frac{3}{8}$   $\frac{1}{2}$   $\frac{5}{8}$   $\frac{3}{4}$

S.T. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{1}{4}$   $\frac{3}{8}$   $\frac{1}{2}$   $\frac{5}{8}$   $\frac{3}{4}$

12       $\frac{2}{4}$       S.T.

15       $\frac{2}{4}$       ORD. ..... S.T.

Va. 1

ORD. .....  $\frac{7}{8}$   $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{1}{4}$

Va. 2

ORD. .....  $\frac{1}{4}$   $\frac{1}{8}$   $\frac{3}{8}$   $\frac{1}{4}$

S.T. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$

16       $\frac{5}{8}$       S.T.

18       $\frac{6}{8}$       ORD. ..... S.T.

Va. 1

ORD. .....  $\frac{7}{8}$   $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$   $\frac{1}{4}$   $\frac{3}{8}$   $\frac{1}{2}$   $\frac{5}{8}$   $\frac{3}{4}$   $\frac{7}{8}$   $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$

Va. 2

ORD. .....  $\frac{1}{4}$   $\frac{3}{8}$   $\frac{1}{2}$   $\frac{5}{8}$   $\frac{3}{4}$   $\frac{7}{8}$

S.T. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$

M.S.T. .....  $\frac{1}{1}$   $\frac{0}{1}$   $\frac{1}{8}$

20       $\frac{2}{4}$       M.S.T.

38

४५

M.S.T. ORD.

Va. 1

PIZZ. ( PIZZ. ) ( PIZZ. ) ( PIZZ. ) ( PIZZ. ) ( PIZZ. )

Va. 2

J = 72

25 5 B Selidor (i)

३४

24

♩ = 108

29

30

2 C Wellogy

33

*ORD.*  $\frac{3}{8}$

*M.S.P.*  $\frac{4}{5} \dots \frac{3}{5}$

*S.P.*  $\frac{3}{5} \dots \frac{4}{5}$

*S.T.*  $\frac{4}{5} \dots \frac{3}{5}$

*ORD.*  $\frac{1}{1} \dots \frac{1}{1}$

*S.T.*  $\frac{2}{5} \dots \frac{3}{5}$

37

*ORD.*  $\frac{1}{1} \dots \frac{1}{1}$

*(V)*  $\frac{4}{5} \dots \frac{1}{1}$

*V*  $\frac{4}{5} \dots \frac{1}{1}$

*ORD.*  $\frac{1}{1} \dots \frac{1}{1}$

*S.T.*  $\frac{2}{5} \dots \frac{3}{5}$

*(V)*  $\frac{1}{1} \dots \frac{1}{1}$

*V*  $\frac{4}{5} \dots \frac{3}{5}$

40

*ORD.*  $\frac{1}{1} \dots \frac{3}{5}$

*M.S.T.*  $\frac{2}{5} \dots \frac{1}{1}$

*S.T.*  $\frac{3}{5} \dots \frac{4}{5}$

*(S.T.)*  $\frac{1}{1} \dots \frac{2}{5}$

*M.S.P.*  $\frac{1}{1} \dots \frac{3}{5}$

*M.S.T.*  $\frac{4}{5} \dots \frac{1}{1}$

*S.P.*  $\frac{3}{5} \dots \frac{4}{5}$

*S.T.*  $\frac{1}{1} \dots \frac{4}{5}$

$\text{♩} = 36$   
 44 **D** The Long Dune (i)

*D.P.*

Va. 1

Va. 2

5

Va. 1

Va. 2

3

2

3

(D.P.)

Va. 1

Va. 2

54 **3**

Va. 1

Va. 2

(D.P.)

**6**

**8**

57 **3**

**2**

**3**

Va. 1

Va. 2

ORD. ----->

60 **6**

(D.P.)

Va. 1

Va. 2

**3**

S.T. -----> M.S.T.

**8**

63       $\frac{8}{3}$

Va. 1

(D.P.)

M.S.T.

ORD.

Va. 2

64       $\frac{8}{3}$

Va. 1

Va. 2

66       $\frac{4}{2}$

Va. 1

ORD.

S.T.

Va. 2

69       $\frac{2}{3}$

Va. 1

ORD.

S.T.

M.S.T.

Va. 2

72

 $\frac{2}{4}$ 

5

 $\text{J} = 72$ 

8

E Selidor (ii)

S.T. .....  $\rightarrow$  M.S.T.

ORD.

Va. 1

Va. 2

M.S.T.

ORD.

(V)

V

6.5

6.5

75

3

5

ORD.)  $\rightarrow$  S.P. .....  $\rightarrow$  ORD. (V)

ORD.)  $\rightarrow$  M.S.P. .....  $\rightarrow$  S.P. .....  $\rightarrow$  ORD. (V)

ORD.)  $\rightarrow$  M.S.P. .....  $\rightarrow$  ORD. (V)

ORD.)

S.P. (V)

ORD. (V)

ORD.)

M.S.P.

S.P. (V)

ORD. (V)

4.5

4.5

79

 $\frac{3}{4}$ 

3

2

5

ORD. .....  $\rightarrow$  S.P. .....  $\rightarrow$  ORD. (V)

ORD.) (V) .....  $\rightarrow$  M.S.P. .....  $\rightarrow$  ORD.

S.P. (V) .....  $\rightarrow$  ORD. (V)

ORD.) (V) .....  $\rightarrow$  M.S.P. (V)

ORD.

S.P. (V)

ORD. (V)

ORD.) (V)

M.S.P.

S.P. (V)

ORD. (V)

ORD.) (V)

3.2

3.2

3.2

83

6

$J = 108$   
**F** The Isle  
 Of the Ear

Violin 1 (Va. 1) and Violin 2 (Va. 2) parts. Measure 83 starts with  $\frac{5}{8}$  time, dynamic *M.S.P.*, and a rhythmic pattern of eighth and sixteenth notes. Measure 84 begins with  $\frac{3}{5}$  time, dynamic *S.P.*, and continues the rhythmic pattern. Measure 85 starts with  $\frac{3}{5}$  time, dynamic *mfp*, and continues the pattern. Measure 86 begins with  $\frac{3}{5}$  time, dynamic *mp*, and continues the pattern.

86

3

8

2

Violin 1 (Va. 1) and Violin 2 (Va. 2) parts. Measure 86 starts with  $\frac{5}{8}$  time, dynamic *mp*, and a rhythmic pattern. Measure 87 begins with  $\frac{3}{5}$  time, dynamic *M.S.P.*, and continues the pattern. Measure 88 starts with  $\frac{3}{5}$  time, dynamic *mf*, and continues the pattern.

90

5

8

Violin 1 (Va. 1) and Violin 2 (Va. 2) parts. Measure 90 starts with  $\frac{5}{8}$  time, dynamic *p*, and a rhythmic pattern. Measure 91 begins with  $\frac{3}{5}$  time, dynamic *pp*, and continues the pattern. Measure 92 starts with  $\frac{3}{5}$  time, dynamic *f*, and continues the pattern.

93 **5****6**

*S.T.* (V) *M.S.T.*

Va. 1

*p* — *ppp* 3.2 *p* *ppp* *p*

*d<sup>(4)</sup>* *b<sup>(4)</sup>*

*ORD.* *S.T.* *S.P.*

Va. 2

*p* — *ppp* *p*

*d<sup>(4)</sup>* *b<sup>(4)</sup>*

96 **2****3****6**

*M.S.P.*

Va. 1

*mf*

*ORD.* *S.P.*

Va. 2

*mf*

*ORD.* *S.P.*

*S.T.*

 $\text{♩} = 72$ 100 **5** [G] Selidor (iii)**3**

*ORD.*

Va. 1

*p* *ppp* *p* *ppp* *p*

*d<sup>(4)</sup>* *b<sup>(4)</sup>*

*ORD.* *S.P.* (V) *ORD.* (V) *M.S.P.*

Va. 2

*p* *ppp* *p* *ppp* *p*

*d<sup>(4)</sup>* *b<sup>(4)</sup>*

*S.P.* V *ORD.* (V) *M.S.P.* *ORD.* (V) *S.P.*

104  $\frac{3}{4}$   $\frac{6}{8}$   $\frac{5}{8}$

M.S.P. (M.S.P.) ORD. (V) S.P. (V) ORD.

Va. 1

ppp ppp ppp ppp ppp

3.2 4.5 6.5

II 7 V (m) V (m) V (m) V (m)

4.5 4.5 4.5 4.5

Va. 2

ORD. S.P. ORD. (ORD.) M.S.P.

ppp p ppp p ppp

4.5 4.5 4.5 4.5

107  $\frac{3}{4}$

ORD. (ORD.) M.S.P. (m) ORD. M.S.P. (m) ORD.

Va. 1

ppp p ppp p ppp

3.2 4.5 6.5

II 7 V (m) V (m) V (m) V (m)

6.5 6.5 6.5 6.5

Va. 2

M.S.P. (V) ORD. (m) V (m) (ORD.) (m) V (m) S.P.

ppp p ppp p ppp p

3.2 4.5 6.5 6.5 6.5

110  $\frac{3}{4}$   $\frac{2}{4}$   $\frac{5}{8}$

(ORD.) (ORD.) M.S.P. (V) ORD. (V)

Va. 1

ppp p ppp p ppp

3.2 4.5 4.5

II 7 V (m) V (m) V (m) V (m)

4.5 4.5 4.5 4.5

Va. 2

S.P. M.S.P. (m) V (m) (m) V (m) S.P.

p ppp p ppp p

3.2 3.2 6.5 6.5 4.5

II 7 V (m) V (m) V (m) V (m)

6.5 6.5 6.5 6.5

II 7 V (m) V (m) V (m) V (m)

6.5 6.5 6.5 6.5

114 **3**

S.P. ..... ORD. (V)  $\downarrow = 36$

**8** [H] The Long Dune (ii)

Va. 1

Va. 2

ORD. ..... S.T.

M.S.T.

118 **6**

**5**

..... S.T.

Va. 1

Va. 2

ORD. ..... S.T. M.S.T.

121 **3**

**8**

**3**

ORD. ..... S.T.

Va. 1

Va. 2

M.S.T. S.T. ORD. ....

125

**2**

**6**

**2**

M.S.T.

V

f ————— p

ff ————— 6.5 ————— p

S.T.

M.S.T.

Va. 1

Va. 2

mf ————— p ————— 3.2 ————— f ————— ff

128

**6**

**3**

**2**

S.T.

M.S.T.

V

V

V

V

V

D.P.

Pizz.

(Pizz.) (Pizz.)

Va. 1

Va. 2

p ————— mf

ppp ————— p ————— ppp ————— p ————— ppp ————— f

p ————— 3 ————— 1/2 ————— 5/8 ————— x

ppp ————— p ————— 1/2 ————— 5/8 ————— 3/4 ————— x

x ————— x ————— x ————— x ————— x

x ————— x ————— x ————— x ————— x

Portland, OR  
Fresh Meadows, NY  
September 2014 - January 2015

This page intentionally left blank.

# 11

## *Invisible Cities (iii): Ersilia (2015)*

A composition for chamber orchestra

Premièred by Ensemble Dal Niente  
on Saturday May 16th, 2015  
in Paine Hall, Harvard University

# PREFACE

## 1

*In Ersilia, to establish the relationships that sustain the city's life, the inhabitants stretch strings from the corners of the houses, white or black or gray or black-and-white according to whether they mark a relationship of blood, of trade, authority, agency. When the strings become so numerous that you can no longer pass among them, the inhabitants leave: the houses are dismantled; only the strings and their supports remain.*

*From a mountainside, camping with their household goods, Ersilia's refugees look at the labyrinth of taut strings and poles that rise in the plain. That is the city of Ersilia still, and they are nothing.*

*They rebuild Ersilia elsewhere. They weave a similar pattern of strings which they would like to be more complex and at the same time more regular than the other. Then they abandon it and take themselves and their houses still farther away.*

*Thus, when traveling in the territory of Ersilia, you come upon the ruins of abandoned cities, without the walls which do not last, without the bones of the dead which the wind rolls away: spiderwebs of intricate relationships seeking a form.*

- Italo Calvino, *Invisible Cities*

## 2

*"Are we still in the South Reach?"*

*"Reach? No. The islands – "The chief moved his slender black hand in an arc, no more than a quarter of the compass, north to east. "The islands are there," he said. "All the islands." Then showing all the evening sea before them, from north through west to south, he said, "The sea."*

*"What land are you from, lord?"*

*"No land. We are the Children of the Open Sea."*

*Arren looked at his keen face. He looked about him at the great raft with its temple and its tall idols, each carved from a single tree, great god-figures mixed of dolphin, fish, man, and seabird; at the people busy at their work, weaving, carving, fishing, cooking on raised platforms, tending babies; at the other rafts, seventy at least, scattered out over the water in a great circle perhaps a mile across. It was a town: smoke rising in thin wisps from distant houses, the voices of children high on the wind. It was a town, and under its floors was the abyss.*

- Ursula LeGuin, *The Farthest Shore*

## 3 Instrumentation

- Flute
- Bass clarinet
- Oboe
- Baritone saxophone
- Acoustic guitar
- Piano
- Percussion
  - bamboo wind chimes
  - four toms
  - five wood blocks
  - snare drum
  - marimba
  - crotalines, two octaves
  - tam-tam
  - bass drum
- Violin
- Viola
- Cello
- Contrabass

## 4 Performance notes

Six players flute, bass clarinet, oboe, violin, viola and cello receive a shaker caxixi, maraca or similar. Four players guitar, piano, percussion and contrabass receive a chromatic pitch pipe a circular harmonic-like instrument generally used for tuning vocal groups.

The shakers should be placed on or suspended from their respective performer's music stands, or wherever convenient.

The pitch-pipes should be played by inhaling or exhaling as indicated through fully half of the circumference of the instrument, creating a rich cluster.

For all winds, a +symbol indicates slap tonguing.

For guitar, the coda symbol indicates a percussive damping of the strings.

Piano plays with pedal to their discretion throughout the first four sections of the piece. The sound should be generally dry, although some pedal should be used when appropriate for phrasing and blending, especially on tremolo passages. The sustain pedal should remain fully depressed for the entirety of section D. Inside-piano glissandi are notated proportional to the lower interior portion of the instrument from the lowest string up to the first cross-bar and should be played with the fingertip.

Percussion should use hard sticks on toms, woodblocks, snares and crotalines, and softer mallets on marimba, bass drum and tam-tam (when ergonomic to do so).

All mordents and trills are a major 2nd, unless otherwise specified. All tremolos are unmeasured and should evoke an even, cloud-like texture.

*Invisible Cities (iii):*  
**ERSILIA**  
*(a botanical survey of the uninhabited northeastern isles)*  
 for Ensemble *Dal Niente*

Josiah Wolf Oberholtzer (1984)

$\text{♩} = 96$

**Komokome**

Flute  
Oboe  
Bass Clarinet  
Baritone Saxophone  
Pitch Pipes  
Guitar  
Pitch Pipes  
Piano  
Pitch Pipes  
Percussion  
Violin  
Viola  
Cello  
Pitch Pipes  
Contrabass

shaker  
snare

i

8

4

8

Fl. *fp*

Ob. *shaker*

Bass cl. *shaker*  
*ppp* 7.6

Bar. sax. *mf* *f* *#* *mf* *5:4* *ppp* *fp*

Gt. *fb*

Pf. *f* *13:21* *pp* *3:2* *mp* *mf* *mp* *fb* *ppp*

Perc. *fff* *fff* *fff* *p* *f* *blocks*

Vn. *ppp* *fp*

Va. *ppp*

Vc. *ppp* *5:4*

Cb. *s* *ppp* *p*

8

8

4

Fl. *p* *mf*

Ob.

Bass cl. *ppp*

Bar. sax. *f* *fp* *fp*

Gt. *f* *fp* *fp* *fp*

Pf. *sva1* *fff* *fff* *ppp*

Perc. *toms* *f* *snare* *fff* *fp*

Vn. *ppp* *pizz.*

Va. *fp* *ppp*

Vc. *fp*

Cb.

Musical score page 7, featuring parts for Flute (Fl.), Oboe (Ob.), Bassoon (Bass cl.), Bass Clarinet (Bass cl.), Bass Saxophone (Bar. sax.), Guitar (Gt.), Piano (Pf.), Percussion (Perc.), Violin (Vn.), Cello (Cello), and Double Bass (Cb.). The score includes dynamic markings such as *p*, *f*, *mf*, *ppp*, *fff*, and *ffff*. Various performance instructions are present, including "shaker" for shaker instruments and "crotales" for crotal bells. Measure numbers 1, 2, 3, 4, and 5 are indicated above the staves.

Fl. *shaker*  
*fp*

Ob. *ppp*

Bass cl. *ppp*

Bar. sax. *f* *fp* *mf* *fp*

Gt. *f* *fp* *ppp*

Pf. *f* *5.4* *ppp* *fp* *p* *mf*

Perc. *fff* *fff* *fff*

Vn.

Va.

Vc.

Cb. *pizz.* *ppp* *5.4*

17

8

5

6

Fl.  $f \xrightarrow{\text{--}} p$

Ob.  $mp \xrightarrow{\text{--}} p$

Bass cl.  $mf \xrightarrow{\text{--}} p$

Bar. sax.

Gt.  $\text{Sva} \xrightarrow{\text{--}} p$

Pf.  $f \xrightarrow{\text{--}} p$

Perc.  $\text{snare} \xrightarrow{\text{--}} fff$

Vn.  $\text{shaker} \xrightarrow{\text{--}} pp$

Va.  $\text{shaker} \xrightarrow{\text{--}} pp$

Vc.  $\text{shaker} \xrightarrow{\text{--}} pp$

Cb.

21      7      4      5      6

Fl. *f* *fp* *ppp* *f*  
 Ob. *f*  
 Bass cl.  
 Bar. sax.  
 Gt. *f* *fp* *mf*  
 Pf. *ppp* *ff* *f* *mf*  
 Perc. *f* *fp* *f* *blocks* *7.6* *toms*  
 Vn.  
 Va.  
 Vc.  
 Cb. *pizz.* *7.6* *ppp* *f*

Fl. *f* *p* *fp* *f* *p* *fp* *f* *p* *fp* *p* *ppp* *3:2* *fp* *mfp* *ppp* *3:2* *fp* *mfp* *ppp* *5:4*

Ob. *Flz.* *p* *7:6* *f* *p* *mp* *ppp* *3:2* *7:6* *ppp* *5:4*

Bass cl. *f* *mf* *f* *p* *ppp* *3:2* *ppp*

Bar. sax. *f* *mf* *fff* *mf* *f* *mf* *f*

Gt. *f* *mf* *f* *p*

Pf. *f* *5:4* *mf* *f* *mf* *f* *5:4* *mf* *p* *f* *Sva-*

Perc. *ppp* *3:2* *f* *p* *4:3* *f* *fp* *f* *bass drum*

Vn. *p* *f* *p* *5:4* *p*

Va. *f* *p* *fp* *shaker* *f* *p* *f* *p* *ppp* *3:2*

Vc. *p* *4:3* *f* *f* *p* *5:4* *p*

Cb. *p* *4:3* *ppp* *p* *4:3* *p* *3:2* *f*

Fl. *shaker*  
*ppp* 7:6

Ob. *shaker*  
*ppp*

Bass cl. *shaker*  
*ppp* 5:4

Bar. sax. *p* *fp* *fp* *f*

Gt. *p* *fp*

Pf. *sva* *p* 3:2 *pp* *fp* *p* *mp* *mf* *p* *mf*

Perc. *p* *ppp*

Vn. *pizz.* *ppp* 7:6 *pizz.* *ppp* 7:6 *pizz.* *ppp*

Va. *ppp* 7:6 *ppp* 7:6

Vc. *pizz.* *ppp* 5:4 *ppp* *ppp* 3:2 *pizz.* *ppp*

Cb. *pizz.* *ppp*

Fl. *fp* *ppp*

Ob.

Bass cl. *fp*

Bar. sax. *f* *mf* *f* *mf* *f* *mf*

Gt. *f* *f* *f* *f* *f* *f* *f*

Pf. *f* *f* *f* *f* *f* *f* *fff*

Perc. *crotales* *mp* *fff*

Vn. *shaker* *ppp* *fp*

Va. *shaker* *ppp*

Vc. *shaker* *ppp*

Cb. *s* *p* *p* *ppp*

Fl. *shaker* *ppp*

Ob. *shaker* *5A* *ppp*

Bass cl. *shaker* *fpp*

Bar. sax. *f* *mf* *fp* *fp* *mf*

Gt. *pp* *f* *fp*

Pf. *f* *pp* *3.2* *mp* *fp* *ppp* *mf* *p* *mf*

Perc. *fff*

Vn.

Va.

Vc.

Cb. *pizz.* *ppp*

Fl. *fp* *mf* *ppp* *fp* *ppp* 5:4 *fp*

Ob. *ppp* 7:6 *ppp* 5:4

Bass cl. *ppp* 5:4

Bar. sax.

Gt. *f* *fz* *f* *fz* *f* *fz* *f* *fz*

*Sva1* *fp* *pp* 4:3 *f* *mp* *mf* *p* *mf* 3:2

Pf. *fff* *fff* *fff* *fff*

Perc. *snare* *fff* *fff* *fff*

Vn. *overpressure* *fff* *fff*

Va. *overpressure* *fff* *ppp* *pizz.* *fff* *overpressure*

Vc. *overpressure* *fff* *ppp* *pizz.* *fff* *overpressure*

Cb. *5:4* *ppp* *5:4* *ppp*

Fl. *ppp*

Ob. *fp* *5.4 ppp*

Bass cl. *ppp* *5.4*

Bar. sax. *fp* *f* *mf* *fp*

shaker

Gt. *p* *f* *fp*

Pf. *f* *pp* *mp* *fp* *mf* *mp* *f* *5.4* *f*

Perc. *fff* *p* *5.4* *f* *fff* *toms* *snare*

Vn. *fff* *overpressure* *shaker* *ppp*

Va. *fff* *overpressure* *shaker* *ppp*

Vc. *ppp* *shaker* *overpressure* *fff*

Cb.

47

८५

三

Fl. *shaker*  
Ob. *shaker*  
Bass cl.  
Bar. sax.  
Gt.  
Pf.  
Perc. *toms* *blocks*  
Vn.  
Va.  
Vc. *shaker*  
Cb.

51

58

Fl. *p*, *fp*, *ppp*, *mf*, *fp*

Ob. *ppp*

Bass cl.

Bar. sax. *mf*, *f*, *mf*, *f*, *mf*, *f*, *mf*, *f*

Gt. *f*, *fp*

Pf. *f*, *f*, *f*, *fff*, *fff*, *fff*, *fff*

Perc. *f*, *snare*, *fff*

Vn. *overpressure*, *fff*

Va. *overpressure*, *fff*

Vc. *overpressure*, *pizz.*, *ppp*, *5:4*

Cb.

55      48      7      58

**Fl.**  
 Measures 55-58: Flute parts show various rhythmic patterns and dynamics (mf, 5:4, etc.).  
  
**Ob.**  
 Measures 55-58: Oboe parts show various rhythmic patterns and dynamics (ppp, 5:4, etc.).  
  
**Bass cl.**  
 Measures 55-58: Bassoon parts show various rhythmic patterns and dynamics (ppp, p, etc.).  
  
**Bar. sax.**  
 Measures 55-58: Baritone Saxophone parts show various rhythmic patterns and dynamics (4:3, f, etc.).  
  
**Gtr.**  
 Measures 55-58: Guitar parts show sustained notes and dynamics (mf, pp, f).  
  
**Pf.**  
 Measures 55-58: Piano parts show various dynamics (f, pp, mp, etc.) and specific fingerings.  
  
**Perc.**  
 Measures 55-58: Percussion parts include shaker, crotalines, and tam-tam.  
  
**Vn.**  
 Measures 55-58: Violin parts include shaker and dynamic markings (5:4, ppp, ff).  
  
**Va.**  
 Measures 55-58: Viola parts include shaker and dynamic markings (ppp, ff).  
  
**Vc.**  
 Measures 55-58: Cello parts include shaker and dynamic markings (5:4, ppp).  
  
**Cb.**  
 Measures 55-58: Double Bass part is mostly silent.

Fl. *p* *ppp*

Ob. *shaker* *ppp* *jdp*

Bass cl. *shaker* *ppp* 7.6

Bar. sax. *fp* *fp*

Gt. *f*

Pf. *solo* *fp* *mp* 3.2 *p* *mf* *f* *ff*

Perc. *mf* *fff* *snare*

Vn. *pizz.* 5.4 *ppp*

Va. *pizz.* 5.4 *ppp*

Vc. *pizz.* *ppp*

Cb. *pizz.* 5.4 *ppp*

Fl.  $\text{fp}$

Ob.  $\text{ppp}$

Bass cl.  $\text{fp}$

Bar. sax.  $\text{ppp}$

Gt.  $f$

Pf.  $\text{pp} \rightarrow f$

*Sh. 1*

Perc.  $\text{fff}$

Vn.  $\text{fff}$

overpressure

shaker

Va.  $\text{fff}$

overpressure

shaker

Vc.  $\text{fff}$

overpressure

shaker

Cb.  $p$

$\text{ppp}$

Fl.

Ob.

Bass cl.

Bar. sax.

Gt.

Pf.

Perc.

Vn.

Va.

Vc.

Cb.

*Flz.*

*Sva1*

*crotales*

*snare*

*overpressure*

♩ = 96  
 (72) 8 [A] [i] 8

Fl. *f* — 7:6 *mp*  
 Ob. *f* *p*  
 Bass cl. *f* — *mf*  
 Bar. sax. *f* — *mf* *f* — *mf* (15:14)  
 Gt. *f* — 9:8 *mf*  
 Pf. *f*  
 Perc. *p* *ppp* (toms) *8:7* *f*  
 Vn. *f* — 7:6 *p*  
 Va. *p* *f*  
 Vc. *f* — *p*  
 Cb. *p* — 7:6 *f*

**78** B Sort

४३५

79

8

8

Fl.

Ob.

Bass cl.

Bar. sax.

Gt.

Pf.

Perc.

Vn.

Va.

Vc.

Cb.

Dynamic markings and performance instructions:

- Flute: dynamic *p*, grace note
- Oboe: dynamic *ppp*, grace note
- Bass clarinet: dynamic *ppp*, grace note
- Baritone saxophone: dynamic *ppp*, grace note, dynamic *ff*, dynamic *f*, dynamic *p*
- Guitar: dynamic *pp*, grace note, instruction *5:4*
- Piano: dynamic *f*, dynamic *ff*, dynamic *f*, dynamic *mp*, dynamic *p*
- Percussion: dynamic *f*, dynamic *mf*, dynamic *f*
- Violin: dynamic *p*, dynamic *mf*
- Viola: dynamic *p*, dynamic *mf*, dynamic *ff*
- Cello: dynamic *p*, dynamic *mf*, dynamic *ff*
- Double Bass: dynamic *mf*, dynamic *ff*

Fl.

Ob.

Bass cl.

Bar. sax.

Pp.

Gt.

Pp.

Pf.

Vn.

Va.

Vc.

Pp.

Cb.

*mp*

*f* *mf*

*7:6*

*5:4*

*inhaler*

*windchimes*

*crotales*

*ppp*

86

8

8

Fl.

Ob.

Bass cl.

Bar. sax.

Gt.

Pp.

Pf.

Perc.

Vn.

Va.

Vc.

Cb.

Fl.  $\text{ppp}$

Ob.  $\text{ppp}$

Bass cl.  $\text{mp}$  5.4  $\text{mf}$

Bar. sax.

Gt.  $\text{p}$   $\text{ff}$   $\text{pp}$   $\text{p}$   $\text{ff}$

Pf.  $\text{pp}$

Perc.  $\text{windchimes}$   $\text{pp}$

Vn.  $\text{fp}$

Va.  $\text{p}$   $\text{fp}$   $\text{fp}$

Vc.  $\text{mf}$   $\text{fp}$

Cb.  $\text{fp}$

Fl.  $\frac{6:4}{\text{6:4}}$   $\text{p}$

Ob.

Bass cl.  $f$

Bar. sax.  $\frac{5:4}{f \text{ m}f}$   $\frac{7:6}{f \text{ m}f}$

Pp.  $\text{fp}$  inhalé

Gt.  $\text{mp}$   $\text{m}f$  L.V.  $\text{pp}$  L.V.

Pp.  $\text{exhale}$   $\text{fp}$

Pf.  $\text{mf}$   $f$   $f$   $f \text{ m}f$   $\text{fp}$   $\text{pp}$  Svā  $\text{pp}$  Svā-----

Pp.  $\text{fp}$  inhalé

Perc.  $\text{marimba}$   $\text{crotales}$   $\text{blocks}$   $\frac{5:4}{f \text{ m}f}$   $\text{marimba}$

Vn.  $\text{p}$   $\text{fp}$   $\text{p}$

Va.  $\text{ff}$   $\text{fp}$   $\text{p}$   $\text{fp}$

Vc.  $\text{fp}$   $\text{p}$

Pp.  $\text{mp}$  inhalé  $\text{fp}$

Cb.  $\text{mf}$   $\text{p}$   $\text{fp}$   $\frac{3:2}{\text{p}}$

Fl. *ppp*

Ob. *ppp*

Bass cl. *ppp*

Bar. sax. *ppp* *fp*

Pp. *exhale*

Gt. *pp* *p* *LV*

Pp.

Pf. *fp* *fp* *sva*

Pp.

Perc. *f* *mf* *f*

Vn. *fp* *p*

Va. *p*

Vc. *ff* *p* *fp*

Pp. *ppp*

Cb. *fp*

Fl.  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *mp*

Ob.  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *mp*

Bass cl.  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *p*

Bar. sax.  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *fff*

Gt. *L.V.*  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *p*

Pf.  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *f*

Perc. *blocks*  $\text{F} \# \text{A} \text{C} \text{D} \text{E} \text{G}$  *f*

Vn. *fp*

Va. *mf*

Vc. *p*

Cb. *fp*

*windchimes* *pp*

*overpressure* *fff*

*overpressure* *fff*

*overpressure* *fp*

Fl.  $\frac{3}{2}$   $\text{mp} \text{ ppp}$

Ob.  $\text{5.4-} \text{mp}$   $\text{ppp} \text{ mf}$

Bass cl.  $\text{8.6-} \text{mf}$   $\text{ppp}$   $\text{f} \text{ 5.4-} \text{mf}$

Bar. sax.  $\text{ppp}$   $\text{ppp-5.4-}$   $\text{ppp}$

Gt.  $\text{6.5-}$   $\text{6.5-}$   $\text{p}$

Pf.  $\text{fp} \text{ f}$   $\text{pp} \text{ fp}$   $\text{p}$   $\text{mp} \text{ 3.2-}$   $\text{fp} \text{ mf}$

Perc.  $\text{blocks}$   $\text{windchimes}$   $\text{marimba}$   $\text{7.6-} \text{mf}$   $\text{f} \text{ 7.6-} \text{mf}$   $\text{f} \text{ 3.2-} \text{mf}$   $\text{f} \text{ 5.4-}$

Vn.  $\text{fff}$   $\text{fff}$   $\text{fff}$   $\text{mf}$   $\text{fp}$

Va.  $\text{fff}$   $\text{6.5-}$   $\text{fp}$

Vc.  $\text{overpressure}$   $\text{fff}$   $\text{fff}$   $\text{fff}$   $\text{fp}$

Cb.  $\text{mf}$

Fl. *ppp*

Ob. *ppp*

Bass cl. *ppp*

Bar. sax. *ppp*

*inhaler*

Pp. *fp*

Gt. *ppp*

Pp. *inhaler*

Pf. *fp*

*exhaler*

Pp. *fp*

Perc. *pp*

Vn. *fff*

Va. *fff*

Vc. *fff*

Pp. *exhale*

Cb. *fp*

Fl.

Ob.

Bass cl.

Bar. sax.

Gt.

Pf.

Perc.

Vn.

Va.

Vc.

Cb.

Musical score page showing measures 1-4 for various instruments:

- Fl.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 7.6 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Ob.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 8.6 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Bass cl.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 5.4 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Bar. sax.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 5.4 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Gt.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 7.6 measures of eighth-note patterns. Measure 4: 5.4 measures of eighth-note patterns.
- Pf.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 8va1. Measure 4: 8va1.
- Perc.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 3.2 measures of eighth-note patterns. Measure 4: 5.4 measures of eighth-note patterns.
- Vn.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 6.5 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Va.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 6.5 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Vc.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 6.5 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.
- Cb.**: Measures 1-2, 4 measures of eighth-note patterns. Measure 3: 6.5 measures of eighth-note patterns. Measure 4: 3.2 measures of eighth-note patterns.

Fl. *ppp* *mp*

Ob. *ppp* *p* *f* *mf*

Bass cl. *ppp* *p* *f* *mf*

Bar. sax. *f* *mf* *fp* *f* *mf* *ppp*

*inhal* *exhal*

Pp. *pp* *fp* *p* *fp*

Gt. *L.V.* *ppp* *p* *fp* *p* *fp*

Pp. *inhal* *exhal*

Pf. *f* *mf* *f* *fp* *p*

Pp. *f* *mf* *f* *inhal* *fp*

Perc. *blocks* *marimba* *f* *p* *fp* *marimba*

Vn. *fp*

Va. *mf* *mp*

Vc. *p*

Pp. *inhal* *fp*

Cb.

Fl.

Ob.

Bass cl.

Bar. sax.

Gt.

Pf.

Perc.

Vn.

Va.

Vc.

Cb.

5.4 *mf*

*L.V.*

*f* *p*

*7.6* *mf*

*3.2*

*5.3*

*p* *ppp*

Fl. *ppp* *mf*

Ob. *p* *p*

Bass cl. *mp* *f* *mp*

Bar. sax. *fp* *mf*

Gt. *ppp* *p* *p* *p* *fp* *sforz.*

Pf. *p* *p* *p* *fp* *sforz.*

Perc. *mf* *ppp* *pp*

*windchimes*

Vn. *overpressure* *fff* *fff* *fp*

Va. *fff* *fff* *p* *fff*

Vc. *fff* *overpressure* *fp*

Cb. *p*

Fl.

Ob.

Bass cl.

Bar. sax.

Gt. *[LV]*

Pf. *Sva 1*

Perc. *windchimes* *crotales*

Vn. *overpressure*

Va. *overpressure*

Vc. *overpressure*

Cb. *p*

This musical score page contains ten staves of music for various instruments. The instruments listed from top to bottom are: Flute, Oboe, Bass Clarinet, Baritone Saxophone, Guitar, Piano, Percussion, Violin, Viola, and Double Bass. The score includes dynamic markings such as *mp*, *f*, *pp*, *mf*, and *fff*. Performance instructions include *overpressure* for the strings and *crotales* for the Percussion. Measure numbers 5.4 and 7.6 are marked above specific measures. The page number 134 and section letter G are at the top left.

138 **C** [ii]

**J = 64**

Fl. *ppp*  
Ob. *7.6* *ppp*  
Bass cl. *f* *5.4 mp*  
Bar. sax. *f* *7.6* *mf*

**D** Chemish

**J = 80**

Gt. *f* *7.6 mf*  
Pf. *p* *f* *8va* *5.4 mf*  
Perc. *toms* *p* *ppp* *5.4 f*  
**bass drum**

Vn. *f* *p*  
Va. *p*  
Vc. *f* *3.2 p*  
Cb. *pizz.* *ppp*

143

7

85

8

Fl. *fp* *mf* *shaker* *fp*

Ob. *ff* *p*

Bass cl. *ff* *p*

Bar. sax. *fp* *mf* *fp* *mp* *p* *p*

Gt.

Pf. *p* *inside*

Perc. *p* *tam*

Vn. *mp*

Va. *mp* *shaker* *fp*

Vc. *p* *mp*

Cb.

147

4

8

8

Fl.

Ob.

Bass cl.

Bar. sax.

Pp.

Gt.

Pp.

Pf.

Pp.

Vn.

Va.

Vc.

Pp.

Cb.

Fl.  $\text{f} \text{p}$

Ob.  $\text{f} \text{p}$

Bass cl.  $\text{f} \text{p}$

Bar. sax.  $\text{f} \text{p} \text{mf}$

Pp.  $\text{p}$

Gt.  $\text{L.V.} \text{mf}$

Pp.  $\text{p}$

Pf.  $\text{ppp}$

Pp.  $\text{mf}$

Perc.  $\text{bass drum}$

Vn.  $\text{f} \text{p}$

Va.  $\text{p}$

Vc.  $\text{shaker} \text{f} \text{p}$

Pp.  $\text{ppp}$

Cb.  $\text{f} \text{p}$

Fl. *fp* *mp*

Ob. *fp* *p*

Bass cl. *p* *fp*

Bar. sax. *f* *fp* *p* *mf*

Pp. *pp*

Gt. *LV* *ppp*

Pf. *ppp*

Pp.

Perc. *ppp*

Vn. *p* *fp*

Va. *fp*

Vc. *p* *fp*

Cb.



161

Fl. *fp* *mf* *fp* *fp* *f* *shaker*

Ob. *fp* *p* *shaker*

Bass cl. *pp* *5A* *fp* *mf* *5A* *fp* *pp* *shaker*

Bar. sax.

Gt. *s* *LV* *LV* *ppp* *p* *ppp* *L.V.* *p* *fz*

Pf.

Perc. *bass drum* *ppp* *tam* *ppp* *ppp*

Vn. *fp* *pp* *shaker*

Va. *ppp* *shaker*

Vc. *fp* *pp* *shaker*

Cb.

165

4

7

5

Fl. *f*  
*fp* *pp*

Ob. *pp* *fp* *mf*

Bass cl. *pp* *7.6 f*

Bar. sax.

Pp. *inhal* *ppp*

Gt. *L.V.* *p*

Pp. *exhal* *fp* *p*

Pf. *ppp*

Pp. *exhal* *fp* *p*

Perc. *p*

Vn. *pp*

Va. *pp*

Vc. *ppp*

Pp. *inhal* *mf*

Cb. *ppp*

169

4

4

5

4

8

Fl. *trill* *shaker*  
*ppp* *fp*

Ob. *shaker*  
*fp*

Bass cl. *trill* *shaker*  
*fp*

Bar. sax. *trill* *9.8* *f* *pp* *fp*

Gt. *p* *s* *LV* *p* *pp* *LV* *p*

Pf.

Perc. *bass drum* *mf*

Vn. *shaker*  
*fp*

Va. *shaker*  
*fp*

Vc. *shaker*  
*fp*

Cb.

Fl. *fp*

Ob.

Bass cl. *fp* *fp* *ppp* *f*

Bar. sax. *p* *mp* *fp* *fp* *7.6*

Pp. *exhale* *fp* *p*

Gt. *p* *LV* *mf* *fz* *inhal* *LV* *ppp*

Pp. *inside* *p* *inside* *p* *inside* *p* *mf*

Pf. *tam* *ppp* *snare* *fff*

Vn. *fp*

Va.

Vc.

Pp. *inhal* *fp* *ppp*

Cb. *p*

Fl.

Ob.

Bass cl.

Bar. sax.

Pp.

Gt.

Pp.

Pf.

Pp.

Perc.

Vn.

Va.

Vc.

Pp.

Cb.

*shaker*

*fp*

*pp*

*fz*

*inhale*

*fp*

*ppp*

*inhale*

*fp*

*shaker*

*fp*

*shaker*

*fp*

*shaker*

*fp*

*exhale*

181

8

5

Fl. *shaker*  
 Ob.  
 Bass cl.  
 Bar. sax.

Pp. *inhale*  
 Gt. *ppp* *fz*  
 Pp. *exhale*  
 Pf.  
 Pp. *exhale*  
 Perc. *bass drum*  
*snare*  
*fff*

Vn. *shaker*  
 Va.  
 Vc.  
 Pp. *inhal*  
 Cb.

Portland, OR  
 January 2015 - April 2015

## **Part III**

## **Appendices**

This page intentionally left blank.

# A

*consort* source

## A.1 CONSORTTOOLS.ABSOLUTEPICHANDLER

```
1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from consort.tools.PitchHandler import PitchHandler
4
5
6 class AbsolutePitchHandler(PitchHandler):
7     r'''Absolute pitch maker.
8
9     :::
10
11     >>> import consort
12     >>> pitch_handler = consort.AbsolutePitchHandler(
13         ...     pitchSpecifier="c' d' e' f",
14         ... )
15     >>> print(format(pitch_handler))
16     consort.tools.AbsolutePitchHandler(
17         pitchSpecifier=consort.tools.PitchSpecifier(
18             pitchSegments=(
19                 pitchtools.PitchSegment(
20                     (
21                         pitchtools.NamedPitch("c"),
22                         pitchtools.NamedPitch("d"),
23                         pitchtools.NamedPitch("e"),
24                         pitchtools.NamedPitch("f"),
25                     ),
26                     itemClass=pitchtools.NamedPitch,
27                     ),
28                     ),
29                     ratio=mathtools.Ratio((1,)),
30                     ),
```

```

31         )
32     ...
33
34     ### CLASS VARIABLES ###
35
36     __slots__ = ()
37
38     ### SPECIAL METHODS ###
39
40
41     def __call__(
42         self,
43         attack_point_signature,
44         logical_tie,
45         musicSpecifier,
46         pitch_choices,
47         previous_pitch,
48         seed_session,
49         ):
50         pitch = self._get_pitch(
51             attack_point_signature,
52             pitch_choices,
53             previous_pitch,
54             seed_session.current_phrased_voicewise_logical_tie_seed,
55             )
56         pitch = self._apply_deviation(
57             pitch,
58             seed_session.current_unphrased_voicewise_logical_tie_seed,
59             )
60         return pitch
61
62     ### PRIVATE METHODS ###
63
64     def __get_pitch(
65         self,
66         attack_point_signature,
67         pitch_choices,
68         previous_pitch,
69         seed,
70         ):
71         pitch = pitch_choices[seed]
72         if pitch_choices and \
73             1 < len(set(pitch_choices)) and \
74             self.forbid_repetitions:
75             if self.pitch_application_rate == 'phrase':
76                 if attack_point_signature.is_first_of_phrase:
77                     while pitch == previous_pitch:
78                         seed += 1
79                         pitch = pitch_choices[seed]
80             elif self.pitch_application_rate == 'division':
81                 if attack_point_signature.is_first_of_division:
82                     while pitch == previous_pitch:
83                         seed += 1
84                         pitch = pitch_choices[seed]

```

```

85         else:
86             while pitch == previous_pitch:
87                 seed += 1
88                 pitch = pitch_choices[seed]
89     return pitch

```

## A.2 CONSORT.TOOLS.AFTERGRACESELECTORCALLBACK

```

1 # -*- encoding: utf-8 -*-
2 from abjad import inspect_
3 from abjad.tools import abctools
4 from abjad.tools import selectiontools
5
6
7 class AfterGraceSelectorCallback(abctools.AbjadValueObject):
8
9     """CLASS VARIABLES"""
10
11     __slots__ = ()
12
13     """SPECIAL METHODS"""
14
15     def __call__(self, expr):
16         assert isinstance(expr, tuple), repr(tuple)
17         result = []
18         for subexpr in expr:
19             subresult = []
20             for x in subexpr:
21                 subresult.append(x)
22                 if inspect_(x).get_grace_containers('after'):
23                     subresult = selectiontools.Selection(subresult)
24                     result.append(subresult)
25                     subresult = []
26             if subresult:
27                 subresult = selectiontools.Selection(subresult)
28                 result.append(subresult)
29         return tuple(result)

```

## A.3 CONSORT.TOOLS.ANNOTATE

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad import attach
4 from abjad import inspect_
5 from abjad import iterate
6 from abjad import override
7 from abjad.tools import durationtools
8 from abjad.tools import scoretools
9 from abjad.tools import schemetools
10
11
12 def make_annotated_phrase(phrase, color=None):
13     duration = inspect_(phrase).get_duration()

```

```

14     annotated_phrase = scoretools.FixedDurationTuplet(duration)
15     durations = [inspect_(_.get_duration() for _ in phrase]
16     leaves = scoretools.make_leaves([0], durations)
17     annotated_phrase.extend(leaves)
18     if color:
19         override(annotated_phrase).tuplet_bracket.color = color
20     return annotated_phrase
21
22
23 def make_annotated_division(division, color=None):
24     duration = inspect_(division).get_duration()
25     if duration != 1:
26         note = scoretools.Note("c''1")
27         annotated_division = scoretools.Tuplet(duration, (note,))
28     else:
29         note = scoretools.Note("c''2")
30         annotated_division = scoretools.Tuplet(2, (note,))
31     leaves = division.select_leaves()
32     prototype = (scoretools.Rest, scoretools.MultimeasureRest)
33     if all(isinstance(_, prototype) for _ in leaves):
34         manager = override(annotated_division)
35         manager.tuplet_bracket.dash_fraction = 0.1
36         manager.tuplet_bracket.dash_period = 1.5
37         manager.tuplet_bracket.style = \
38             schemetools.SchemeSymbol('dashed-line')
39     if color:
40         override(annotated_division).tuplet_bracket.color = color
41     return annotated_division
42
43
44 def make_empty_phrase_divisions(phrase):
45     inner_container = scoretools.Container()
46     outer_container = scoretools.Container()
47     for division in phrase:
48         duration = inspect_(division).get_duration()
49         multiplier = durationtools.Multiplier(duration)
50         inner_skip = scoretools.Skip(1)
51         outer_skip = scoretools.Skip(1)
52         attach(multiplier, inner_skip)
53         attach(multiplier, outer_skip)
54         inner_container.append(inner_skip)
55         outer_container.append(outer_skip)
56     return inner_container, outer_container
57
58
59 def annotate(context, nonsilence=None):
60     prototype = consort.MusicSpecifier
61     silenceSpecifier = consort.MusicSpecifier()
62     annotated_context = context
63     context_mapping = {}
64     for voice in iterate(annotated_context).by_class(scoretools.Voice):
65         if voice.context_name == 'Dynamics':
66             continue
67         division_voice = scoretools.Context(

```

```

68         context_name='AnnotatedDivisionsVoice',
69     )
70     phrase_voice = scoretools.Context(
71         context_name='AnnotatedPhrasesVoice',
72     )
73     for phrase in voice:
74         color = None
75         if nonsilence:
76             music_specifier = inspect_(phrase).get_indicator(prototype)
77             if music_specifier == silenceSpecifier:
78                 inner_container, outer_container = \
79                     make_empty_phrase_divisions(phrase)
80                 division_voice.append(inner_container)
81                 phrase_voice.append(outer_container)
82                 continue
83             if music_specifier:
84                 color = music_specifier.color
85             for division in phrase:
86                 annotation_division = make_annotated_division(division, color)
87                 division_voice.append(annotation_division)
88                 annotation_phrase = make_annotated_phrase(phrase, color)
89                 phrase_voice.append(annotation_phrase)
90             parent = inspect_(voice).get_parentage().parent
91             if parent not in context_mapping:
92                 context_mapping[parent] = []
93             context_mapping[parent].append(division_voice)
94             context_mapping[parent].append(phrase_voice)
95     for context, annotation_voices in context_mapping.items():
96         context.is_simultaneous = True
97         context.extend(annotation_voices)

```

## A.4 CONSORT.TOOLS.ATTACHMENTEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import collections
4 import copy
5 import funcsigs
6 from abjad import attach
7 from abjad import new
8 from abjad.tools import datastructuretools
9 from abjad.tools import scoretools
10 from abjad.tools import selectortools
11 from abjad.tools import sequencetools
12 from abjad.tools import spannertools
13 from consort.tools.HashCachingObject import HashCachingObject
14
15
16 class AttachmentExpression(HashCachingObject):
17     r'''An attachment specifier.
18
19     .. container:: example
20
21     ::
```

```

22
23     >>> import consort
24     >>> attachment_expression = consort.AttachmentExpression(
25         ...     attachments=(indicatortools.Articulation('>'),),
26         ...     selector=selectortools.Selector().by_leaves().by_run(Note)[0],
27         ... )
28     >>> print(format(attachment_expression))
29     consort.tools.AttachmentExpression(
30         attachments=datastructuretools.TypedList(
31             [
32                 indicatortools.Articulation('>'),
33             ]
34         ),
35         selector=selectortools.Selector(
36             callbacks=(
37                 selectortools.PrototypeSelectorCallback(
38                     prototype=scoretools.Leaf,
39                     ),
40                 selectortools.RunSelectorCallback(
41                     prototype=scoretools.Note,
42                     ),
43                 selectortools.ItemSelectorCallback(
44                     item=0,
45                     apply_to_each=True,
46                     ),
47                 ),
48             ),
49         )
50
51     :::
52
53     >>> attachment_expression = consort.AttachmentExpression(
54         ...     attachments=(
55             ...         consort.SimpleDynamicExpression(
56                 ...             hairpin_start_token='sfz',
57                 ...             hairpin_stop_token='o',
58                 ...             ),
59                 ...             ),
60             ...     selector=selectortools.Selector().by_leaves().by_run(Note),
61             ... )
62
63     :::
64
65     >>> staff = Staff("c'8 r8 d'8 e'8 r8 f'8 g'8 a'8")
66     >>> attachment_expression(staff)
67     >>> print(format(staff))
68     \new Staff {
69         c'8 \sfz
70         r8
71         \override Hairpin #'circled-tip = ##t
72         d'8 \> \sfz
73         e'8 \!
74         \revert Hairpin #'circled-tip
75         r8

```

```

76         \override Hairpin #'circled-tip = ##t
77         f'8 \> \sfz
78         g'8
79         a'8 \!
80         \revert Hairpin #'circled-tip
81     }
82
83 .. container:: example
84
85 :::
86
87     >>> attachment_expression = consort.AttachmentExpression(
88     ...     attachments=spannertools.Slur(),
89     ... )
90     >>> staff = Staff("c'4 d'4 e'4 f'4")
91     >>> attachment_expression(staff)
92     >>> print(format(staff))
93     \new Staff {
94         c'4 (
95         d'4
96         e'4
97         f'4 )
98     }
99
100 .. container:: example
101
102 :::
103
104     >>> staff = Staff("c'4 d'4 e'4 f'4")
105     >>> attachment_expression = consort.AttachmentExpression(
106     ...     attachments=[
107     ...         [
108         ...             indicatortools.Articulation('accent'),
109         ...             indicatortools.Articulation('staccato'),
110         ...         ],
111         ...     ],
112         ...     selector=selectortools.Selector().by_logical_tie()[0]
113         ... )
114     >>> attachment_expression(staff)
115     >>> print(format(staff))
116     \new Staff {
117         c'4 -\accent -\staccato
118         d'4 -\accent -\staccato
119         e'4 -\accent -\staccato
120         f'4 -\accent -\staccato
121     }
122
123 """
124
125 ### CLASS VARIABLES ###
126
127 __slots__ = (
128     '_attachments',
129     '_scope',

```

```

130     '_selector',
131     '_is_annotation',
132     '_is_destructive',
133 )
134
135     ### INITIALIZER ###
136
137     def __init__(
138         self,
139         attachments=None,
140         selector=None,
141         scope=None,
142         is_annotation=None,
143         is_destructive=None,
144     ):
145         HashCachingObject.__init__(self)
146         if attachments is not None:
147             if not isinstance(attachments, collections.Sequence):
148                 attachments = (attachments,)
149             attachments = datastructuretools.TypedList(attachments)
150         self._attachments = attachments
151         if selector is not None:
152             assert isinstance(selector, selectortools.Selector)
153         self._selector = selector
154         if scope is not None:
155             if isinstance(scope, type):
156                 assert issubclass(scope, scoretools.Component)
157             else:
158                 assert isinstance(scope, (scoretools.Component, str)))
159         self._scope = scope
160         if is_annotation is not None:
161             is_annotation = bool(is_annotation)
162         self._is_annotation = is_annotation
163         if is_destructive is not None:
164             is_destructive = bool(is_destructive)
165         self._is_destructive = is_destructive
166
167     ### PUBLIC METHODS ###
168
169     def __call__(
170         self,
171         music,
172         name=None,
173         rotation=0,
174     ):
175         selector = self.selector
176         if selector is None:
177             selector = selectortools.Selector()
178         selections = selector(music, rotation=rotation)
179         self._apply_attachments(
180             selections,
181             name=name,
182             rotation=rotation,
183         )

```

```

184
185     ### PRIVATE METHODS ####
186
187     def _apply_attachments(self, selections, name=None, rotation=None):
188         if not self.attachments:
189             return
190         all_attachments = datastructuretools.CyclicTuple(self.attachments)
191         for i, selection in enumerate(selections, rotation):
192             attachments = all_attachments[i]
193             # print(i, selection, attachments)
194             if attachments is None:
195                 continue
196             if not isinstance(attachments, collections.Sequence):
197                 attachments = (attachments,)
198             for attachment in attachments:
199                 # print('\t' + repr(attachment))
200                 # spanners
201                 if isinstance(attachment, spannertools.Spanner):
202                     attachment = copy.copy(attachment)
203                     attach(attachment, selection, name=name)
204                 elif isinstance(attachment, type) and \
205                     issubclass(attachment, spannertools.Spanner):
206                     attachment = attachment()
207                     attach(attachment, selection, name=name)
208                 # expressions
209                 elif hasattr(attachment, '__call__'):
210                     signature = funcsigs.signature(attachment.__call__)
211                     if 'seed' in signature.parameters:
212                         attachment(selection, seed=rotation, name=name)
213                     elif 'rotation' in signature.parameters:
214                         attachment(selection, rotation=rotation, name=name)
215                     elif 'name' in signature.parameters:
216                         attachment(selection, name=name)
217                     else:
218                         attachment(selection)
219                 # indicators
220             else:
221                 if isinstance(selection, scoretools.Leaf):
222                     selection = (selection,)
223                 for component in selection:
224                     attachment = copy.copy(attachment)
225                     attach(
226                         attachment,
227                         component,
228                         name=name,
229                         scope=self.scope,
230                         is_annotation=self.is_annotation,
231                     )
232
233     ### PUBLIC METHODS ####
234
235     def reverse(self):
236         attachments = sequencetools.Sequence(*self.attachments)
237         return new(self,

```

```

238         attachments=attachments.reverse(),
239     )
240
241     def rotate(self, n=0):
242         attachments = sequencetools.Sequence(*self.attachments)
243         return new(self,
244             attachments=attachments.rotate(n),
245         )
246
247     ### PUBLIC PROPERTIES ###
248
249     @property
250     def attachments(self):
251         return self._attachments
252
253     @property
254     def is_annotation(self):
255         return self._is_annotation
256
257     @property
258     def is_destructive(self):
259         return self._is_destructive
260
261     @property
262     def scope(self):
263         return self._scope
264
265     @property
266     def selector(self):
267         return self._selector

```

## A.5 CONSORT.TOOLS.ATTACHMENTHANDLER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import collections
4 from abjad import inspect_
5 from abjad import iterate
6 from abjad.tools import abctools
7 from abjad.tools import scoretools
8 from abjad.tools import selectortools
9 from abjad.tools import systemtools
10
11
12 class AttachmentHandler(abctools.AbjadValueObject):
13     r'''An attachment maker.
14
15     :::
16
17     >>> import consort
18     >>> attachment_handler = consort.AttachmentHandler()
19     >>> print(format(attachment_handler))
20     consort.tools.AttachmentHandler()
21

```

```

22     """
23
24     """ CLASS VARIABLES """
25
26     __slots__ = (
27         '_attachment_expressions',
28     )
29
30     """ INITIALIZER """
31
32     def __init__(
33         self,
34         **attachment_expressions
35     ):
36         import consort
37         prototype = consort.AttachmentExpression
38         validated_attachment_expressions = {}
39         for name, attachment_expression in attachment_expressions.items():
40             if attachment_expression is None:
41                 continue
42             if not isinstance(attachment_expression, prototype):
43                 attachment_expression = consort.AttachmentExpression(
44                     attachments=attachment_expression,
45                 )
46             validated_attachment_expressions[name] = attachment_expression
47         self._attachment_expressions = validated_attachment_expressions
48
49     """ SPECIAL METHODS """
50
51     def __call__(
52         self,
53         music,
54         seed=0,
55     ):
56         assert isinstance(music, scoretools.Container)
57         if not self.attachment_expressions:
58             return
59         items = self.attachment_expressions.items()
60
61         #print('\nnondestructive...')
62
63         destructive_expressions = set()
64         selectors = set()
65         selectors_to_expressions = {}
66         for item in items:
67             name, attachment_expression = item
68             if attachment_expression.is_destructive:
69                 destructive_expressions.add(item)
70                 continue
71             selector = attachment_expression.selector
72             if selector is None:
73                 selector = selectortools.Selector()
74             selectors.add(selector)
75             if selector not in selectors_to_expressions:

```

```

76         selectors_to_expressions[selector] = set()
77         selectors_to_expressions[selector].add(item)
78     selectors_to_selections = selectortools.Selector.run_selectors(
79         music, selectors,
80         rotation=seed,
81     )
82     for selector in selectors:
83         expressions = selectors_to_expressions[selector]
84         selections = selectors_to_selections[selector]
85         for name, attachment_expression in expressions:
86             attachment_expression._apply_attachments(
87                 selections,
88                 name=name,
89                 rotation=seed,
90             )
91
92         #print('\t...done')
93         if destructive_expressions:
94             #print('destructive...')
95             for name, attachment_expression in destructive_expressions:
96                 #print('\t', name)
97                 attachment_expression(
98                     music,
99                     name=name,
100                    rotation=seed,
101                )
102             #print('\t...done')
103
104     def __getattr__(self, item):
105         if item in self.attachment_expressions:
106             return self.attachment_expressions[item]
107         return object.__getattribute__(self, item)
108
109     ### PRIVATE METHODS ###
110
111     @staticmethod
112     def _process_session(segment_maker, verbose=None):
113         import consort
114         score = segment_maker.score
115         counter = collections.Counter()
116         for voice in iterate(score).by_class(scoretools.Voice):
117             progress_indicator = systemtools.ProgressIndicator(
118                 message='decorating {}'.format(voice.name),
119                 verbose=verbose,
120             )
121             with progress_indicator:
122                 for container in voice:
123                     prototype = consort.MusicSpecifier
124                     music_specifier = inspect_(container).get_effective(prototype)
125                     maker = music_specifier.attachment_handler
126                     if maker is None:
127                         continue
128                     if music_specifier not in counter:
129                         seed = music_specifier.seed or 0

```

```

130         counter[musicSpecifier] = seed
131         seed = counter[musicSpecifier]
132         maker(container, seed=seed)
133         counter[musicSpecifier] -= 1
134         progressIndicator.advance()
135
136     ### PRIVATE PROPERTIES ###
137
138     @property
139     def _storageFormatSpecification(self):
140         return systemtools.StorageFormatSpecification(
141             self,
142             keywordArgumentNames=sorted(self.attachmentExpressions.keys()),
143         )
144
145     ### PUBLIC PROPERTIES ###
146
147     @property
148     def attachmentExpressions(self):
149         return self._attachmentExpressions.copy()

```

## A.6 CONSORT.TOOLS.ATTACKPOINTSIGNATURE

```

1 # -*- encoding: utf-8 -*-
2 from abjad import inspect_
3 from abjad import iterate
4 from abjad.tools import abctools
5 from abjad.tools import durationtools
6 from abjad.tools import scoretools
7
8
9 class AttackPointSignature(abctools.AbjadValueObject):
10     r'''An attack point signature.
11
12     :::
13
14     >>> import consort
15     >>> attack_point_signature = consort.AttackPointSignature(
16         ...     divisionPosition=0,
17         ...     phrasePosition=(1, 2),
18         ...     segmentPosition=(4, 5),
19         ...     )
20     >>> print(format(attack_point_signature))
21     consort.tools.AttackPointSignature(
22         divisionIndex=0,
23         divisionPosition=durationtools.Multiplier(0, 1),
24         logicalTieIndex=0,
25         phrasePosition=durationtools.Multiplier(1, 2),
26         segmentPosition=durationtools.Multiplier(4, 5),
27         totalDivisionsInPhrase=1,
28         totalLogicalTiesInDivision=1,
29         )
30
31     '''

```

```

32
33     ### CLASS VARIABLES ###
34
35     __slots__ = (
36         '_division_index',
37         '_division_position',
38         '_logical_tie_index',
39         '_phrase_position',
40         '_segment_position',
41         '_total_divisions_in_phrase',
42         '_total_logical_ties_in_division',
43     )
44
45     ### INITIALIZER ###
46
47     def __init__(
48         self,
49         division_index=0,
50         division_position=0,
51         logical_tie_index=0,
52         phrase_position=0,
53         segment_position=0,
54         total_divisions_in_phrase=1,
55         total_logical_ties_in_division=1,
56     ):
57         division_index = int(division_index)
58         division_position = durationtools.Multiplier(division_position)
59         logical_tie_index = int(logical_tie_index)
60         phrase_position = durationtools.Multiplier(phrase_position)
61         segment_position = durationtools.Multiplier(segment_position)
62         total_divisions_in_phrase = int(total_divisions_in_phrase)
63         total_logical_ties_in_division = int(total_logical_ties_in_division)
64         assert 0 <= logical_tie_index < total_logical_ties_in_division
65         assert 0 <= division_index < total_divisions_in_phrase
66         assert 0 <= division_position <= 1
67         assert 0 <= phrase_position <= 1
68         assert 0 <= segment_position <= 1
69         self._division_index = division_index
70         self._division_position = division_position
71         self._logical_tie_index = logical_tie_index
72         self._phrase_position = phrase_position
73         self._segment_position = segment_position
74         self._total_divisions_in_phrase = total_divisions_in_phrase
75         self._total_logical_ties_in_division = \
76             total_logical_ties_in_division
77
78     ### PRIVATE METHODS ###
79
80     @staticmethod
81     def _find_position(
82         logical_tie_start_offset,
83         bounding_start_offset,
84         bounding_stop_offset,
85     ):

```

```

86     duration = bounding_stop_offset - bounding_start_offset
87     start_offset = logical_tie_start_offset - bounding_start_offset
88     if duration == 0:
89         return durationtools.Multiplier(0)
90     position = start_offset / duration
91     assert 0 <= position <= 1
92     return position
93
94     ### PUBLIC METHODS ###
95
96     @classmethod
97     def from_logical_tie(cls, logical_tie):
98         import consort
99         logical_tie_start_offset = logical_tie.get_timespan().start_offset
100
101        phrase = consort.SegmentMaker.logical_tie_to_phrase(logical_tie)
102        phrase_logical_ties = cls._collect_logical_ties(phrase)
103        phrase_position = cls._find_position(
104            logical_tie_start_offset,
105            phrase_logical_ties[0].get_timespan().start_offset,
106            phrase_logical_ties[-1].get_timespan().start_offset,
107            )
108
109        division = consort.SegmentMaker.logical_tie_to_division(logical_tie)
110        division_index = phrase.index(division)
111        total_divisions_in_phrase = len(phrase)
112        division_logical_ties = cls._collect_logical_ties(division)
113        division_position = cls._find_position(
114            logical_tie_start_offset,
115            division_logical_ties[0].get_timespan().start_offset,
116            division_logical_ties[-1].get_timespan().start_offset,
117            )
118
119        logical_tie_index = division_logical_ties.index(logical_tie)
120        total_logical_ties_in_division = len(division_logical_ties)
121
122        voice = consort.SegmentMaker.logical_tie_to_voice(logical_tie)
123        segment_timespan = inspect_(voice).get_timespan()
124        segment_position = cls._find_position(
125            logical_tie_start_offset,
126            segment_timespan.start_offset,
127            segment_timespan.stop_offset,
128            )
129
130        signature = cls(
131            division_index=division_index,
132            division_position=division_position,
133            logical_tie_index=logical_tie_index,
134            phrase_position=phrase_position,
135            segment_position=segment_position,
136            total_divisions_in_phrase=total_divisions_in_phrase,
137            total_logical_ties_in_division=total_logical_ties_in_division,
138            )
139        return signature

```

```

140
141     ### PRIVATE METHODS #####
142
143     @staticmethod
144     def _collect_logical_ties(container):
145         logical_ties = []
146         for leaf in iterate(container).by_class(scoretools.Note):
147             leaf_logical_tie = inspect_(leaf).get_logical_tie()
148             if leaf is not leaf_logical_tie.head:
149                 continue
150             logical_ties.append(leaf_logical_tie)
151     return logical_ties
152
153     ### PUBLIC PROPERTIES #####
154
155     @property
156     def division_index(self):
157         return self._division_index
158
159     @property
160     def division_position(self):
161         return self._division_position
162
163     @property
164     def is_first_of_division(self):
165         if not self.logical_tie_index:
166             return True
167         return False
168
169     @property
170     def is_first_of_phrase(self):
171         if not self.logical_tie_index and not self.division_index:
172             return True
173         return False
174
175     @property
176     def logical_tie_index(self):
177         return self._logical_tie_index
178
179     @property
180     def phrase_position(self):
181         return self._phrase_position
182
183     @property
184     def segment_position(self):
185         return self._segment_position
186
187     @property
188     def total_divisions_in_phrase(self):
189         return self._total_divisions_in_phrase
190
191     @property
192     def total_logical_ties_in_division(self):
193         return self._total_logical_ties_in_division

```

## A.7 CONSORT.TOOLS.BOUNDARYTIMESPANMAKER

```
1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import timespanools
6 from consort.tools.TimespanMaker import TimespanMaker
7
8
9 class BoundaryTimespanMaker(TimespanMaker):
10     r'''A boundary timespan-maker.
11
12     :::
13
14     >>> import consort
15     >>> timespan_maker = consort.BoundaryTimespanMaker(
16         ...     start_talea=rhythmmakertools.Talea(
17             ...         counts=[1],
18             ...         denominator=2,
19             ...         ),
20             ...     stop_talea=rhythmmakertools.Talea(
21                 ...         counts=[1],
22                 ...         denominator=4,
23                 ...         ),
24             ...     voice_names=('Violin 1 Voice', 'Violin 2 Voice'),
25             ...         )
26     >>> print(format(timespan_maker))
27     consort.tools.BoundaryTimespanMaker(
28         start_talea=rhythmmakertools.Talea(
29             counts=(1,),
30             denominator=2,
31             ),
32         stop_talea=rhythmmakertools.Talea(
33             counts=(1,),
34             denominator=4,
35             ),
36         voice_names=('Violin 1 Voice', 'Violin 2 Voice'),
37         )
38
39     :::
40
41     >>> timespan_inventory = timespanools.TimespanInventory([
42         ...     consort.PerformedTimespan(
43             ...         start_offset=0,
44             ...         stop_offset=1,
45             ...         voice_name='Violin 1 Voice',
46             ...         ),
47             ...     consort.PerformedTimespan(
48                 ...         start_offset=(1, 2),
49                 ...         stop_offset=(3, 2),
50                 ...         voice_name='Violin 2 Voice',
51                 ...         ),
52             ...     consort.PerformedTimespan(
```

```

53     ...         start_offset=3,
54     ...         stop_offset=4,
55     ...         voice_name='Violin 2 Voice',
56     ...         ),
57     ...     ])
58
59 :::
60
61 >>> music_specifiers = {'Cello Voice': None}
62 >>> target_timespan = timespantools.Timespan(0, 10)
63 >>> timespan_inventory = timespan_maker(
64     ...     music_specifiers=music_specifiers,
65     ...     target_timespan=target_timespan,
66     ...     timespan_inventory=timespan_inventory,
67     ...     )
68 >>> print(format(timespan_inventory))
69 timespantools.TimespanInventory(
70     [
71         consort.tools.PerformedTimespan(
72             start_offset=durationtools.Offset(0, 1),
73             stop_offset=durationtools.Offset(1, 2),
74             voice_name='Cello Voice',
75             ),
76         consort.tools.PerformedTimespan(
77             start_offset=durationtools.Offset(0, 1),
78             stop_offset=durationtools.Offset(1, 1),
79             voice_name='Violin 1 Voice',
80             ),
81         consort.tools.PerformedTimespan(
82             start_offset=durationtools.Offset(1, 2),
83             stop_offset=durationtools.Offset(3, 2),
84             voice_name='Violin 2 Voice',
85             ),
86         consort.tools.PerformedTimespan(
87             start_offset=durationtools.Offset(3, 2),
88             stop_offset=durationtools.Offset(7, 4),
89             voice_name='Cello Voice',
90             ),
91         consort.tools.PerformedTimespan(
92             start_offset=durationtools.Offset(3, 1),
93             stop_offset=durationtools.Offset(7, 2),
94             voice_name='Cello Voice',
95             ),
96         consort.tools.PerformedTimespan(
97             start_offset=durationtools.Offset(3, 1),
98             stop_offset=durationtools.Offset(4, 1),
99             voice_name='Violin 2 Voice',
100            ),
101        consort.tools.PerformedTimespan(
102            start_offset=durationtools.Offset(4, 1),
103            stop_offset=durationtools.Offset(17, 4),
104            voice_name='Cello Voice',
105            ),
106    ]

```

```

107         )
108
109     """
110
111     """ CLASS VARIABLES """
112
113     __slots__ = (
114         '_labels',
115         '_start_talea',
116         '_start_groupings',
117         '_stop_talea',
118         '_stop_groupings',
119         '_voice_names',
120     )
121
122     """ INITIALIZER """
123
124     def __init__(
125         self,
126         start_talea=None,
127         stop_talea=None,
128         start_groupings=None,
129         stop_groupings=None,
130         labels=None,
131         output_masks=None,
132         padding=None,
133         seed=None,
134         timespanSpecifier=None,
135         voice_names=None,
136     ):
137         TimespanMaker.__init__(
138             self,
139             output_masks=output_masks,
140             padding=padding,
141             seed=seed,
142             timespanSpecifier=timespanSpecifier,
143         )
144
145     if start_talea is not None:
146         if not isinstance(start_talea, rhythmmakertools.Talea):
147             start_duration = durationtools.Duration(start_talea)
148             counts = [start_duration.numerator]
149             denominator = start_duration.denominator
150             start_talea = rhythmmakertools.Talea(
151                 counts=counts,
152                 denominator=denominator,
153             )
154             assert isinstance(start_talea, rhythmmakertools.Talea)
155             assert start_talea.counts
156             assert all(0 < x for x in start_talea.counts)
157             self._start_talea = start_talea
158
159     if start_groupings is not None:
160         if not isinstance(start_groupings, collections.Sequence):

```

```

161         start_groupings = (start_groupings,)
162         start_groupings = tuple(int(x) for x in start_groupings)
163         assert len(start_groupings)
164         assert all(0 < x for x in start_groupings)
165         self._start_groupings = start_groupings
166
167     if stop_talea is not None:
168         if not isinstance(stop_talea, rhythmmakertools.Talea):
169             stop_duration = durationtools.Duration(stop_talea)
170             counts = [stop_duration.numerator]
171             denominator = stop_duration.denominator
172             stop_talea = rhythmmakertools.Talea(
173                 counts=counts,
174                 denominator=denominator,
175             )
176             assert isinstance(stop_talea, rhythmmakertools.Talea)
177             assert stop_talea.counts
178             assert all(0 < x for x in stop_talea.counts)
179             self._stop_talea = stop_talea
180
181     if stop_groupings is not None:
182         if not isinstance(stop_groupings, collections.Sequence):
183             stop_groupings = (stop_groupings,)
184             stop_groupings = tuple(int(x) for x in stop_groupings)
185             assert len(stop_groupings)
186             assert all(0 < x for x in stop_groupings)
187             self._stop_groupings = stop_groupings
188
189     if labels is not None:
190         if isinstance(labels, str):
191             labels = (labels,)
192             labels = tuple(str(_) for _ in labels)
193             self._labels = labels
194
195     if voice_names is not None:
196         voice_names = tuple(voice_names)
197         self._voice_names = voice_names
198
199     ### PRIVATE METHODS ###
200
201     def _collect_preexisting_timespans(
202         self,
203         target_timespan=None,
204         timespan_inventory=None,
205     ):
206         import consort
207         preexisting_timespans = timespantools.TimespanInventory()
208         for timespan in timespan_inventory:
209             if not isinstance(timespan, consort.PerformedTimespan):
210                 continue
211             if (
212                 self.voice_names and
213                 timespan.voice_name not in self.voice_names
214             ):

```

```

215         continue
216     if not self.labels:
217         preexisting_timespans.append(timespan)
218     elif not hasattr(timespan, 'music_specifier') or \
219         not timespan.music_specifier or \
220         not timespan.music_specifier.labels:
221         continue
222     elif any(label in timespan.music_specifier.labels
223             for label in self.labels):
224         preexisting_timespans.append(timespan)
225     preexisting_timespans & target_timespan
226     return preexisting_timespans
227
228 def _make_timespans(
229     self,
230     layer=None,
231     music_specifiers=None,
232     target_timespan=None,
233     timespan_inventory=None,
234 ):
235     import consort
236
237     new_timespans = timespantools.TimespanInventory()
238     if not self.voice_names and not self.labels:
239         return new_timespans
240
241     start_talea = self.start_talea
242     if start_talea is None:
243         start_talea = rhythmmakertools.Talea((0,), 1)
244     start_talea = consort.Cursor(start_talea)
245
246     start_groupings = self.start_groupings
247     if start_groupings is None:
248         start_groupings = (1,)
249     start_groupings = consort.Cursor(start_groupings)
250
251     stop_talea = self.stop_talea
252     if stop_talea is None:
253         stop_talea = rhythmmakertools.Talea((0,), 1)
254     stop_talea = consort.Cursor(stop_talea)
255
256     stop_groupings = self.stop_groupings
257     if stop_groupings is None:
258         stop_groupings = (1,)
259     stop_groupings = consort.Cursor(stop_groupings)
260
261     if self.seed:
262         if self.seed < 0:
263             for _ in range(abs(self.seed)):
264                 start_talea.backtrack()
265                 start_groupings.backtrack()
266                 stop_talea.backtrack()
267                 stop_groupings.backtrack()
268         else:

```

```

269         next(start_talea)
270         next(start_groupings)
271         next(stop_talea)
272         next(stop_groupings)
273
274     context_counter = collections.Counter()
275     preexisting_timepans = self._collect_preexisting_timepans(
276         target_timespan=target_timespan,
277         timepan_inventory=timepan_inventory,
278     )
279     new_timepan_mapping = {}
280     for group_index, group in enumerate(
281         preexisting_timepans.partition(True)
282     ):
283         for context_name, music_specifier in music_specifiers.items():
284             if context_name not in new_timepan_mapping:
285                 continue
286             new_timepan_mapping[context_name] = group.timepan
287         for context_name, music_specifier in music_specifiers.items():
288             if context_name not in new_timepan_mapping:
289                 new_timepan_mapping[context_name] = \
290                     timespantools.TimespanInventory()
291             context_seed = context_counter[context_name]
292             start_durations = []
293             for _ in range(next(start_groupings)):
294                 start_durations.append(next(start_talea))
295             stop_durations = []
296             for _ in range(next(stop_groupings)):
297                 stop_durations.append(next(stop_talea))
298             start_timepans, stop_timepans = (), ()
299             if start_durations:
300                 start_timepans = music_specifier(
301                     durations=start_durations,
302                     layer=layer,
303                     output_masks=self.output_masks,
304                     padding=self.padding,
305                     seed=context_seed,
306                     start_offset=group.start_offset,
307                     timepan_specifier=self.timepan_specifier,
308                     voice_name=context_name,
309                 )
310                 context_counter[context_name] += 1
311             if stop_durations:
312                 stop_timepans = music_specifier(
313                     durations=stop_durations,
314                     layer=layer,
315                     output_masks=self.output_masks,
316                     padding=self.padding,
317                     seed=context_seed,
318                     start_offset=group.stop_offset,
319                     timepan_specifier=self.timepan_specifier,
320                     voice_name=context_name,
321                 )
322             context_counter[context_name] += 1

```

```

323         if start_timespans and stop_timespans:
324             start_timespans & group.timespan
325             new_timespan_mapping[context_name].extend(start_timespans)
326             new_timespan_mapping[context_name].extend(stop_timespans)
327             for context_name, timespans in new_timespan_mapping.items():
328                 timespans.compute_logical_or()
329                 new_timespans.extend(timespans)
330             return new_timespans
331
332     ### PUBLIC PROPERTIES ###
333
334     @property
335     def labels(self):
336         return self._labels
337
338     @property
339     def start_talea(self):
340         return self._start_talea
341
342     @property
343     def stop_talea(self):
344         return self._stop_talea
345
346     @property
347     def start_groupings(self):
348         return self._start_groupings
349
350     @property
351     def stop_groupings(self):
352         return self._stop_groupings
353
354     @property
355     def voice_names(self):
356         return self._voice_names

```

## A.8 CONSORT.TOOLS.CHORDEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import attach
4 from abjad import mutate
5 from abjad.tools import indicatortools
6 from abjad.tools import pitchtools
7 from abjad.tools import scoretools
8 from abjad.tools import selectiontools
9 from consort.tools.LogicalTieExpression import LogicalTieExpression
10
11
12 class ChordExpression(LogicalTieExpression):
13     r'''A chord expression.
14
15     :::
16
17     >>> import consort

```

```

18     >>> chord_expression = consort.ChordExpression(
19         ...     arpeggio_direction=Down,
20         ...     chord_expr=(-1, 3, 7),
21         ... )
22     >>> print(format(chord_expression))
23     consort.tools.ChordExpression(
24         chord_expr=pitchtools.IntervalSegment(
25             (
26                 pitchtools.NumberedInterval(-1),
27                 pitchtools.NumberedInterval(3),
28                 pitchtools.NumberedInterval(7),
29                 ),
30                 item_class=pitchtools.NumberedInterval,
31                 ),
32                 arpeggio_direction=Down,
33             )
34
35 :::
36
37     >>> staff = Staff(r"c'4 d'4 \p -\accent ~ d'4 e'4")
38     >>> pitch_range = pitchtools.PitchRange.from_pitches('C3', 'C6')
39     >>> attach(pitch_range, staff, scope=Staff)
40     >>> logical_tie = inspect_(staff[1]).get_logical_tie()
41     >>> chord_expression(logical_tie)
42     >>> print(format(staff))
43     \new Staff {
44         c'4
45         \arpeggioArrowDown
46         <cs' f' a'>4 -\accent \arpeggio \p ~
47         <cs' f' a'>4
48         e'4
49     }
50
51     ...
52
53     ### CLASS VARIABLES ###
54
55     __slots__ = (
56         '_arpeggio_direction',
57         '_chord_expr',
58     )
59
60     ### INITIALIZER ###
61
62     def __init__(
63         self,
64         chord_expr=None,
65         arpeggio_direction=None,
66     ):
67         assert arpeggio_direction in (Up, Down, Center, None)
68         if chord_expr is not None:
69             assert len(chord_expr)
70             prototype = (pitchtools.IntervalSegment, pitchtools.PitchSegment)
71             if isinstance(chord_expr, prototype):

```

```

72         result = chord_expr
73     elif isinstance(chord_expr, str):
74         result = pitchtools.PitchSegment(chord_expr)
75     else:
76         try:
77             result = sorted(chord_expr)
78             result = pitchtools.IntervalSegment(result)
79         except:
80             result = pitchtools.PitchSegment(chord_expr)
81         chord_expr = result
82     self._arpeggio_direction = arpeggio_direction
83     self._chord_expr = chord_expr
84
85     ### SPECIAL METHODS ###
86
87     def __call__(
88         self,
89         logical_tie,
90         pitch_range=None,
91     ):
92         assert isinstance(logical_tie, selectiontools.LogicalTie), logical_tie
93         if isinstance(self.chord_expr, pitchtools.IntervalSegment):
94             pitches = self._get_pitches_from_intervals(
95                 logical_tie.head.written_pitch,
96                 pitch_range,
97                 logical_tie,
98             )
99         else:
100            pitches = self.chord_expr
101        if len(pitches) == 2:
102            interval = pitches[0] - pitches[1]
103            if interval.quality_string in ('augmented', 'diminished'):
104                chord = scoretools.Chord(pitches, 1)
105                mutate(chord).respell_with_sharps()
106                pitches = chord.written_pitches
107                interval = pitches[0] - pitches[1]
108            if interval.quality_string in ('augmented', 'diminished'):
109                chord = scoretools.Chord(pitches, 1)
110                mutate(chord).respell_with_flats()
111                pitches = chord.written_pitches
112            for i, leaf in enumerate(logical_tie):
113                chord = scoretools.Chord(leaf)
114                chord.written_pitches = pitches
115                self._replace(leaf, chord)
116                if not i and self.arpeggio_direction is not None:
117                    arpeggio = indicatortools.Arpeggio(self.arpeggio_direction)
118                    attach(arpeggio, chord)
119
120     ### PRIVATE METHODS ###
121
122     @staticmethod
123     def _score_pitch_set(pitch_set):
124         buckets = {}
125         for pitch in pitch_set:

```

```

126     if pitch.diatonic_pitch_number not in buckets:
127         buckets[pitch.diatonic_pitch_number] = set()
128         buckets[pitch.diatonic_pitch_number].add(pitch)
129     penalty = 0
130     for diatonic_pitch_number, bucket in buckets.items():
131         penalty = penalty + (len(bucket) - 1)
132     return penalty
133
134     @staticmethod
135     def _flip_accidental(pitch):
136         if not pitch.alteration_in_semitones:
137             return pitch
138         elif 0 < pitch.alteration_in_semitones:
139             return pitch.respell_with_flats()
140         return pitch.respell_with_sharps()
141
142     @staticmethod
143     def _reshape_pitch_set(pitch_set):
144         altered = {}
145         unaltered = {}
146         for pitch in pitch_set:
147             diatonic_pitch_number = pitch.diatonic_pitch_number
148             if pitch.alteration_in_semitones:
149                 if diatonic_pitch_number not in altered:
150                     altered[diatonic_pitch_number] = []
151                     altered[diatonic_pitch_number].append(pitch)
152                 else:
153                     unaltered[diatonic_pitch_number] = pitch
154         new_altered = {}
155         for diatonic_pitch_number, altered_pitches in altered.items():
156             while altered_pitches:
157                 altered_pitch = altered_pitches.pop()
158                 if diatonic_pitch_number not in unaltered and \
159                     diatonic_pitch_number not in new_altered:
160                     if diatonic_pitch_number not in new_altered:
161                         new_altered[diatonic_pitch_number] = []
162                         new_altered[diatonic_pitch_number].append(altered_pitch)
163                     else:
164                         new_pitch = ChordExpression._flip_accidental(altered_pitch)
165                         new_diatonic_pitch_number = new_pitch.diatonic_pitch_number
166                         if new_diatonic_pitch_number not in new_altered:
167                             new_altered[new_diatonic_pitch_number] = []
168                             new_altered[new_diatonic_pitch_number].append(new_pitch)
169             result = set(unaltered.values())
170             for altered_pitches in new_altered.values():
171                 result.update(altered_pitches)
172             result = pitchtools.PitchSet(result)
173             return result
174
175     @staticmethod
176     def _respell_pitch_set(pitch_set):
177         r'''Respell pitch set.
178
179         :::
```

```

180
181     >>> pitch_set = pitchtools.PitchSet("c' e' g''")
182     >>> consort.ChordExpression._respell_pitch_set(pitch_set)
183     PitchSet(["c'", "e'", "g'"])
184
185     :::
186
187     >>> pitch_set = pitchtools.PitchSet("c' e' g' c'' cs'' g'''")
188     >>> consort.ChordExpression._respell_pitch_set(pitch_set)
189     PitchSet(["c'", "e'", "g'", "c''", "df'''", "g'''"])
190
191     :::
192
193     >>> pitch_set = pitchtools.PitchSet("bf d' f' fs' bf' f'''")
194     >>> consort.ChordExpression._respell_pitch_set(pitch_set)
195     PitchSet(['bf', "d''", "f'", "gf'''", "bf'", "f'''"])
196
197     :::
198
199     >>> pitch_set = pitchtools.PitchSet("b' c'' cs'' d'''")
200     >>> consort.ChordExpression._respell_pitch_set(pitch_set)
201     PitchSet(["b'", "c''", "cs''", "d'''"])
202
203     :::
204
205     >>> pitch_set = pitchtools.PitchSet("cf' c' cs'")
206     >>> consort.ChordExpression._respell_pitch_set(pitch_set)
207     PitchSet(['b', "c'", "df'"])
208
209     :::
210
211     >>> pitch_set = pitchtools.PitchSet("e ff f fs g")
212     >>> consort.ChordExpression._respell_pitch_set(pitch_set)
213     PitchSet(['e', 'f', 'gf', 'g'])
214
215     """
216     initial_score = ChordExpression._score_pitch_set(pitch_set)
217     if not initial_score:
218         return pitch_set
219     flat_pitch_set = pitchtools.PitchSet(
220         _.respell_with_flats()
221         for _ in pitch_set
222     )
223     flat_score = ChordExpression._score_pitch_set(flat_pitch_set)
224     if not flat_score:
225         return flat_pitch_set
226     sharp_pitch_set = pitchtools.PitchSet(
227         _.respell_with_sharps()
228         for _ in pitch_set
229     )
230     sharp_score = ChordExpression._score_pitch_set(sharp_pitch_set)
231     if not sharp_score:
232         return sharp_pitch_set
233     scored_pitch_sets = [

```

```

234         (initial_score, pitch_set),
235         (flat_score, flat_pitch_set),
236         (sharp_score, sharp_pitch_set),
237     ]
238 scored_pitch_sets.sort()
239 reshaped_pitch_set = ChordExpression._reshape_pitch_set(
240     scored_pitch_sets[0][1])
241 reshaped_score = ChordExpression._score_pitch_set(reshaped_pitch_set)
242 scored_pitch_sets.append((reshaped_score, reshaped_pitch_set))
243 scored_pitch_sets.sort()
244 return scored_pitch_sets[0][1]
245
246 def _get_pitches_from_intervals(self, base_pitch, pitch_range, logical_tie):
247     import consort
248     chord_expr = self.chord_expr or ()
249     new_chord_expr = chord_expr
250     if pitch_range is not None:
251         if base_pitch not in pitch_range:
252             print('Voice:', consort.SegmentMaker.logical_tie_to_voice(
253                 logical_tie).name)
254             print('Pitch range:', pitch_range)
255             print('Base pitch:', base_pitch)
256             print('Chord expression:', chord_expr)
257             print('MusicSpec:')
258             print(format(
259                 consort.SegmentMaker.logical_tie_to_music_specifier(
260                     logical_tie)))
261             raise Exception
262
263     sorted_intervals = sorted(chord_expr, key=lambda x: x.semitones)
264     maximum = sorted_intervals[-1]
265     maximum_pitch = base_pitch.transpose(maximum)
266     minimum = sorted_intervals[0]
267     minimum_pitch = base_pitch.transpose(minimum)
268     if maximum_pitch not in pitch_range:
269         new_chord_expr = [x - maximum for x in chord_expr]
270     elif minimum_pitch not in pitch_range:
271         new_chord_expr = [x - minimum for x in chord_expr]
272
273     pitches = [base_pitch.transpose(x) for x in new_chord_expr]
274     pitches = [pitchtools.NamedPitch(float(x)) for x in pitches]
275     if pitch_range is not None:
276         if not all(pitch in pitch_range for pitch in pitches):
277             print('Voice:', consort.SegmentMaker.logical_tie_to_voice(
278                 logical_tie).name)
279             print('Pitch range:', pitch_range)
280             print('Base pitch:', base_pitch)
281             print('Chord expression:', chord_expr)
282             print('Resulting pitches:', pitches)
283             print('MusicSpec:')
284             print(format(
285                 consort.SegmentMaker.logical_tie_to_music_specifier(
286                     logical_tie)))
287             raise Exception

```

```

288     pitch_set = self._respell_pitch_set(pitchtools.PitchSet(pitches))
289     pitches = pitchtools.PitchSegment(sorted(pitch_set))
290
291     return pitches
292
293     ##### PUBLIC PROPERTIES #####
294
295
296     @property
297     def arpeggio_direction(self):
298         return self._arpeggio_direction
299
300     @property
301     def chord_expr(self):
302         return self._chord_expr

```

## A.9 CONSORT.TOOLS.CHORDSPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3
4
5 class ChordSpecifier(abctools.AbjadValueObject):
6
7     ##### CLASS VARIABLES #####
8
9     __slots__ = ()

```

## A.10 CONSORT.TOOLS.CLEFSPANNER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import inspect_
4 from abjad.tools import indicatortools
5 from abjad.tools import scoretools
6 from abjad.tools import spannertools
7
8
9 class ClefSpanner(spannertools.Spanner):
10     r'''Clef spanner.
11
12     :::
13
14     >>> import consort
15     >>> staff = Staff("c' d' e' f' g' a' b' c''")
16     >>> clef = Clef('treble')
17     >>> attach(clef, staff[0])
18     >>> print(format(staff))
19
20     \new Staff {
21         \clef "treble"
22         c'4
23         d'4
24         e'4

```

```

24         f'4
25         g'4
26         a'4
27         b'4
28         c''4
29     }
30
31 :::
32
33     >>> clef_spinner = consort.ClefSpanner('percussion')
34     >>> attach(clef_spinner, staff[2:-2])
35     >>> print(format(staff))
36 \new Staff {
37     \clef "treble"
38     c'4
39     d'4
40     \clef "percussion"
41     e'4
42     f'4
43     g'4
44     a'4
45     \clef "treble"
46     b'4
47     c''4
48 }
49
50 :::
51
52     >>> staff = Staff("r4 c'4 d'4 r4 e'4 f'4 r4")
53     >>> clef = Clef('treble')
54     >>> attach(clef, staff[0])
55     >>> clef_spinner = consort.ClefSpanner('percussion')
56     >>> attach(clef_spinner, staff[1:3])
57     >>> clef_spinner = consort.ClefSpanner('percussion')
58     >>> attach(clef_spinner, staff[4:6])
59     >>> print(format(staff))
60 \new Staff {
61     \clef "treble"
62     r4
63     \clef "percussion"
64     c'4
65     d'4
66     r4
67     e'4
68     f'4
69     \clef "treble"
70     r4
71 }
72 """
73
74     ### CLASS VARIABLES ###
75
76     __slots__ = (
77

```

```

78     '_clef',
79 )
80
81     ### INITIALIZER ###
82
83     def __init__(
84         self,
85         clef='percussion',
86         overrides=None,
87     ):
88         spannertools.Spanner.__init__(
89             self,
90             overrides=overrides,
91         )
92         clef = indicatortools.Clef(clef)
93         self._clef = clef
94
95     ### SPECIAL METHODS ###
96
97     def __getnewargs__(self):
98         r'''Gets new arguments of spanner.
99
100        Returns empty tuple.
101        '''
102
103        return (
104            self.clef,
105        )
106
107     ### PRIVATE METHODS ###
108
109     def _copy_keyword_args(self, new):
110         new._clef = self.clef
111
112     def _get_lilypond_format_bundle(self, leaf):
113         import consort
114         lilypond_format_bundle = self._get_basic_lilypond_format_bundle(leaf)
115
116         prototype = (scoretools.Note, scoretools.Chord, type(None))
117
118         first_leaf = self._get_leaves()[0]
119         current_clef = inspect_(first_leaf).get_effective(indicatortools.Clef)
120
121         set_clef = False
122         reset_clef = False
123
124         if self._is_my_only_leaf(leaf):
125             consort.debug('ONLY', leaf)
126             if self.clef != current_clef:
127                 set_clef = True
128                 reset_clef = True
129
130             previous_leaf = inspect_(leaf).get_leaf(-1)
131             consort.debug('\tP', previous_leaf)
132             while not isinstance(previous_leaf, prototype):

```

```

132     previous_leaf = inspect_(previous_leaf).get_leaf(-1)
133     consort.debug('\tP', previous_leaf)
134     if previous_leaf is not None:
135         spanners = inspect_(previous_leaf).get_spanners(type(self))
136         spanners = tuple(spanners)
137         if spanners:
138             consort.debug('\tPREV?', spanners)
139             if spanners[0].clef == self.clef:
140                 set_clef = False
141
142             next_leaf = inspect_(leaf).get_leaf(1)
143             consort.debug('\tN', next_leaf)
144             while not isinstance(next_leaf, prototype):
145                 next_leaf = inspect_(next_leaf).get_leaf(1)
146                 consort.debug('\tN', next_leaf)
147                 if next_leaf is not None:
148                     spanners = inspect_(next_leaf).get_spanners(type(self))
149                     spanners = tuple(spanners)
150                     if spanners:
151                         consort.debug('\tNEXT?', spanners)
152                         if spanners[0].clef == self.clef:
153                             reset_clef = False
154
155             elif self._is_my_first_leaf(leaf):
156                 consort.debug('FIRST', leaf)
157                 if self.clef != current_clef:
158                     set_clef = True
159
160             previous_leaf = inspect_(leaf).get_leaf(-1)
161             consort.debug('\tP', previous_leaf)
162             while not isinstance(previous_leaf, prototype):
163                 previous_leaf = inspect_(previous_leaf).get_leaf(-1)
164                 consort.debug('\tP', previous_leaf)
165                 if previous_leaf is not None:
166                     spanners = inspect_(previous_leaf).get_spanners(type(self))
167                     spanners = tuple(spanners)
168                     if spanners:
169                         consort.debug('\tPREV?', spanners)
170                         if spanners[0].clef == self.clef:
171                             set_clef = False
172
173             elif self._is_my_last_leaf(leaf):
174                 consort.debug('LAST', leaf)
175                 if self.clef != current_clef and current_clef is not None:
176                     reset_clef = True
177
178             next_leaf = inspect_(leaf).get_leaf(1)
179             consort.debug('\tN', next_leaf)
180             while not isinstance(next_leaf, prototype):
181                 next_leaf = inspect_(next_leaf).get_leaf(1)
182                 consort.debug('\tN', next_leaf)
183                 if next_leaf is not None:
184                     spanners = inspect_(next_leaf).get_spanners(type(self))
185                     spanners = tuple(spanners)

```

```

186         if spanners:
187             consort.debug('\tNEXT?', spanners)
188             if spanners[0].clef == self.clef:
189                 reset_clef = False
190
191         if set_clef:
192             string = format(self.clef, 'lilypond')
193             lilypond_format_bundle.before.indicators.append(string)
194
195         if reset_clef and current_clef is not None:
196             string = format(current_clef, 'lilypond')
197             lilypond_format_bundle.after.indicators.append(string)
198
199     return lilypond_format_bundle
200
201     ### PUBLIC PROPERTIES ###
202
203     @property
204     def clef(self):
205         return self._clef

```

## A.11 CONSORTTOOLS.CLEFSPANNEREXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import attach
4 from abjad import inspect_
5 from abjad.tools import abctools
6 from abjad.tools import scoretools
7
8
9 class ClefSpannerExpression(abctools.AbjadValueObject):
10     r'''A clef spanner expression.
11     '''
12
13     ### CLASS VARIABLES ###
14
15     __slots__ = ()
16
17     ### INITIALIZER ###
18
19     def __init__(self):
20         pass
21
22     ### SPECIAL METHODS ###
23
24     def __call__(self, music, name=None):
25         import consort
26         leaves = music.select_leaves()
27         weights = []
28         weighted_pitches = []
29         for leaf in leaves:
30             weight = float(inspect_(leaf).get_duration())
31             if isinstance(leaf, scoretools.Note):

```

```

32         pitch = float(leaf.written_pitch)
33         weighted_pitch = pitch * weight
34         weights.append(weight)
35         weighted_pitches.append(weighted_pitch)
36     elif isinstance(leaf, scoretools.Chord):
37         for pitch in leaf.written_pitches:
38             pitch = float(pitch)
39             weighted_pitch = pitch * weight
40             weighted_pitches.append(weighted_pitch)
41             weights.append(weight)
42         sum_of_weights = sum(weights)
43         sum_of_weighted_pitches = sum(weighted_pitches)
44         weighted_average = sum_of_weighted_pitches / sum_of_weights
45     if weighted_average < 0:
46         clef_spinner = consort.ClefSpanner('bass')
47     else:
48         clef_spinner = consort.ClefSpanner('treble')
49     attach(clef_spinner, music, name=name)

```

## A.12 CONSORT.TOOLS.COMPLEXPIANOPEDALSPANNER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import spannertools
3
4
5 class ComplexPianoPedalSpanner(spannertools.Spanner):
6     r'''Complex piano pedal spanner.
7
8     :::
9
10    >>> import consort
11    >>> spanner = consort.ComplexPianoPedalSpanner()
12    >>> print(format(spanner))
13    consort.tools.ComplexPianoPedalSpanner(
14        include_inner_leaves=False,
15        )
16
17    '''
18
19    ### CLASS VARIABLES ###
20
21    __slots__ = (
22        '_include_inner_leaves',
23        )
24
25    ### INITIALIZER ###
26
27    def __init__(
28        self,
29        include_inner_leaves=False,
30        overrides=None,
31        ):
32        spannertools.Spanner.__init__(
33            self,

```

```

34         overrides=overrides,
35     )
36     self._include_inner_leaves = bool(include_inner_leaves)
37
38     ### PRIVATE PROPERTIES ###
39
40     def _copy_keyword_args(self, new):
41         new._include_inner_leaves = self.include_inner_leaves
42
43     def _get_lilypond_format_bundle(self, leaf):
44         lilypond_format_bundle = self._get_basic_lilypond_format_bundle(leaf)
45         if self._is_my_first_leaf(leaf):
46             string = r'\sustainOn'
47             lilypond_format_bundle.right.spanner_starts.append(string)
48         elif self.include_inner_leaves and not self._is_my_last_leaf(leaf):
49             string = r'\sustainOff \sustainOn'
50             lilypond_format_bundle.right.spanner_starts.append(string)
51         if self._is_my_last_leaf(leaf):
52             string = r'<> \sustainOff'
53             lilypond_format_bundle.after.indicators.append(string)
54         return lilypond_format_bundle
55
56     ### PUBLIC PROPERTIES ###
57
58     @property
59     def include_inner_leaves(self):
60         return self._include_inner_leaves

```

## A.13 CONSORTTOOLS.COMPLEXTEXTSPANNER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import durationtools
3 from abjad.tools import lilypondnametools
4 from abjad.tools import spannertools
5 from abjad.tools import markuptools
6 from abjad.tools import scoretools
7
8
9 class ComplexTextSpanner(spannertools.Spanner):
10     r'''A complex text spanner.
11
12     .. container:: example
13
14     :::
15
16         >>> import consort
17         >>> staff = Staff("c'4 d'4 r4 e'4")
18         >>> spanner_one = consort.ComplexTextSpanner(
19             ...     direction=Up,
20             ...     markup='foo',
21             ... )
22         >>> spanner_two = consort.ComplexTextSpanner(
23             ...     direction=Down,
24             ...     markup='bar',

```

```

25     ...
26     >>> attach(spanner_one, staff[:2])
27     >>> attach(spanner_two, staff[3:])
28
29     :::
30
31     >>> print(format(staff))
32     \new Staff {
33         \once \override TextSpanner.bound-details.left-broken.text = ##f
34         \once \override TextSpanner.bound-details.left.text = \markup { foo }
35         \once \override TextSpanner.bound-details.right-broken.text = ##f
36         \once \override TextSpanner.bound-details.right.text = \markup {
37             \draw-line
38             #'(0 . -1)
39         }
40         \once \override TextSpanner.dash-fraction = 1
41         \once \override TextSpanner.direction = #up
42         c'4 \startTextSpan
43         d'4
44         <> \stopTextSpan
45         r4
46         e'4 _ \markup { bar }
47     }
48
49 .. container:: example
50
51     :::
52
53     >>> import consort
54     >>> staff = Staff("c'4 d'4 e'4 f'4")
55     >>> spanner_one = consort.ComplexTextSpanner(
56     ...     direction=Up,
57     ...     markup='foo',
58     ... )
59     >>> spanner_two = consort.ComplexTextSpanner(
60     ...     direction=Down,
61     ...     markup='bar',
62     ... )
63     >>> attach(spanner_one, staff[:2])
64     >>> attach(spanner_two, staff[3:])
65
66     :::
67
68     >>> print(format(staff))
69     \new Staff {
70         \once \override TextSpanner.bound-details.left-broken.text = ##f
71         \once \override TextSpanner.bound-details.left.text = \markup { foo }
72         \once \override TextSpanner.bound-details.right-broken.text = ##f
73         \once \override TextSpanner.bound-details.right.text = \markup {
74             \draw-line
75             #'(0 . -1)
76         }
77         \once \override TextSpanner.dash-fraction = 1
78         \once \override TextSpanner.direction = #up

```

```

79      c'4 \startTextSpan
80      d'4
81      <> \stopTextSpan
82      e'4
83      f'4 _ \markup { bar }
84  }
85
86 .. container:: example
87
88 :::
89
90     >>> staff = Staff("c'8 d' e' r r a' b' c''")
91     >>> spanner_one = consort.ComplexTextSpanner(
92         ... direction=Up,
93         ... markup='foo',
94         ...
95     >>> spanner_two = consort.ComplexTextSpanner(
96         ... direction=Up,
97         ... markup='foo',
98         ...
99     >>> attach(spanner_one, staff[:3])
100    >>> attach(spanner_two, staff[5:])
101
102 :::
103
104     >>> print(format(staff))
105 \new Staff {
106     \once \override TextSpanner.bound-details.left-broken.text = ##f
107     \once \override TextSpanner.bound-details.left.text = \markup { foo }
108     \once \override TextSpanner.bound-details.right-broken.text = ##f
109     \once \override TextSpanner.bound-details.right.text = \markup {
110         \draw-line
111         #'(0 . -1)
112     }
113     \once \override TextSpanner.dash-fraction = 1
114     \once \override TextSpanner.direction = #up
115     c'8 \startTextSpan
116     d'8
117     e'8
118     r8
119     r8
120     a'8
121     b'8
122     c'8
123         <> \stopTextSpan
124     }
125
126 .. container:: example
127
128 :::
129
130     >>> staff = Staff("c'8 d' e' f' g' a' b' c''")
131     >>> spanner_one = consort.ComplexTextSpanner(
132         ... direction=Up,

```

```

133     ...     markup='foo',
134     ...     )
135     >>> spanner_two = consort.ComplexTextSpanner(
136         ...     direction=Down,
137         ...     markup='bar',
138         ...     )
139     >>> spanner_three = consort.ComplexTextSpanner(
140         ...     direction=Up,
141         ...     markup='foo',
142         ...     )
143     >>> attach(spanner_one, staff[:3])
144     >>> attach(spanner_two, staff[3:5])
145     >>> attach(spanner_three, staff[5:])
146
147 :::
148
149     >>> print(format(staff))
150     \new Staff {
151         \once \override TextSpanner.bound-details.left-broken.text = ##f
152         \once \override TextSpanner.bound-details.left.text = \markup { foo }
153         \once \override TextSpanner.bound-details.right-broken.text = ##f
154         \once \override TextSpanner.bound-details.right.text = \markup {
155             \draw-line
156                 #'(0 . -1)
157             }
158         \once \override TextSpanner.dash-fraction = 1
159         \once \override TextSpanner.direction = #up
160         c'8 \startTextSpan
161         d'8
162         e'8
163         <> \stopTextSpan
164         \once \override TextSpanner.bound-details.left-broken.text = ##f
165         \once \override TextSpanner.bound-details.left.text = \markup { bar }
166         \once \override TextSpanner.bound-details.right-broken.text = ##f
167         \once \override TextSpanner.bound-details.right.text = \markup {
168             \draw-line
169                 #'(0 . -1)
170             }
171         \once \override TextSpanner.dash-fraction = 1
172         \once \override TextSpanner.direction = #down
173         f'8 \startTextSpan
174         g'8
175         <> \stopTextSpan
176         \once \override TextSpanner.bound-details.left-broken.text = ##f
177         \once \override TextSpanner.bound-details.left.text = \markup { foo }
178         \once \override TextSpanner.bound-details.right-broken.text = ##f
179         \once \override TextSpanner.bound-details.right.text = \markup {
180             \draw-line
181                 #'(0 . -1)
182             }
183         \once \override TextSpanner.dash-fraction = 1
184         \once \override TextSpanner.direction = #up
185         a'8 \startTextSpan
186         b'8

```

```

187         c' '8
188         <> \stopTextSpan
189     }
190     '',
191
192
193     ### CLASS VARIABLES ###
194
195     __slots__ = (
196         '_direction',
197         '_markup',
198     )
199
200     ### INITIALIZER ###
201
202     def __init__(
203         self,
204         direction=None,
205         markup=None,
206         overrides=None,
207     ):
208         spannertools.Spanner.__init__(
209             self,
210             overrides=overrides,
211         )
212         assert direction in (Up, Down, None)
213         self._direction = direction
214         self._markup = markuptools.Markup(markup)
215
216     ### PRIVATE METHODS ###
217
218     def _copy_keyword_args(self, new):
219         new._markup = self._markup
220
221     def _get_lilypond_format_bundle(self, leaf):
222         lilypond_format_bundle = self._get_basic_lilypond_format_bundle(leaf)
223
224         if self._is_my_only_leaf(leaf):
225             previous_is_similar = self._previous_spacer_is_similar(leaf)
226             next_is_similar = self._next_spacer_is_similar(leaf)
227
228             if previous_is_similar and next_is_similar:
229                 pass
230
231             elif previous_is_similar:
232                 self._make_spacer_stop(lilypond_format_bundle)
233
234             elif next_is_similar:
235                 self._make_spacer_start(lilypond_format_bundle)
236
237             elif leaf.written_duration <= durationtools.Duration(1, 4):
238                 self._make_markup(lilypond_format_bundle)
239
240         else:

```

```

241         self._make_spinner_start(lilypond_format_bundle)
242         self._make_spinner_stop(lilypond_format_bundle)
243
244     elif self._is_my_first_leaf(leaf):
245         if not self._previous_spinner_is_similar(leaf):
246             self._make_spinner_start(lilypond_format_bundle)
247
248     elif self._is_my_last_leaf(leaf):
249         if not self._next_spinner_is_similar(leaf):
250             self._make_spinner_stop(lilypond_format_bundle)
251
252     return lilypond_format_bundle
253
254 def _make_markup(self, lilypond_format_bundle):
255     direction = self.direction or Up
256     markup = markuptools.Markup(
257         self.markup.contents,
258         direction,
259     )
260     lilypond_format_bundle.right.markup.append(markup)
261
262 def _make_spinner_start(self, lilypond_format_bundle):
263     override = lilypondnametools.LilyPondGrobOverride(
264         grob_name='TextSpanner',
265         is_once=True,
266         property_path=('bound-details', 'left', 'text'),
267         value=self.markup,
268     )
269     lilypond_format_bundle.update(override)
270     override = lilypondnametools.LilyPondGrobOverride(
271         grob_name='TextSpanner',
272         is_once=True,
273         property_path=('bound-details', 'left-broken', 'text'),
274         value=False,
275     )
276     lilypond_format_bundle.update(override)
277     override = lilypondnametools.LilyPondGrobOverride(
278         grob_name='TextSpanner',
279         is_once=True,
280         property_path=('bound-details', 'right', 'text'),
281         value=markuptools.Markup(r"\draw-line #'(0 . -1)")
282     )
283     lilypond_format_bundle.update(override)
284     override = lilypondnametools.LilyPondGrobOverride(
285         grob_name='TextSpanner',
286         is_once=True,
287         property_path=('bound-details', 'right-broken', 'text'),
288         value=False,
289     )
290     lilypond_format_bundle.update(override)
291     override = lilypondnametools.LilyPondGrobOverride(
292         grob_name='TextSpanner',
293         is_once=True,
294         property_path=('dash-fraction',),

```

```

295         value=1,
296     )
297     lilypond_format_bundle.update(override)
298     if self.direction is not None:
299         override = lilypondnametools.LilyPondGrobOverride(
300             grob_name='TextSpanner',
301             is_once=True,
302             property_path=(('direction',),
303             value=self.direction,
304             )
305         lilypond_format_bundle.update(override)
306         string = r'\startTextSpan'
307         lilypond_format_bundle.right.spanner_starts.append(string)
308
309     def _make_spanner_stop(self, lilypond_format_bundle):
310         string = r'\> \stopTextSpan'
311         lilypond_format_bundle.after.indicators.append(string)
312
313     # def _next_spanner_is_similar(self, leaf):
314     #     next_leaf = leaf._get_leaf(1)
315     #     next_spanner = None
316     #     next_spanner_is_similar = False
317     #     if next_leaf is not None:
318     #         spanners = next_leaf._get_spanners(type(self))
319     #         if spanners:
320     #             assert len(spanners) == 1
321     #             next_spanner = tuple(spanners)[0]
322     #             if next_spanner.direction == self.direction:
323     #                 if next_spanner.markup == self.markup:
324     #                     next_spanner_is_similar = True
325     #     return next_spanner_is_similar
326
327     def _next_spanner_is_similar(self, leaf):
328         leaf_prototype = (scoretools.Note, scoretools.Chord)
329         next_spanner = None
330         next_spanner_is_similar = False
331         for index in range(1, 5):
332             next_leaf = leaf._get_leaf(index)
333             if next_leaf is None:
334                 break
335             has_spinner = next_leaf._has_spinner(type(self))
336             if not has_spinner:
337                 if isinstance(next_leaf, leaf_prototype):
338                     break
339                 continue
340             next_spinner = next_leaf._get_spinner(type(self))
341             if next_spinner.direction != self.direction:
342                 break
343             if next_spinner.markup != self.markup:
344                 break
345             next_spinner_is_similar = True
346     return next_spinner_is_similar
347
348     def _previous_spinner_is_similar(self, leaf):

```

```

349     leaf_prototype = (scoretools.Note, scoretools.Chord)
350     previous_spacer = None
351     previous_spacer_is_similar = False
352     for index in range(1, 5):
353         previous_leaf = leaf._get_leaf(-index)
354         if previous_leaf is None:
355             break
356         has_spacer = previous_leaf._has_spacer(type(self))
357         if not has_spacer:
358             if isinstance(previous_leaf, leaf_prototype):
359                 break
360             continue
361         previous_spacer = previous_leaf._get_spacer(type(self))
362         if previous_spacer.direction != self.direction:
363             break
364         if previous_spacer.markup != self.markup:
365             break
366         previous_spacer_is_similar = True
367     return previous_spacer_is_similar
368
369 #     def _previous_spacer_is_similar(self, leaf):
370 #         previous_leaf = leaf._get_leaf(-1)
371 #         previous_spacer = None
372 #         previous_spacer_is_similar = False
373 #         if previous_leaf is not None:
374 #             spacers = previous_leaf._get_spacers(type(self))
375 #             if spacers:
376 #                 assert len(spacers) == 1
377 #                 previous_spacer = tuple(spacers)[0]
378 #                 if previous_spacer.direction == self.direction:
379 #                     if previous_spacer.markup == self.markup:
380 #                         previous_spacer_is_similar = True
381 #         return previous_spacer_is_similar
382
383     ### PUBLIC PROPERTIES ###
384
385     @property
386     def direction(self):
387         return self._direction
388
389     @property
390     def markup(self):
391         return self._markup

```

## A.14 CONSORT.TOOLS.COMPOSITEMUSICSPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad.tools import datastructuretools
4 from abjad.tools import sequencetools
5 from abjad.tools import timespantools
6 from consort.tools.HashCachingObject import HashCachingObject
7
8

```

```

9 class CompositeMusicSpecifier(HashCachingObject):
10     r'''A composite music specifier.
11
12     :::
13
14     >>> import consort
15     >>> musicSpecifier = consort.CompositeMusicSpecifier(
16         ...     primary_musicSpecifier='one',
17         ...     primary_voice_name='Viola 1 RH',
18         ...     rotation_indices=(0, 1, -1),
19         ...     secondary_voice_name='Viola 1 LH',
20         ...     secondary_musicSpecifier=consort.MusicSpecifierSequence(
21             ...         application_rate='phrase',
22             ...         music_specifiers=['two', 'three', 'four'],
23             ...         ),
24         ...     )
25     >>> print(format(musicSpecifier))
26     consort.tools.CompositeMusicSpecifier(
27         primary_musicSpecifier=consort.tools.MusicSpecifierSequence(
28             music_specifiers=datastructuretools.CyclicTuple(
29                 ['one']
30             ),
31             ),
32         primary_voice_name='Viola 1 RH',
33         rotation_indices=(0, 1, -1),
34         secondary_musicSpecifier=consort.tools.MusicSpecifierSequence(
35             application_rate='phrase',
36             music_specifiers=datastructuretools.CyclicTuple(
37                 ['two', 'three', 'four']
38             ),
39             ),
40         secondary_voice_name='Viola 1 LH',
41     )
42
43     :::
44
45     >>> durations = [1, 2]
46     >>> timespans = musicSpecifier(
47         ...     durations=durations,
48         ...     layer=1,
49         ...     )
50     >>> print(format(timespans))
51     timespanTools.TimespanInventory(
52         [
53             consort.tools.PerformedTimespan(
54                 start_offset=durationtools.Offset(0, 1),
55                 stop_offset=durationtools.Offset(1, 1),
56                 layer=1,
57                 musicSpecifier='two',
58                 voice_name='Viola 1 LH',
59                 ),
60             consort.tools.PerformedTimespan(
61                 start_offset=durationtools.Offset(0, 1),
62                 stop_offset=durationtools.Offset(1, 1),

```

```

63         layer=1,
64         musicSpecifier='one',
65         voiceName='Viola 1 RH',
66         ),
67     consort.tools.PerformedTimespan(
68         startOffset=durationtools.Offset(1, 1),
69         stopOffset=durationtools.Offset(3, 1),
70         layer=1,
71         musicSpecifier='two',
72         voiceName='Viola 1 LH',
73         ),
74     consort.tools.PerformedTimespan(
75         startOffset=durationtools.Offset(1, 1),
76         stopOffset=durationtools.Offset(3, 1),
77         layer=1,
78         musicSpecifier='one',
79         voiceName='Viola 1 RH',
80         ),
81     ],
82 )
83 :::
84
85
86 >>> durations = [1, 2]
87 >>> timespans = musicSpecifier(
88 ...     durations=durations,
89 ...     layer=2,
90 ...     seed=1,
91 ...     )
92 >>> print(format(timespans))
93 timespantools.TimespanInventory(
94 [
95     consort.tools.PerformedTimespan(
96         startOffset=durationtools.Offset(0, 1),
97         stopOffset=durationtools.Offset(1, 1),
98         layer=2,
99         musicSpecifier='one',
100        voiceName='Viola 1 RH',
101        ),
102     consort.tools.PerformedTimespan(
103         startOffset=durationtools.Offset(0, 1),
104         stopOffset=durationtools.Offset(2, 1),
105         layer=2,
106         musicSpecifier='three',
107         voiceName='Viola 1 LH',
108         ),
109     consort.tools.PerformedTimespan(
110         startOffset=durationtools.Offset(1, 1),
111         stopOffset=durationtools.Offset(3, 1),
112         layer=2,
113         musicSpecifier='one',
114         voiceName='Viola 1 RH',
115         ),
116     consort.tools.PerformedTimespan(

```

```

117     start_offset=durationtools.Offset(2, 1),
118     stop_offset=durationtools.Offset(3, 1),
119     layer=2,
120     musicSpecifier='three',
121     voice_name='Viola 1 LH',
122     ),
123   ],
124 )
125
126 :::
127
128 >>> durations = [1, 2]
129 >>> timespans = musicSpecifier(
130   ...   durations=durations,
131   ...   layer=3,
132   ...   padding=1,
133   ...   seed=2,
134   ...   )
135 >>> print(format(timespans))
136 timespantools.TimespanInventory(
137 [
138   consort.tools.SilentTimespan(
139     start_offset=durationtools.Offset(-1, 1),
140     stop_offset=durationtools.Offset(0, 1),
141     layer=3,
142     voice_name='Viola 1 RH',
143     ),
144   consort.tools.SilentTimespan(
145     start_offset=durationtools.Offset(-1, 1),
146     stop_offset=durationtools.Offset(0, 1),
147     layer=3,
148     voice_name='Viola 1 LH',
149     ),
150   consort.tools.PerformedTimespan(
151     start_offset=durationtools.Offset(0, 1),
152     stop_offset=durationtools.Offset(1, 1),
153     layer=3,
154     musicSpecifier='one',
155     voice_name='Viola 1 RH',
156     ),
157   consort.tools.PerformedTimespan(
158     start_offset=durationtools.Offset(0, 1),
159     stop_offset=durationtools.Offset(2, 1),
160     layer=3,
161     musicSpecifier='four',
162     voice_name='Viola 1 LH',
163     ),
164   consort.tools.PerformedTimespan(
165     start_offset=durationtools.Offset(1, 1),
166     stop_offset=durationtools.Offset(3, 1),
167     layer=3,
168     musicSpecifier='one',
169     voice_name='Viola 1 RH',
170     ),

```

```

171     consort.tools.PerformedTimespan(
172         start_offset=durationtools.Offset(2, 1),
173         stop_offset=durationtools.Offset(3, 1),
174         layer=3,
175         music_specifier='four',
176         voice_name='Viola 1 LH',
177         ),
178     consort.tools.SilentTimespan(
179         start_offset=durationtools.Offset(3, 1),
180         stop_offset=durationtools.Offset(4, 1),
181         layer=3,
182         voice_name='Viola 1 RH',
183         ),
184     consort.tools.SilentTimespan(
185         start_offset=durationtools.Offset(3, 1),
186         stop_offset=durationtools.Offset(4, 1),
187         layer=3,
188         voice_name='Viola 1 LH',
189         ),
190     ],
191 )
192 ...
193
194
195     """ CLASS VARIABLES """
196
197     __slots__ = (
198         '_discard_inner_offsets',
199         '_primary_musicSpecifier',
200         '_primary_voice_name',
201         '_rotation_indices',
202         '_secondary_musicSpecifier',
203         '_secondary_voice_name',
204     )
205
206     """ INITIALIZER """
207
208     def __init__(
209         self,
210         discard_inner_offsets=None,
211         primary_musicSpecifier=None,
212         primary_voice_name=None,
213         rotation_indices=None,
214         secondary_musicSpecifier=None,
215         secondary_voice_name=None,
216     ):
217         import consort
218         HashCachingObject.__init__(self)
219         prototype = consort.MusicSpecifierSequence
220         if discard_inner_offsets is not None:
221             discard_inner_offsets = bool(discard_inner_offsets)
222         self._discard_inner_offsets = discard_inner_offsets
223         if not isinstance(primary_musicSpecifier, prototype):
224             primary_musicSpecifier = consort.MusicSpecifierSequence(

```

```

225         music_specifiers=primary_musicSpecifier,
226         )
227     self._primary_musicSpecifier = primary_musicSpecifier
228     if primary_voice_name is not None:
229         primary_voice_name = str(primary_voice_name)
230     self._primary_voice_name = primary_voice_name
231     if rotation_indices is not None:
232         if not isinstance(rotation_indices, collections.Sequence):
233             rotation_indices = int(rotation_indices)
234             rotation_indices = (rotation_indices,)
235             rotation_indices = tuple(rotation_indices)
236     self._rotation_indices = rotation_indices
237     if not isinstance(secondary_musicSpecifier, prototype):
238         secondary_musicSpecifier = consort.MusicSpecifierSequence(
239             music_specifiers=secondary_musicSpecifier,
240             )
241     self._secondary_musicSpecifier = secondary_musicSpecifier
242     if secondary_voice_name is not None:
243         secondary_voice_name = str(secondary_voice_name)
244     self._secondary_voice_name = secondary_voice_name
245
246     ### PUBLIC METHODS ###
247
248     def __call__(
249         self,
250         durations=None,
251         layer=None,
252         output_masks=None,
253         padding=None,
254         seed=None,
255         start_offset=None,
256         timespanSpecifier=None,
257         voice_name=None,
258         ):
259         seed = seed or 0
260         rotation_indices = self.rotation_indices or (0,)
261         rotation_indices = datastructuretools.CyclicTuple(rotation_indices)
262         primary_durations = durations
263         start_offset = start_offset or 0
264         if self.discard_inner_offsets:
265             secondary_durations = [sum(primary_durations)]
266         else:
267             secondary_durations = sequencetools.rotate_sequence(
268                 primary_durations,
269                 rotation_indices[seed],
270                 )
271         primary_timespans = self.primary_musicSpecifier(
272             durations=primary_durations,
273             layer=layer,
274             output_masks=output_masks,
275             padding=padding,
276             seed=seed,
277             start_offset=start_offset,
278             timespanSpecifier=timespanSpecifier,

```

```

279         voice_name=self.primary_voice_name,
280     )
281     secondary_timespans = self.secondary_music_specifier(
282         durations=secondary_durations,
283         layer=layer,
284         output_masks=output_masks,
285         padding=padding,
286         seed=seed,
287         start_offset=start_offset,
288         timespanSpecifier=timespanSpecifier,
289         voice_name=self.secondary_voice_name,
290     )
291     timespans = primary_timespans[:] + secondary_timespans[:]
292     timespans = timespantools.TimespanInventory(timespans)
293     timespans.sort()
294     return timespans
295
296     """ PUBLIC PROPERTIES """
297
298     @property
299     def discard_inner_offsets(self):
300         return self._discard_inner_offsets
301
302     @property
303     def primary_music_specifier(self):
304         return self._primary_music_specifier
305
306     @property
307     def primary_voice_name(self):
308         return self._primary_voice_name
309
310     @property
311     def rotation_indices(self):
312         return self._rotation_indices
313
314     @property
315     def secondary_music_specifier(self):
316         return self._secondary_music_specifier
317
318     @property
319     def secondary_voice_name(self):
320         return self._secondary_voice_name

```

## A.15 CONSORTTOOLS.COMPOSITERHYTHMMAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import abctools
4 from abjad.tools import durationtools
5 from abjad.tools import rhythmmakertools
6
7
8 class CompositeRhythmMaker(abctools.AbjadValueObject):
9     r'''A composite rhythm-maker.

```

```

10
11 :::
12
13     >>> import consort
14     >>> composite_rhythm_maker = consort.CompositeRhythmMaker(
15         ...     default=rhythmmakertools.EvenDivisionRhythmMaker(),
16         ...     first=rhythmmakertools.NoteRhythmMaker(),
17         ...     last=rhythmmakertools.IncisedRhythmMaker(
18             ...         inciseSpecifier=rhythmmakertools.InciseSpecifier(
19                 ...             prefixCounts=[0],
20                 ...             suffixTalea=[1],
21                 ...             suffixCounts=[1],
22                 ...             taleaDenominator=16,
23                 ...             ),
24                 ...             ),
25             ...         only=rhythmmakertools.EvenDivisionRhythmMaker(
26                 ...             denominators=[32],
27                 ...             ),
28             ...         )
29     >>> print(format(composite_rhythm_maker))
30     consort.tools.CompositeRhythmMaker(
31         default=rhythmmakertools.EvenDivisionRhythmMaker(
32             denominators=(8,),
33             preferredDenominator='from_counts',
34             ),
35             first=rhythmmakertools.NoteRhythmMaker(),
36             last=rhythmmakertools.IncisedRhythmMaker(
37                 inciseSpecifier=rhythmmakertools.InciseSpecifier(
38                     prefixCounts=(0,),
39                     suffixTalea=(1,),
40                     suffixCounts=(1,),
41                     taleaDenominator=16,
42                     ),
43                     ),
44             only=rhythmmakertools.EvenDivisionRhythmMaker(
45                 denominators=(32,),
46                 preferredDenominator='from_counts',
47                 ),
48             )
49
50 .. container:: example
51
52 :::
53
54     >>> divisions = [(1, 4), (1, 4), (1, 4), (1, 4)]
55     >>> result = composite_rhythm_maker(divisions)
56     >>> staff = Staff()
57     >>> for x in result:
58         ...     staff.extend(x)
59         ...
60     >>> print(format(staff))
61     \new Staff {
62         c'4
63         {

```

```

64          c'8 [
65          c'8 ]
66      }
67      {
68          c'8 [
69          c'8 ]
70      }
71      c'8. [
72          c'16 ]
73  }
74
75 .. container:: example
76
77 :::
78
79     >>> divisions = [(1, 4), (1, 4)]
80     >>> result = composite_rhythm_maker(divisions)
81     >>> staff = Staff()
82     >>> for x in result:
83         ...     staff.extend(x)
84         ...
85     >>> print(format(staff))
86     \new Staff {
87         c'4
88         c'8. [
89             c'16 ]
90     }
91
92 .. container:: example
93
94 :::
95
96     >>> divisions = [(1, 4)]
97     >>> result = composite_rhythm_maker(divisions)
98     >>> staff = Staff()
99     >>> for x in result:
100        ...     staff.extend(x)
101        ...
102     >>> print(format(staff))
103     \new Staff {
104     {
105         c'32 [
106         c'32
107         c'32
108         c'32
109         c'32
110         c'32
111         c'32
112         c'32 ]
113     }
114 }
115
116 ...
117

```

```

118     """ CLASS VARIABLES """
119
120     __slots__ = (
121         '_default',
122         '_first',
123         '_last',
124         '_only',
125     )
126
127     """ INITIALIZER """
128
129     def __init__(
130         self,
131         default=None,
132         first=None,
133         last=None,
134         only=None,
135     ):
136         if first is not None:
137             assert isinstance(first, rhythmtools.RhythmMaker)
138         if last is not None:
139             assert isinstance(last, rhythmtools.RhythmMaker)
140         if only is not None:
141             assert isinstance(only, rhythmtools.RhythmMaker)
142         if default is None:
143             default = rhythmtools.NoteRhythmMaker()
144         assert isinstance(default, rhythmtools.RhythmMaker)
145         self._first = first
146         self._last = last
147         self._only = only
148         self._default = default
149
150     """ SPECIAL METHODS """
151
152     def __call__(self, divisions, rotation=None):
153         divisions = [durationtools.Division(x) for x in divisions]
154         result = []
155         if not divisions:
156             pass
157         elif len(divisions) == 1:
158             if self.only:
159                 result.extend(self.only(divisions, rotation=rotation))
160             elif self.last:
161                 result.extend(self.last(divisions, rotation=rotation))
162             elif self.first:
163                 result.extend(self.first(divisions, rotation=rotation))
164             else:
165                 result.extend(self.default(divisions, rotation=rotation))
166         elif len(divisions) == 2:
167             if self.first and self.last:
168                 first = self.first(divisions=[divisions[0]], rotation=rotation)
169                 last = self.last(divisions=[divisions[1]], rotation=rotation)
170                 result.extend(first)
171                 result.extend(last)

```

```

172     elif self.first:
173         first = self.first(divisions=[divisions[0]], rotation=rotation)
174         default = self.default(divisions=[divisions[1]], rotation=rotation)
175         result.extend(first)
176         result.extend(default)
177     elif self.last:
178         default = self.default(divisions=[divisions[0]], rotation=rotation)
179         last = self.last(divisions=[divisions[1]], rotation=rotation)
180         result.extend(default)
181         result.extend(last)
182     else:
183         default = self.default(divisions=divisions, rotation=rotation)
184         result.extend(default)
185     else:
186         if self.first and self.last:
187             first = self.first(divisions=[divisions[0]], rotation=rotation)
188             default = self.default(divisions=divisions[1:-1], rotation=rotation)
189             last = self.last(divisions=[divisions[-1]], rotation=rotation)
190             result.extend(first)
191             result.extend(default)
192             result.extend(last)
193         elif self.first:
194             first = self.first(divisions=[divisions[0]], rotation=rotation)
195             default = self.default(divisions=divisions[1:], rotation=rotation)
196             result.extend(first)
197             result.extend(default)
198         elif self.last:
199             default = self.default(divisions=divisions[:-1], rotation=rotation)
200             last = self.last(divisions=[divisions[-1]], rotation=rotation)
201             result.extend(default)
202             result.extend(last)
203     else:
204         default = self.default(divisions=divisions, rotation=rotation)
205         result.extend(default)
206     return result
207
208 def __illustrate__(self, divisions=None):
209     r'''Illustrates composite rhythm-maker.
210
211     Returns LilyPond file.
212     '''
213     divisions = divisions or [
214         (3, 8),
215         (4, 8),
216         (3, 16),
217         (4, 16),
218         (5, 8),
219         (2, 4),
220         (5, 16),
221         (2, 8),
222         (7, 8),
223     ]
224     selections = self(divisions)
225     lilypond_file = rhythmmakertools.make_lilypond_file(

```

```

226         selections,
227         divisions,
228     )
229     return lilypond_file
230
231     ### PUBLIC METHODS ###
232
233     def new(
234         self,
235         first=None,
236         last=None,
237         only=None,
238         default=None,
239         **kwargs
240     ):
241         first = first or self.first
242         last = last or self.last
243         only = only or self.only
244         default = default or self.default
245         if first is not None:
246             first = new(first, **kwargs)
247         if last is not None:
248             last = new(last, **kwargs)
249         if only is not None:
250             only = new(only, **kwargs)
251         if default is not None:
252             default = new(default, **kwargs)
253         result = new(
254             self,
255             first=first,
256             last=last,
257             only=only,
258             default=default,
259         )
260         return result
261
262     ### PUBLIC PROPERTIES ###
263
264     @property
265     def first(self):
266         return self._first
267
268     @property
269     def last(self):
270         return self._last
271
272     @property
273     def only(self):
274         return self._only
275
276     @property
277     def default(self):
278         return self._default

```

## A.16 CONSORTTOOLS.CONSORTTRILLSPANNER

```
 1 # -*- encoding: utf-8 -*-
 2 from abjad import inspect_
 3 from abjad.tools import lilypondnametools
 4 from abjad.tools import pitchtools
 5 from abjad.tools import spannertools
 6
 7
 8 class ConsortTrillSpanner(spannertools.Spanner):
 9     r'''A complex trill spanner.
10
11     .. container:: example
12
13         :::
14
15         >>> staff = Staff("c'4 ~ c'8 d'8 r8 e'8 ~ e'8 r8")
16         >>> show(staff) # doctest: +SKIP
17
18     .. doctest::
19
20         >>> print(format(staff))
21         \new Staff {
22             c'4 ~
23             c'8
24             d'8
25             r8
26             e'8 ~
27             e'8
28             r8
29         }
30
31         :::
32
33         >>> import consort
34         >>> complex_trill = consort.ConsortTrillSpanner(
35             ...     interval='P4',
36             ... )
37         >>> attach(complex_trill, staff.select_leaves())
38         >>> show(staff) # doctest: +SKIP
39
40     .. doctest::
41
42         >>> print(format(staff))
43         \new Staff {
44             \pitchedTrill
45             c'4 ~ \startTrillSpan f'
46             c'8
47             <> \stopTrillSpan
48             \pitchedTrill
49             d'8 \startTrillSpan g'
50             <> \stopTrillSpan
51             r8
52             \pitchedTrill
```

```

53         e'8 ~ \startTrillSpan a'
54         e'8
55         <> \stopTrillSpan
56         r8
57     }
58
59     Allows for specifying a trill pitch via a named interval.
60
61     Avoids silences.
62
63     Restarts the trill on every new pitched logical tie.
64     ''
65
66     ### CLASS VARIABLES ###
67
68     __slots__ = (
69         '_interval',
70     )
71
72     ### INITIALIZER ###
73
74     def __init__(
75         self,
76         overrides=None,
77         interval=None,
78     ):
79         spannertools.Spanner.__init__(
80             self,
81             overrides=overrides,
82         )
83         if interval is not None:
84             interval = pitchtools.NamedInterval(interval)
85         self._interval = interval
86
87     ### PRIVATE METHODS ###
88
89     def _copy_keyword_args(self, new):
90         new._interval = self.interval
91
92     def _get_lilypond_format_bundle(self, leaf):
93         from abjad.tools import scoretools
94         lilypond_format_bundle = self._get_basic_lilypond_format_bundle(leaf)
95         prototype = (
96             scoretools.Rest,
97             scoretools.MultimeasureRest,
98             scoretools.Skip,
99         )
100        if isinstance(leaf, prototype):
101            return lilypond_format_bundle
102        logical_tie = inspect_(leaf).get_logical_tie()
103
104        starts_spinner, stops_spinner = False, False
105        if leaf is logical_tie.head:
106            starts_spinner = True

```

```

107     after_graces = inspect_(leaf).get_grace_containers('after')
108     if leaf is logical_tie.tail and not len(after_graces):
109         stops_spinner = True
110     elif self._is_my_last_leaf(leaf):
111         stops_spinner = True
112
113     if starts_spinner:
114         previous_leaf = leaf._get_leaf(-1)
115         if previous_leaf is not None:
116             after_graces = inspect_(previous_leaf).get_grace_containers(
117                 'after')
118             if after_graces:
119                 grob_override = lilypondnametools.LilyPondGrobOverride(
120                     grob_name='TrillSpanner',
121                     is_once=True,
122                     property_path=(
123                         'bound-details',
124                         'left',
125                         'padding',
126                         ),
127                         value=2,
128                         )
129                 string = '\n'.join(grob_override._override_format_pieces)
130                 lilypond_format_bundle.grob_overrides.append(string)
131             if self.interval is not None:
132                 string = r'\pitchedTrill'
133                 lilypond_format_bundle.opening.spanners.append(string)
134                 if hasattr(leaf, 'written_pitch'):
135                     written_pitch = leaf.written_pitch
136                 elif hasattr(leaf, 'written_pitches'):
137                     if 0 < self.interval.semitones:
138                         written_pitch = max(leaf.written_pitches)
139                     elif self.interval.semitones < 0:
140                         written_pitch = min(leaf.written_pitches)
141                     trill_pitch = written_pitch.transpose(self.interval)
142                     string = r'\startTrillSpan {!s}'.format(trill_pitch)
143                 else:
144                     string = r'\startTrillSpan'
145                 lilypond_format_bundle.right.trill_pitches.append(string)
146
147             if stops_spinner:
148                 next_leaf = leaf._get_leaf(1)
149                 if next_leaf is not None:
150                     string = r'>\stopTrillSpan'
151                     lilypond_format_bundle.after.commands.append(string)
152                 else:
153                     string = r'\stopTrillSpan'
154                     lilypond_format_bundle.right.spanner_stops.append(string)
155
156     return lilypond_format_bundle
157
158     ### PUBLIC PROPERTIES ###
159
160     @property

```

```

161     def interval(self):
162         r'''Gets optional interval of trill spanner.
163
164         .. container:: example
165
166             :::
167
168             >>> import consort
169             >>> staff = Staff("c'4 d'4 e'4 f'4")
170             >>> interval = pitchtools.NamedInterval('m3')
171             >>> complex_trill = consort.ConsortTrillSpanner(
172                 ...     interval=interval)
173             >>> attach(complex_trill, staff[1:-1])
174             >>> show(staff) # doctest: +SKIP
175
176         .. doctest::
177
178             >>> print(format(staff))
179             \new Staff {
180                 c'4
181                 \pitchedTrill
182                 d'4 \startTrillSpan f'
183                 <> \stopTrillSpan
184                 \pitchedTrill
185                 e'4 \startTrillSpan g'
186                 <> \stopTrillSpan
187                 f'4
188             }
189
190             :::
191
192             >>> complex_trill.interval
193             NamedInterval('+m3')
194
195             ...
196
197             return self._interval

```

## A.17 CONSORTTOOLS.CURSOR

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3 from abjad.tools import datastructuretools
4
5
6 class Cursor(abctools.AbjadValueObject):
7     r'''A cursor.
8
9     .. container:: example
10
11         :::
12
13         >>> import consort
14         >>> cursor = consort.Cursor([1, 2, 3])
15         >>> next(cursor)

```

```

16      1
17
18      :::
19
20      >>> next(cursor)
21      2
22
23      :::
24
25      >>> next(cursor)
26      3
27
28      :::
29
30      >>> next(cursor)
31      1
32
33      :::
34
35      >>> next(cursor)
36      2
37
38      :::
39
40      >>> cursor.backtrack()
41      2
42
43      :::
44
45      >>> cursor.backtrack()
46      1
47
48      :::
49
50      >>> cursor.backtrack()
51      3
52
53      :::
54
55      >>> next(cursor)
56      3
57
58      :::
59
60      >>> next(cursor)
61      1
62
63  .. container:: example
64
65      :::
66
67      >>> talea = rhythmmakertools.Talea(
68      ...     counts=(2, 1, 3, 2, 4, 1, 1),
69      ...     denominator=16,

```

```

70         ...
71     >>> cursor = consort.Cursor(talea)
72     >>> for _ in range(10):
73         ...     next(cursor)
74         ...
75         Duration(1, 8)
76         Duration(1, 16)
77         Duration(3, 16)
78         Duration(1, 8)
79         Duration(1, 4)
80         Duration(1, 16)
81         Duration(1, 16)
82         Duration(1, 8)
83         Duration(1, 16)
84         Duration(3, 16)
85
86     '',
87
88     ### CLASS VARIABLES ###
89
90     __slots__ = (
91         '_sequence',
92         '_index',
93     )
94
95     ### INITIALIZER ###
96
97     def __init__(self, sequence=(1, 2, 3), index=None):
98         self._sequence = datastructuretools.CyclicTuple(sequence)
99         if index is not None:
100             index = int(index)
101         self._index = index
102
103     ### SPECIAL METHODS ###
104
105     def __iter__(self):
106         while True:
107             yield self.next()
108
109     def __next__(self):
110         return self.next()
111
112     ### PUBLIC METHODS ###
113
114     def backtrack(self):
115         if not self._sequence:
116             return
117         if self._index is None:
118             self._index = 0
119         self._index -= 1
120         index = self._index
121         return self._sequence[index]
122
123     def next(self):

```

```

124     if not self._sequence:
125         return
126     if self._index is None:
127         self._index = 1
128         return self._sequence[0]
129     index = self._index
130     self._index += 1
131     return self._sequence[index]
132
133     ### PUBLIC PROPERTIES ###
134
135     @property
136     def index(self):
137         return self._index
138
139     @property
140     def sequence(self):
141         return self._sequence

```

## A.18 CONSORT.TOOLS.DEPENDENTTIMESPANMAKER

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad import inspect_
4 from abjad.tools import datastructuretools
5 from abjad.tools import durationtools
6 from abjad.tools import mathtools
7 from abjad.tools import sequencetools
8 from abjad.tools import timespantools
9 from consort.tools.TimespanMaker import TimespanMaker
10
11
12 class DependentTimespanMaker(TimespanMaker):
13     r'''A dependent timespan-maker.
14
15     :::
16
17     >>> import consort
18     >>> timespan_maker = consort.DependentTimespanMaker(
19     ...     include_inner_starts=True,
20     ...     include_inner_stops=True,
21     ...     voice_names=(
22     ...         'Viola Voice',
23     ...         ),
24     ...     )
25     >>> print(format(timespan_maker))
26     consort.tools.DependentTimespanMaker(
27         include_inner_starts=True,
28         include_inner_stops=True,
29         voice_names=('Viola Voice',),
30         )
31
32     :::
33

```

```

34     >>> timespan_inventory = timespantools.TimespanInventory([
35         ...     consort.tools.PerformedTimespan(
36             ...         voice_name='Viola Voice',
37             ...         start_offset=(1, 4),
38             ...         stop_offset=(1, 1),
39             ...         ),
40         ...     consort.tools.PerformedTimespan(
41             ...         voice_name='Viola Voice',
42             ...         start_offset=(3, 4),
43             ...         stop_offset=(3, 2),
44             ...         ),
45         ...     ])
46
47 :::
48
49     >>> music_specifiers = {
50         ...     'Violin Voice': None,
51         ...     'Cello Voice': None,
52         ...     }
53     >>> target_timespan = timespantools.Timespan((1, 2), (2, 1))
54     >>> timespan_inventory = timespan_maker(
55         ...     music_specifiers=music_specifiers,
56         ...     target_timespan=target_timespan,
57         ...     timespan_inventory=timespan_inventory,
58         ...     )
59     >>> print(format(timespan_inventory))
60 timespantools.TimespanInventory(
61     [
62         consort.tools.PerformedTimespan(
63             start_offset=durationtools.Offset(1, 4),
64             stop_offset=durationtools.Offset(1, 1),
65             voice_name='Viola Voice',
66             ),
67         consort.tools.PerformedTimespan(
68             start_offset=durationtools.Offset(1, 2),
69             stop_offset=durationtools.Offset(3, 4),
70             voice_name='Cello Voice',
71             ),
72         consort.tools.PerformedTimespan(
73             start_offset=durationtools.Offset(1, 2),
74             stop_offset=durationtools.Offset(3, 4),
75             voice_name='Violin Voice',
76             ),
77         consort.tools.PerformedTimespan(
78             start_offset=durationtools.Offset(3, 4),
79             stop_offset=durationtools.Offset(1, 1),
80             voice_name='Cello Voice',
81             ),
82         consort.tools.PerformedTimespan(
83             start_offset=durationtools.Offset(3, 4),
84             stop_offset=durationtools.Offset(1, 1),
85             voice_name='Violin Voice',
86             ),
87         consort.tools.PerformedTimespan(

```

```

88             start_offset=durationtools.Offset(3, 4),
89             stop_offset=durationtools.Offset(3, 2),
90             voice_name='Viola Voice',
91             ),
92             consort.tools.PerformedTimespan(
93                 start_offset=durationtools.Offset(1, 1),
94                 stop_offset=durationtools.Offset(3, 2),
95                 voice_name='Cello Voice',
96                 ),
97             consort.tools.PerformedTimespan(
98                 start_offset=durationtools.Offset(1, 1),
99                 stop_offset=durationtools.Offset(3, 2),
100                voice_name='Violin Voice',
101                ),
102            ],
103        )
104
105    """
106
107    ### CLASS VARIABLES ###
108
109    __slots__ = (
110        '_hysteresis',
111        '_include_inner_starts',
112        '_include_inner_stops',
113        '_inspect_music',
114        '_labels',
115        '_rotation_indices',
116        '_voice_names',
117    )
118
119    ### INITIALIZER ###
120
121    def __init__(
122        self,
123        hysteresis=None,
124        include_inner_starts=None,
125        include_inner_stops=None,
126        inspect_music=None,
127        labels=None,
128        output_masks=None,
129        padding=None,
130        rotation_indices=None,
131        seed=None,
132        timespanSpecifier=None,
133        voiceNames=None,
134    ):
135        TimespanMaker.__init__(
136            self,
137            output_masks=output_masks,
138            padding=padding,
139            seed=seed,
140            timespanSpecifier=timespanSpecifier,
141        )

```

```

142     if hysteresis is not None:
143         hysteresis = durationtools.Duration(hysteresis)
144         assert 0 < hysteresis
145     self._hysteresis = hysteresis
146     if include_inner_starts is not None:
147         include_inner_starts = bool(include_inner_starts)
148     self._include_inner_starts = include_inner_starts
149     if include_inner_stops is not None:
150         include_inner_stops = bool(include_inner_stops)
151     self._include_inner_stops = include_inner_stops
152     if inspect_music is not None:
153         inspect_music = bool(inspect_music)
154     self._inspect_music = inspect_music
155     if rotation_indices is not None:
156         if not isinstance(rotation_indices, collections.Sequence):
157             rotation_indices = int(rotation_indices)
158             rotation_indices = (rotation_indices,)
159             rotation_indices = tuple(rotation_indices)
160         self._rotation_indices = rotation_indices
161     if labels is not None:
162         if isinstance(labels, str):
163             labels = (labels,)
164             labels = tuple(str(_) for _ in labels)
165         self._labels = labels
166     if voice_names is not None:
167         voice_names = tuple(voice_names)
168     self._voice_names = voice_names
169
170     ### PRIVATE METHODS ###
171
172     def _collect_preexisting_time spans(
173         self,
174         target_time span=None,
175         time span_inventory=None,
176     ):
177         import consort
178         preexisting_time spans = timespan tools.TimespanInventory()
179         for time span in time span_inventory:
180             if not isinstance(time span, consort.PerformedTime span):
181                 continue
182             if (
183                 self.voice_names and
184                 time span.voice_name not in self.voice_names
185             ):
186                 continue
187             if not self.labels:
188                 pass
189             elif not hasattr(time span, 'music specifier') or \
190                 not time span.music specifier or \
191                 not time span.music specifier.labels:
192                 continue
193             elif not any(label in time span.music specifier.labels
194                 for label in self.labels):
195                 continue

```

```

196     preexisting_timespans.append(timespan)
197     if self.inspect_music and timespan.music:
198         outer_start_offset = timespan.start_offset
199         inner_start_offset = \
200             inspect_(timespan.music).get_timespan().start_offset
201         assert inner_start_offset == 0
202         for division in timespan.music:
203             division_timespan = inspect_(division).get_timespan()
204             division_timespan = division_timespan.translate(
205                 outer_start_offset)
206             preexisting_timespans.append(division_timespan)
207     preexisting_timespans & target_timespan
208     return preexisting_timespans
209
210 def _partition_preexisting_timespans(self, timespans):
211     shards = timespans.partition(include_tangent_timespans=True)
212     if not self.hysteresis or not shards:
213         return shards
214     coalesced_shards = [shards[0]]
215     for shard in shards[1:]:
216         last_stop = coalesced_shards[-1].stop_offset
217         this_start = shard.start_offset
218         gap = this_start - last_stop
219         if self.hysteresis <= gap:
220             coalesced_shards.append(shard)
221         else:
222             coalesced_shards[-1].extend(shard)
223             coalesced_shards[-1].sort()
224     return coalesced_shards
225
226 def _make_timespans(
227     self,
228     layer=None,
229     music_specifiers=None,
230     target_timespan=None,
231     timespan_inventory=None,
232 ):
233     new_timespans = timespantools.TimespanInventory()
234     if not self.voice_names and not self.labels:
235         return new_timespans
236     rotation_indices = self.rotation_indices or (0,)
237     rotation_indices = datastructuretools.CyclicTuple(rotation_indices)
238     context_counter = collections.Counter()
239     preexisting_timespans = self._collect_preexisting_timespans(
240         target_timespan=target_timespan,
241         timespan_inventory=timespan_inventory,
242     )
243     partitioned_timespans = self._partition_preexisting_timespans(
244         preexisting_timespans)
245     for group_index, group in enumerate(partitioned_timespans):
246         rotation_index = rotation_indices[group_index]
247         offsets = set()
248         offsets.add(group.start_offset)
249         offsets.add(group.stop_offset)

```

```

250     for timespan in group:
251         if self.include_inner_starts:
252             offsets.add(timespan.start_offset)
253         if self.include_inner_stops:
254             offsets.add(timespan.stop_offset)
255     offsets = tuple(sorted(offsets))
256     durations = mathtools.difference_series(offsets)
257     durations = sequencetools.rotate_sequence(
258         durations, rotation_index)
259     start_offset = offsets[0]
260     for context_name, music_specifier in music_specifiers.items():
261         context_seed = context_counter[context_name]
262         timespans = music_specifier(
263             durations=durations,
264             layer=layer,
265             output_masks=self.output_masks,
266             padding=self.padding,
267             seed=context_seed,
268             start_offset=start_offset,
269             timespanSpecifier=self.timespanSpecifier,
270             voice_name=context_name,
271         )
272         context_counter[context_name] += 1
273         new_timespans.extend(timespans)
274     return new_timespans
275
276     ### PUBLIC PROPERTIES ###
277
278     @property
279     def hysteresis(self):
280         return self._hysteresis
281
282     @property
283     def include_inner_starts(self):
284         return self._include_inner_starts
285
286     @property
287     def include_inner_stops(self):
288         return self._include_inner_stops
289
290     @property
291     def inspect_music(self):
292         return self._inspect_music
293
294     @property
295     def is_dependent(self):
296         return True
297
298     @property
299     def labels(self):
300         return self._labels
301
302     @property
303     def rotation_indices(self):

```

```

304     return self._rotation_indices
305
306     @property
307     def voice_names(self):
308         return self._voice_names

```

## A.19 CONSORTTOOLS.DYNAMICEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import attach
4 from abjad import inspect_
5 from abjad.tools import abctools
6 from abjad.tools import datastructuretools
7 from abjad.tools import durationtools
8 from abjad.tools import indicatortools
9 from abjad.tools import lilypondnametools
10 from abjad.tools import mathtools
11 from abjad.tools import schemetools
12 from abjad.tools import scoretools
13 from abjad.tools import sequencetools
14 from abjad.tools import spannertools
15
16
17 class DynamicExpression(abctools.AbjadValueObject):
18     r"""A dynamic phrasing expression.
19
20     :::
21
22     >>> import consort
23     >>> dynamic_expression = consort.DynamicExpression(
24     ...     dynamic_tokens='f p pp pp',
25     ...     transitions=('flared', None),
26     ... )
27     >>> print(format(dynamic_expression))
28     consort.tools.DynamicExpression(
29         dynamic_tokens=datastructuretools.CyclicTuple(
30             ['f', 'p', 'pp', 'pp']
31             ),
32         transitions=datastructuretools.CyclicTuple(
33             ['flared', None]
34             ),
35         )
36
37     .. container:: example
38
39     :::
40
41     >>> music = Staff(r''
42     ...     { c'4 d'4 e'4 f'4 }
43     ...     { g'4 a'4 b'4 }
44     ...     { c''4 }
45     ...   ')
46     >>> print(format(music))

```

```

47      \new Staff {
48      {
49          c'4
50          d'4
51          e'4
52          f'4
53      }
54      {
55          g'4
56          a'4
57          b'4
58      }
59      {
60          c''4
61      }
62  }
63
64  :::
65
66  >>> dynamic_expression(music)
67  >>> print(format(music))
68  \new Staff {
69  {
70      \once \override Hairpin.stencil = #flared-hairpin
71      c'4 \f \>
72      d'4
73      e'4
74      f'4
75  }
76  {
77      g'4 \p \>
78      a'4
79      b'4
80  }
81  {
82      c''4 \pp
83  }
84  }
85
86 .. container:: example
87
88  :::
89
90  >>> music = Staff(r'''
91  ...     { c'4 d'4 e'4 f'4 }
92  ...     { g'4 a'4 b'4 }
93  ...     { c''4 c'4 }
94  ... ''')
95
96  :::
97
98  >>> dynamic_expression(music, seed=2)
99  >>> print(format(music))
100 \new Staff {

```

```

101      {
102          c'4 \pp
103          d'4
104          e'4
105          f'4
106      }
107      {
108          g'4 \c
109          a'4
110          b'4
111      }
112      {
113          \once \override Hairpin.stencil = #flared-hairpin
114          c''4 \f \c
115          c'4 \p
116      }
117  }
118
119 .. container:: example
120
121 :::
122
123     >>> music = Staff("{ c'4 }")
124     >>> dynamic_expression(music, seed=1)
125     >>> print(format(music))
126     \new Staff {
127         {
128             c'4 \p
129         }
130     }
131
132 .. container:: example
133
134 :::
135
136     >>> music = Staff("{ c'4 d'4 }")
137     >>> dynamic_expression(music, seed=1)
138     >>> print(format(music))
139     \new Staff {
140         {
141             c'4 \p \c
142             d'4 \pp
143         }
144     }
145
146 .. container:: example
147
148 :::
149
150     >>> music = Staff("{ r4 c'4 r4 } { r4 d'4 r4 } { r4 e' r4 } ")
151     >>> dynamic_expression(music)
152     >>> print(format(music))
153     \new Staff {
154         {

```

```

155          r4
156          \once \override Hairpin.stencil = #flared-hairpin
157          c'4 \f \>
158          r4
159      }
160      {
161          r4
162          d'4 \p \>
163          r4
164      }
165      {
166          r4
167          e'4 \pp
168          r4
169      }
170  }
171
172 .. container:: example
173 :::
174
175
176     >>> music = Staff("{ c'16 c'16 }")
177     >>> dynamic_expression(music)
178     >>> print(format(music))
179     \new Staff {
180         {
181             c'16 \f
182             c'16
183         }
184     }
185
186 .. container:: example
187 :::
188
189
190     >>> music = Staff("{ c'1 }")
191     >>> dynamic_expression = consort.DynamicExpression(
192         ...     dynamic_tokens='fp',
193         ... )
194     >>> dynamic_expression(music)
195     >>> print(format(music))
196     \new Staff {
197         {
198             c'1 \fp
199         }
200     }
201
202 .. container:: example
203 :::
204
205
206     >>> music = Staff("{ c'1 }")
207     >>> dynamic_expression = consort.DynamicExpression(
208         ...     dynamic_tokens='fp',

```

```

209     ...      unsustained=True,
210     ...      )
211     >>> dynamic_expression(music)
212     >>> print(format(music))
213     \new Staff {
214         {
215             c'1 \p
216         }
217     }
218
219 .. container:: example
220
221 :::
222
223     >>> music = Staff(r'''
224     ... { c'4 d'4 e'4 }
225     ... { c'4 d'4 e'4 }
226     ... { c'4 d'4 e'4 }
227     ... { c'4 d'4 e'4 }
228     ... { c'4 d'4 e'4 }
229     ... ''')
230     >>> dynamic_expression = consort.DynamicExpression(
231     ... division_period=2,
232     ... dynamic_tokens='p ppp',
233     ... start_dynamic_tokens='o',
234     ... stop_dynamic_tokens='o',
235     ... )
236     >>> dynamic_expression(music)
237     >>> print(format(music))
238     \new Staff {
239         {
240             \once \override Hairpin.circled-tip = ##t
241             c'4 \l
242             d'4
243             e'4
244         }
245         {
246             c'4
247             d'4
248             e'4
249         }
250         {
251             \once \override Hairpin.circled-tip = ##t
252             c'4 \p \r
253             d'4
254             e'4
255         }
256         {
257             c'4
258             d'4
259             e'4
260         }
261         {
262             c'4

```

```

263          d'4
264          e'4 \!
265      }
266  }
267
268 .. container:: exmaple
269
270 :::
271
272     >>> music = Staff("{ c'8. } { e'8. } { g'8. }")
273     >>> dynamic_expression = consort.DynamicExpression(
274         ... division_period=2,
275         ... dynamic_tokens='p ppp',
276         ... start_dynamic_tokens='o',
277         ... stop_dynamic_tokens='o',
278         ... )
279     >>> dynamic_expression(music)
280     >>> print(format(music))
281 \new Staff {
282     {
283         \once \override Hairpin.circled-tip = ##t
284         c'8. \<
285     }
286     {
287         e'8.
288     }
289     {
290         g'8. \p
291     }
292 }
293
294 .. container:: example
295
296 :::
297
298     >>> music = Staff(r'''
299     ... { c'8 ~ c'4 }
300     ... \times 3/4 { d'16 d' d' d' r d' d' r }
301     ... ''')
302     >>> dynamic_expression = consort.DynamicExpression(
303         ... dynamic_tokens='mf mp fff',
304         ... start_dynamic_tokens='f',
305         ... stop_dynamic_tokens='mf',
306         ... )
307     >>> dynamic_expression(music)
308     >>> print(format(music))
309 \new Staff {
310     {
311         c'8 \f ~ \>
312         c'4
313     }
314     \tweak #'text #tuplet-number::calc-fraction-text
315     \times 3/4 {
316         d'16 \mf

```

```

317             d'16
318             d'16
319             d'16
320             r16
321             d'16
322             d'16
323             r16
324         }
325     }
326
327 """
328
329     """ CLASS VARIABLES """
330
331     __slots__ = (
332         '_division_period',
333         '_dynamic_tokens',
334         '_only_first',
335         '_start_dynamic_tokens',
336         '_stop_dynamic_tokens',
337         '_transitions',
338         '_unsustained',
339     )
340
341     _transition_types = (
342         'constante',
343         'flared',
344         'simple',
345         None,
346     )
347
348     """ INITIALIZER """
349
350     def __init__(
351         self,
352         dynamic_tokens=('ppp',),
353         division_period=None,
354         only_first=None,
355         start_dynamic_tokens=None,
356         stop_dynamic_tokens=None,
357         transitions=None,
358         unsustained=None,
359     ):
360         dynamic_tokens = self._tokens_to_cyclic_tuple(dynamic_tokens)
361         assert dynamic_tokens
362         self._dynamic_tokens = dynamic_tokens
363         if division_period is not None:
364             division_period = int(division_period)
365             assert 0 < division_period
366         self._division_period = division_period
367         self._start_dynamic_tokens = self._tokens_to_cyclic_tuple(
368             start_dynamic_tokens)
369         self._stop_dynamic_tokens = self._tokens_to_cyclic_tuple(
370             stop_dynamic_tokens)

```

```

371     if isinstance(transitions, (str, type(None))):
372         transitions = [transitions]
373     assert all(_ in self._transition_types for _ in transitions)
374     transitions = datastructuretools.CyclicTuple(transitions)
375     self._transitions = transitions
376     if only_first is not None:
377         only_first = bool(only_first)
378     self._only_first = only_first
379     if unsustained is not None:
380         unsustained = bool(unsustained)
381     self._unsustained = unsustained
382
383     ### SPECIAL METHODS ###
384
385     def __call__(self, music, name=None, seed=0):
386         original_seed = seed
387         current_dynamic = None
388         current_hairpin = None
389         selections, components = self._get_selections(music)
390         #print(selections)
391         #print(components)
392         length = len(components)
393         if self.only_first:
394             length = 1
395             components = components[:1]
396         for index, component in enumerate(components[:-1]):
397             selection = selections[index]
398             dynamic, hairpin, hairpin_override = self._get_attachments(
399                 index, length, seed, original_seed)
400             if dynamic != current_dynamic:
401                 if dynamic.name != 'o':
402                     attach(dynamic, component, name=name)
403                     current_dynamic = dynamic
404             if self.unsustained:
405                 inner_leaves = selection[1:-1]
406                 prototype = scoretools.Rest
407                 if (
408                     len(inner_leaves) and
409                     all(isinstance(_, prototype) for _ in inner_leaves)
410                 ):
411                     hairpin = None
412             if hairpin is not None:
413                 attach(hairpin, selection, name=name)
414                 current_hairpin = hairpin
415             if current_hairpin is not None and hairpin_override is not None:
416                 attach(hairpin_override, component, name=name)
417             seed += 1
418             dynamic, _, _ = self._get_attachments(
419                 length - 1, length, seed, original_seed)
420             if self.unsustained:
421                 if dynamic is not None:
422                     if length == 1:
423                         if not selections or len(selections[0]) < 4:
424                             if dynamic.name in dynamic._composite_dynamic_name_to_steady_state_dynamic_name:

```

```

425                 dynamic_name = dynamic._composite_dynamic_name_to_steady_state_dynamic_name[dynamic.name]
426                 dynamic = indicatortools.Dynamic(dynamic_name)
427             if dynamic != current_dynamic and dynamic.name != 'o':
428                 attach(dynamic, components[-1], name=name)
429             if dynamic.name == 'o' and current_hairpin:
430                 next_leaf = components[-1]._get_leaf(1)
431                 if next_leaf is not None:
432                     current_hairpin._append(next_leaf)
433
434     ### PRIVATE METHODS ###
435
436     def _get_attachments(self, index, length, seed, original_seed):
437         dynamic_seed = seed
438         if self.start_dynamic_tokens:
439             dynamic_seed -= 1
440
441         this_token = None
442         next_token = None
443         this_dynamic = None
444         next_dynamic = None
445         hairpin = None
446         hairpin_override = None
447
448         if length == 1:
449             if self.start_dynamic_tokens:
450                 this_token = self.start_dynamic_tokens[original_seed]
451             elif self.stop_dynamic_tokens:
452                 this_token = self.stop_dynamic_tokens[original_seed]
453             else:
454                 this_token = self.dynamic_tokens[dynamic_seed]
455             if this_token == 'o':
456                 this_token = self.dynamic_tokens[dynamic_seed]
457         elif length == 2:
458             if index == 0:
459                 if self.start_dynamic_tokens:
460                     this_token = self.start_dynamic_tokens[original_seed]
461                 else:
462                     this_token = self.dynamic_tokens[dynamic_seed]
463                 if self.stop_dynamic_tokens:
464                     next_token = self.stop_dynamic_tokens[original_seed]
465                 else:
466                     next_token = self.dynamic_tokens[dynamic_seed + 1]
467             elif index == 1:
468                 if self.stop_dynamic_tokens:
469                     this_token = self.stop_dynamic_tokens[original_seed]
470                 if (
471                     this_token == 'o' and
472                     self.start_dynamic_tokens and
473                     self.start_dynamic_tokens[original_seed] == 'o'
474                 ):
475                     this_token = self.dynamic_tokens[dynamic_seed]
476                 else:
477                     this_token = self.dynamic_tokens[dynamic_seed]
478             if this_token == next_token == 'o':

```

```

479         next_token = self.dynamic_tokens[dynamic_seed]
480     else:
481         #print('!!!', index)
482         if index == 0:
483             if self.start_dynamic_tokens:
484                 this_token = self.start_dynamic_tokens[original_seed]
485                 next_token = self.dynamic_tokens[dynamic_seed + 1]
486                 #print('A1', this_token, next_token)
487             else:
488                 this_token = self.dynamic_tokens[dynamic_seed]
489                 next_token = self.dynamic_tokens[dynamic_seed + 1]
490                 #print('A2', this_token, next_token)
491         elif index == length - 1: # Last component.
492             if self.stop_dynamic_tokens:
493                 this_token = self.stop_dynamic_tokens[original_seed]
494                 #print('B1', this_token, next_token)
495             else:
496                 this_token = self.dynamic_tokens[dynamic_seed]
497                 #print('B2', this_token, next_token)
498         elif index == length - 2: # Next to last component.
499             this_token = self.dynamic_tokens[dynamic_seed]
500             if self.stop_dynamic_tokens:
501                 next_token = self.stop_dynamic_tokens[original_seed]
502                 #print('C1', this_token, next_token)
503             else:
504                 next_token = self.dynamic_tokens[dynamic_seed + 1]
505                 #print('C2', this_token, next_token)
506             else:
507                 this_token = self.dynamic_tokens[dynamic_seed]
508                 next_token = self.dynamic_tokens[dynamic_seed + 1]
509                 #print('D1', this_token, next_token)
510
511     this_dynamic = indicatortools.Dynamic(this_token)
512     this_dynamic_ordinal = mathtools.NegativeInfinity()
513     if this_dynamic.name != 'o':
514         this_dynamic_ordinal = this_dynamic.ordinal
515     if next_token is not None:
516         next_dynamic = indicatortools.Dynamic(next_token)
517         next_dynamic_ordinal = mathtools.NegativeInfinity()
518         if next_dynamic.name != 'o':
519             next_dynamic_ordinal = next_dynamic.ordinal
520
521     if next_dynamic is not None:
522         if this_dynamic_ordinal < next_dynamic_ordinal:
523             hairpin = spannertools.Crescendo(include_rests=True)
524         elif next_dynamic_ordinal < this_dynamic_ordinal:
525             hairpin = spannertools.Decrescendo(include_rests=True)
526
527     if hairpin is not None:
528         transition = self.transitions[seed]
529         if transition == 'constante':
530             hairpin = spannertools.Crescendo(include_rests=True)
531         if transition in ('flared', 'constante'):
532             hairpin_override = lilypondnametools.LilyPondGrobOverride(

```

```

533             grob_name='Hairpin',
534             is_once=True,
535             property_path='stencil',
536             value=schemetools.Scheme('{}-hairpin'.format(transition)),
537             )
538     if this_dynamic.name == 'o' or next_dynamic.name == 'o':
539         hairpin_override = lilypondnametools.LilyPondGrobOverride(
540             grob_name='Hairpin',
541             is_once=True,
542             property_path='circled-tip',
543             value=True,
544             )
545
546     #print(index, this_dynamic, next_dynamic, hairpin)
547
548     return this_dynamic, hairpin, hairpin_override
549
550 def _partition_selections(self, music):
551     period = self.division_period or 1
552     selections = [_.select_leaves() for _ in music]
553     parts = sequencetools.partition_sequence_by_counts(
554         selections, [period], cyclic=True, overhang=True)
555     if len(parts[-1]) < period and 1 < len(parts):
556         part = parts.pop()
557         parts[-1].extend(part)
558     selections = []
559     for part in parts:
560         selection = part[0]
561         for next_selection in part[1:]:
562             selection = selection + next_selection
563         selections.append(selection)
564     return selections
565
566 def _reorganize_selections(self, selections):
567     prototype = (scoretools.Note, scoretools.Chord)
568     for i, leaf in enumerate(selections[0]):
569         if isinstance(leaf, prototype):
570             break
571     selections[0] = selections[0][i:]
572     for i, leaf in enumerate(reversed(selections[-1])):
573         if isinstance(leaf, prototype):
574             break
575     if i == 0:
576         i = None
577     else:
578         i = -i
579     selections[-1] = selections[-1][:i]
580     if len(selections) == 1:
581         return selections
582     for i in range(len(selections) - 1):
583         selection_one, selection_two = selections[i], selections[i + 1]
584         for j, leaf in enumerate(selection_two):
585             if isinstance(leaf, prototype):
586                 break

```

```

587     if 0 < j:
588         left, right = selection_two[:j], selection_two[j:]
589         selection_one = selection_one + left
590         selection_two = right
591         selections[i] = selection_one
592         selections[i + 1] = selection_two
593     return selections
594
595 def _get_selections(self, music):
596     """Gets selections and attach components from 'music'.
597
598     ... container:: example
599
600     :::
601
602     >>> music = Staff(r'''
603     ...     { c'4 d'4 e'4 f'4 }
604     ...     { g'4 a'4 b'4 }
605     ...     { c''4 }
606     ... ''')
607     >>> dynamic_expression = consort.DynamicExpression("f")
608     >>> result = dynamic_expression._get_selections(music)
609     >>> selections, attach_components = result
610     >>> for _ in selections:
611         ...
612         -
613         ...
614         ContiguousSelection(Note("c'4"), Note("d'4"), Note("e'4"), Note("f'4"), Note("g'4"))
614         ContiguousSelection(Note("g'4"), Note("a'4"), Note("b'4"), Note("c''4"))
615
616     :::
617
618     >>> for _ in attach_components:
619         ...
619         -
620         ...
621         Note("c'4")
622         Note("g'4")
623         Note("c''4")
624
625     ... container:: example
626
627     :::
628
629     >>> music = Staff(r'''
630     ...     { c'4 d'4 e'4 }
631     ...     { f'4 g'4 a'4 }
632     ...     { b'4 c''4 }
633     ... ''')
634     >>> dynamic_expression = consort.DynamicExpression("f")
635     >>> result = dynamic_expression._get_selections(music)
636     >>> selections, attach_components = result
637     >>> for _ in selections:
638         ...
638         -
639         ...
639         ContiguousSelection(Note("c'4"), Note("d'4"), Note("e'4"), Note("f'4"))

```

```

641     ContiguousSelection(Note("f'4"), Note("g'4"), Note("a'4"), Note("b'4"))
642     ContiguousSelection(Note("b'4"), Note("c''4"))
643
644     :::
645
646     >>> for _ in attach_components:
647         ...
648         ...
649         Note("c'4")
650         Note("f'4")
651         Note("b'4")
652         Note("c''4")
653
654     ... container:: example
655
656     :::
657
658     >>> music = Staff(r'''
659     ...
660     { c'8 d'8 e'8 }
661     ...
662     { f'8 g'8 a'8 }
663     ...
664     { b'32 c''16. }
665     ...
666     ''')
667
668     >>> result = dynamic_expression._get_selections(music)
669     >>> selections, attach_components = result
670     >>> for _ in selections:
671         ...
672         ...
673         ...
674         Note("c'8")
675         Note("f'8")
676         Note("c''16.")
677
678     ... container:: example
679
680     :::
681
682     :::
683
684     >>> music = Staff("{ r4 c'4 r4 } { r4 d'4 r4 } { r4 e' r4 } ")
685     >>> result = dynamic_expression._get_selections(music)
686     >>> selections, attach_components = result
687     >>> for _ in selections:
688         ...
689         ...
690         ContiguousSelection(Note("c'4"), Rest('r4'), Rest('r4'), Note("d'4"))
691         ContiguousSelection(Note("d'4"), Rest('r4'), Rest('r4'), Note("e'4"))
692
693     :::
694

```

```

695     >>> for _ in attach_components:
696         ...
697         ...
698         Note("c'4")
699         Note("d'4")
700         Note("e'4")
701
702     ... container:: example
703
704     :::
705
706     >>> music = Staff("{ c'8. } { e'8. } { g'8. }")
707     >>> dynamic_expression = consort.DynamicExpression(
708         ...     division_period=2,
709         ...     dynamic_tokens='p ppp',
710         ...     start_dynamic_tokens='o',
711         ...     stop_dynamic_tokens='o',
712         ... )
713     >>> result = dynamic_expression._get_selections(music)
714     >>> selections, attach_components = result
715     >>> for _ in selections:
716         ...
717         ...
718     ContiguousSelection(Note("c'8."), Note("e'8."), Note("g'8."))
719
720     :::
721
722     >>> for _ in attach_components:
723         ...
724         ...
725         Note("c'8.")
726         Note("g'8.")
727
728     """
729     #print('---', music)
730     initial_selections = self._partition_selections(music)
731     #print(' ', initial_selections)
732     initial_selections = self._reorganize_selections(initial_selections)
733     #print(' ', initial_selections)
734     attach_components = []
735     selections = []
736     assert len(initial_selections)
737     for i, selection in enumerate(initial_selections):
738         #print(' ', i, selection)
739         if i < len(initial_selections) - 1:
740             #print(' ', 'A')
741             selection = selection + (initial_selections[i + 1][0],)
742             selections.append(selection)
743             attach_components.append(selection[0])
744             elif ((selection.get_duration() <= durationtools.Duration(1, 8) and
745                   1 < len(selections)) or len(selection) == 1
746                   ):
747                 #print(' ', 'B')
748                 attach_components.append(selection[-1])

```

```

749         if selections:
750             #print(' ', 'B1')
751             selections[-1] = selections[-1] + selection[1:]
752         elif (
753             durationtools.Duration(1, 8) < (
754                 selection[-1]._get_timespan().start_offset -
755                 selection[0]._get_timespan().start_offset
756             )
757         ):
758             #print(' ', 'C')
759             selections.append(selection)
760             attach_components.append(selection[0])
761             attach_components.append(selection[-1])
762         else:
763             #print(' ', 'D')
764             attach_components.append(selection[0])
765             #print(' ', initial_selections)
766             #print(' ', attach_components)
767             return selections, attach_components
768
769     def _tokens_to_cyclic_tuple(self, tokens):
770         if tokens is None:
771             return tokens
772         if isinstance(tokens, str):
773             tokens = tokens.split()
774         for token in tokens:
775             if token == 'o':
776                 continue
777             assert token in indicatortools.Dynamic._dynamic_names
778             assert len(tokens)
779             tokens = datastructuretools.CyclicTuple(tokens)
780         return tokens
781
782     ### PUBLIC PROPERTIES ###
783
784     @property
785     def division_period(self):
786         return self._division_period
787
788     @property
789     def dynamic_tokens(self):
790         return self._dynamic_tokens
791
792     @property
793     def only_first(self):
794         return self._only_first
795
796     @property
797     def period(self):
798         return self._period
799
800     @property
801     def start_dynamic_tokens(self):
802         return self._start_dynamic_tokens

```

```

803
804     @property
805     def stop_dynamic_tokens(self):
806         return self._stop_dynamic_tokens
807
808     @property
809     def transitions(self):
810         return self._transitions
811
812     @property
813     def unsustained(self):
814         return self._unsustained

```

## A.20 CONSORTTOOLS.FLOODEDTIMESPANMAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import timespanools
3 from consort.tools.TimespanMaker import TimespanMaker
4
5
6 class FloodedTimespanMaker(TimespanMaker):
7     r'''A flooded timespan maker.
8
9     :::
10
11     >>> import consort
12     >>> timespan_maker = consort.FloodedTimespanMaker()
13     >>> print(format(timespan_maker))
14     consort.tools.FloodedTimespanMaker()
15
16     :::
17
18     >>> music_specifiers = {
19     ...     'Violin Voice': 'violin music',
20     ...     'Cello Voice': 'cello music',
21     ... }
22     >>> target_timespan = timespanools.Timespan((1, 2), (2, 1))
23     >>> timespan_inventory = timespan_maker(
24     ...     music_specifiers=music_specifiers,
25     ...     target_timespan=target_timespan,
26     ... )
27     >>> print(format(timespan_inventory))
28     timespanools.TimespanInventory(
29         [
30             consort.tools.PerformedTimespan(
31                 start_offset=durationtools.Offset(1, 2),
32                 stop_offset=durationtools.Offset(2, 1),
33                 musicSpecifier='cello music',
34                 voiceName='Cello Voice',
35                 ),
36             consort.tools.PerformedTimespan(
37                 start_offset=durationtools.Offset(1, 2),
38                 stop_offset=durationtools.Offset(2, 1),
39                 musicSpecifier='violin music',

```

```

40                 voice_name='Violin Voice',
41                 ),
42             ],
43         )
44
45     :::
46
47     >>> musicSpecifier = consort.CompositeMusicSpecifier(
48     ...     primary_musicSpecifier='one',
49     ...     primary_voice_name='Viola 1 RH',
50     ...     rotation_indices=(0, 1, -1),
51     ...     secondary_voice_name='Viola 1 LH',
52     ...     secondary_musicSpecifier=consort.MusicSpecifierSequence(
53     ...         application_rate='phrase',
54     ...         music_specifiers=['two', 'three', 'four'],
55     ...         ),
56     ...     )
57     >>> musicSpecifiers = {
58     ...     'Viola 1 Performer Group': musicSpecifier,
59     ...     }
60     >>> timespanInventory = timespan_maker(
61     ...     music_specifiers=musicSpecifiers,
62     ...     target_timespan=target_timespan,
63     ...     )
64     >>> print(format(timespanInventory))
65     timespantools.TimespanInventory(
66         [
67             consort.tools.PerformedTimespan(
68                 start_offset=durationtools.Offset(1, 2),
69                 stop_offset=durationtools.Offset(2, 1),
70                 musicSpecifier='two',
71                 voice_name='Viola 1 LH',
72                 ),
73             consort.tools.PerformedTimespan(
74                 start_offset=durationtools.Offset(1, 2),
75                 stop_offset=durationtools.Offset(2, 1),
76                 musicSpecifier='one',
77                 voice_name='Viola 1 RH',
78                 ),
79         ]
80     )
81
82     """
83
84     ### CLASS VARIABLES ###
85
86     __slots__ = ()
87
88     ### INITIALIZER ###
89
90     def __init__(
91         self,
92         output_masks=None,
93         padding=None,

```

```

94     seed=None,
95     timespanSpecifier=None,
96     ):
97     TimespanMaker.__init__(
98         self,
99         output_masks=output_masks,
100        padding=padding,
101        seed=seed,
102        timespanSpecifier=timespanSpecifier,
103        )
104
105    ### PRIVATE METHODS ###
106
107    def _make_timespans(
108        self,
109        layer=None,
110        musicSpecifiers=None,
111        targetTimespan=None,
112        timespanInventory=None,
113        ):
114        startOffset = targetTimespan.startOffset
115        durations = [targetTimespan.duration]
116        newTimespans = timespantools.TimespanInventory()
117        for contextName, musicSpecifier in musicSpecifiers.items():
118            timespans = musicSpecifier(
119                durations=durations,
120                layer=layer,
121                output_masks=self.output_masks,
122                padding=self.padding,
123                seed=self.seed,
124                startOffset=startOffset,
125                timespanSpecifier=self.timespanSpecifier,
126                voiceName=contextName,
127                )
128            newTimespans.extend(timespans)
129    return newTimespans

```

## A.21 CONSORTTOOLS.GRACEHANDLER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import collections
4 from abjad import attach
5 from abjad import new
6 from abjad import override
7 from abjad.tools import abctools
8 from abjad.tools import datastructuretools
9 from abjad.tools import mathtools
10 from abjad.tools import schemetools
11 from abjad.tools import scoretools
12 from abjad.tools import selectiontools
13
14
15 class GraceHandler(abctools.AbjadValueObject):

```

```

16     r'''A grace maker.
17
18     :::
19
20     >>> import consort
21     >>> grace_handler = consort.GraceHandler(
22     ...     counts=(0, 1, 0, 0, 2),
23     ...     )
24     >>> print(format(grace_handler))
25     consort.tools.GraceHandler(
26         counts=datastructuretools.CyclicTuple(
27             [0, 1, 0, 0, 2]
28         ),
29     )
30
31     '''
32
33     ### CLASS VARIABLES ###
34
35     __slots__ = (
36         '_counts',
37         '_only_if_preceded_by_nonsilence',
38         '_only_if_preceded_by_silence',
39     )
40
41     ### INITIALIZER ###
42
43     def __init__(
44         self,
45         counts=(1,),
46         only_if_preceded_by_nonsilence=None,
47         only_if_preceded_by_silence=None,
48     ):
49         if not counts:
50             counts = (0,)
51         if not isinstance(counts, collections.Sequence):
52             counts = (counts,)
53         assert len(counts)
54         assert mathtools.all_are_nonnegative_integer_equivalent_numbers(
55             counts)
56         self._counts = datastructuretools.CyclicTuple(counts)
57         if only_if_preceded_by_nonsilence is not None:
58             only_if_preceded_by_nonsilence = bool(
59                 only_if_preceded_by_nonsilence)
60         if only_if_preceded_by_silence is not None:
61             only_if_preceded_by_silence = bool(
62                 only_if_preceded_by_silence)
63         assert not (
64             only_if_preceded_by_silence and only_if_preceded_by_nonsilence
65         )
66         self._only_if_preceded_by_silence = only_if_preceded_by_silence
67         self._only_if_preceded_by_nonsilence = only_if_preceded_by_nonsilence
68
69     ### SPECIAL METHODS ###

```

```

70
71     def __call__(
72         self,
73         logical_tie,
74         seed=0,
75     ):
76         assert isinstance(logical_tie, selectiontools.LogicalTie)
77         if self.counts is None:
78             return
79         previous_leaf = logical_tie.head._get_leaf(-1)
80         if previous_leaf is None:
81             return
82         silence_prototype = (
83             scoretools.Rest,
84             scoretools.MultimeasureRest,
85             scoretools.Skip,
86         )
87         if self.only_if_preceded_by_silence:
88             if not isinstance(previous_leaf, silence_prototype):
89                 return
90         if self.only_if_preceded_by_nonsense:
91             if isinstance(previous_leaf, silence_prototype):
92                 return
93         grace_count = self.counts[seed]
94         if not grace_count:
95             return
96         kind = 'after'
97         leaf_to_attach_to = previous_leaf
98         leaves = []
99         notes = scoretools.make_notes([0], [(1, 16)] * grace_count)
100        leaves.extend(notes)
101        assert len(leaves)
102        grace_container = scoretools.GraceContainer(
103            leaves,
104            kind=kind,
105        )
106        override(grace_container).flag.stroke_style = \
107            schemetools.Scheme('grace', force_quotes=True)
108        override(grace_container).script.font_size = 0.5
109        attach(grace_container, leaf_to_attach_to)

110
111     ### PRIVATE METHODS ###

112
113     @staticmethod
114     def _process_session(segment_maker):
115         import consort
116         counter = collections.Counter()
117         attack_point_map = segment_maker.attack_point_map
118         for logical_tie in attack_point_map:
119             musicSpecifier = \
120                 consort.SegmentMaker.logical_tie_to_musicSpecifier(
121                     logical_tie)
122             if not musicSpecifier:
123                 continue

```

```

124     grace_handler = music_specifier.grace_handler
125     if not grace_handler:
126         continue
127     previous_leaf = logical_tie.head._get_leaf(-1)
128     if previous_leaf is None:
129         continue
130     if music_specifier not in counter:
131         seed = music_specifier.seed or 0
132         counter[music_specifier] = seed
133     seed = counter[music_specifier]
134     grace_handler(
135         logical_tie,
136         seed=seed,
137     )
138     counter[music_specifier] += 1
139
140     ### PUBLIC METHODS ###
141
142     def reverse(self):
143         counts = self.counts
144         if counts is not None:
145             counts = counts.reverse()
146         return new(self,
147                    counts=counts,
148                    )
149
150     def rotate(self, n=0):
151         counts = self.counts
152         if counts is not None:
153             counts = counts.rotate(n)
154         return new(self,
155                    counts=counts,
156                    )
157
158     ### PUBLIC PROPERTIES ###
159
160     @property
161     def counts(self):
162         return self._counts
163
164     @property
165     def only_if_preceded_by_nonsilence(self):
166         return self._only_if_preceded_by_nonsilence
167
168     @property
169     def only_if_preceded_by_silence(self):
170         return self._only_if_preceded_by_silence

```

## A.22 CONSORTTOOLS.HARMONICEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import pitchtools
3 from abjad.tools import scoretools
4 from consort.tools.LogicalTieExpression import LogicalTieExpression

```

```

5
6
7 class HarmonicExpression(LogicalTieExpression):
8     r'''A harmonic expression.
9
10    :::
11
12    >>> import consort
13    >>> harmonic_expression = consort.HarmonicExpression()
14    >>> print(format(harmonic_expression))
15    consort.tools.HarmonicExpression(
16        touch_interval=pitchtools.NamedInterval('+P4'),
17        )
18
19    :::
20
21    >>> staff = Staff("c'4 d'4 ~ d'4 e'4")
22    >>> logical_tie = inspect_(staff[1]).get_logical_tie()
23    >>> harmonic_expression(logical_tie)
24    >>> print(format(staff))
25    \new Staff {
26        c'4
27        <
28            d'
29            \tweak #'style #'harmonic
30            g'
31            >4 ~
32            <
33            d'
34            \tweak #'style #'harmonic
35            g'
36            >4
37            e'4
38        }
39
40    ,,
41
42    ### CLASS VARIABLES ###
43
44    __slots__ = (
45        '_touch_interval',
46        )
47
48    ### INITIALIZER ###
49
50    def __init__(
51        self,
52        touch_interval='P4',
53        ):
54        touch_interval = pitchtools.NamedInterval(touch_interval)
55        self._touch_interval = touch_interval
56
57    ### SPECIAL METHODS ###
58
```

```

59     def __call__(  
60         self,  
61         logical_tie,  
62         pitch_range=None,  
63     ):  
64         for i, leaf in enumerate(logical_tie):  
65             stopped_pitch = leaf.written_pitch  
66             touched_pitch = stopped_pitch.transpose(self.touch_interval)  
67             chord = scoretools.Chord(leaf)  
68             chord.written_pitches = [stopped_pitch, touched_pitch]  
69             #chord.note_heads[0].is_parenthesized = True  
70             #chord.note_heads[0].tweak.font_size = -4  
71             chord.note_heads[1].tweak.style = 'harmonic'  
72             self._replace(leaf, chord)  
73  
74     ### PUBLIC PROPERTIES ###  
75  
76     @property  
77     def touch_interval(self):  
78         return self._touch_interval

```

## A.23 CONSORTTOOLS.HASHCACHINGOBJECT

```

1 # -*- encoding: utf-8 -*-  
2 from abjad.tools import abctools  
3  
4  
5 class HashCachingObject(abctools.AbjadValueObject):  
6  
7     ### CLASS VARIABLES ###  
8  
9     __slots__ = (  
10         '_hash',  
11     )  
12  
13     ### INITIALIZER ###  
14  
15     def __init__(self):  
16         self._hash = None  
17  
18     ### SPECIAL METHODS ###  
19  
20     def __eq__(self, expr):  
21         if isinstance(expr, type(self)):  
22             if format(self) == format(expr):  
23                 return True  
24             return False  
25  
26     def __hash__(self):  
27         if self._hash is None:  
28             hash_values = (type(self), format(self))  
29             self._hash = hash(hash_values)  
30         return self._hash

```

## A.24 CONSORTTOOLS.KEYCLUSTEREXPRESSION

```
1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import attach
4 from abjad.tools import indicatortools
5 from abjad.tools import pitchtools
6 from abjad.tools import scoretools
7 from abjad.tools import selectiontools
8 from consort.tools.LogicalTieExpression import LogicalTieExpression
9
10
11 class KeyClusterExpression(LogicalTieExpression):
12     r'''A key cluster expression.
13
14     >>> import consort
15     >>> key_cluster_expression = consort.KeyClusterExpression(
16         ...     arpeggio_direction=Up,
17         ...     include_black_keys=False,
18         ...     )
19     >>> print(format(key_cluster_expression))
20     consort.tools.KeyClusterExpression(
21         arpeggio_direction=Up,
22         include_black_keys=False,
23         include_white_keys=True,
24         staff_space_width=5,
25         )
26
27     :::
28
29     >>> staff = Staff("c'4 d'4 ~ d'4 e'4")
30     >>> logical_tie = inspect_(staff[1]).get_logical_tie()
31     >>> key_cluster_expression(logical_tie)
32     >>> print(format(staff))
33     \new Staff {
34         c'4
35         \arpeggioArrowUp
36         \once \override Accidental.stencil = ##f
37         \once \override AccidentalCautionary.stencil = ##f
38         \once \override Arpeggio.X-offset = #-2
39         \once \override NoteHead.stencil = #ly:text-interface::print
40         \once \override NoteHead.text = \markup {
41             \filled-box #'(-0.6 . 0.6) #'(-0.7 . 0.7) #0.25
42         }
43         <b d' f'>4 \arpeggio ~
44             ^ \markup {
45                 \center-align
46                 \natural
47             }
48         \once \override Accidental.stencil = ##f
49         \once \override AccidentalCautionary.stencil = ##f
50         \once \override Arpeggio.X-offset = #-2
51         \once \override NoteHead.stencil = #ly:text-interface::print
52         \once \override NoteHead.text = \markup {
```

```

53             \filled-box #'(-0.6 . 0.6) #'(-0.7 . 0.7) #0.25
54         }
55         <b d' f'>4
56         e'4
57     }
58
59     ...
60
61     ### CLASS VARIABLES ###
62
63     __slots__ = (
64         '_arpeggio_direction',
65         '_include_black_keys',
66         '_include_white_keys',
67         '_staff_space_width',
68     )
69
70     ### INITIALIZER ###
71
72     def __init__(
73         self,
74         arpeggio_direction=None,
75         include_black_keys=True,
76         include_white_keys=True,
77         staff_space_width=5,
78     ):
79         assert 2 < staff_space_width and (int(staff_space_width) % 2)
80         assert include_black_keys or include_white_keys
81         assert arpeggio_direction in (Up, Down, None)
82         self._arpeggio_direction = arpeggio_direction
83         self._include_black_keys = bool(include_black_keys)
84         self._include_white_keys = bool(include_white_keys)
85         self._staff_space_width = int(staff_space_width)
86
87     ### SPECIAL METHODS ###
88
89     def __call__(
90         self,
91         logical_tie,
92         pitch_range=None,
93     ):
94         assert isinstance(logical_tie, selectiontools.LogicalTie), logical_tie
95         center_pitch = logical_tie[0].written_pitch
96         chord_pitches = self._get_chord_pitches(center_pitch)
97         if pitch_range is not None:
98             maximum_pitch = max(chord_pitches)
99             minimum_pitch = min(chord_pitches)
100            if maximum_pitch not in pitch_range:
101                interval = maximum_pitch - pitch_range.stop_pitch
102                center_pitch = center_pitch.transpose(interval)
103                chord_pitches = self._get_chord_pitches(center_pitch)
104            elif minimum_pitch not in pitch_range:
105                interval = minimum_pitch - pitch_range.start_pitch
106                center_pitch = center_pitch.transpose(interval)

```

```

107         chord_pitches = self._get_chord_pitches(center_pitch)
108     for i, leaf in enumerate(logical_tie):
109         chord = scoretools.Chord(leaf)
110         chord.written_pitches = chord_pitches
111         self._replace(leaf, chord)
112         if i:
113             key_cluster = indicatortools.KeyCluster(
114                 include_black_keys=self.include_black_keys,
115                 include_white_keys=self.include_white_keys,
116                 suppress_markup=True,
117                 )
118             attach(key_cluster, chord)
119         else:
120             key_cluster = indicatortools.KeyCluster(
121                 include_black_keys=self.include_black_keys,
122                 include_white_keys=self.include_white_keys,
123                 markup_direction=Up,
124                 )
125             attach(key_cluster, chord)
126         if self.arpeggio_direction is not None:
127             arpeggio = indicatortools.Arpeggio(
128                 direction=self.arpeggio_direction,
129                 )
130             attach(arpeggio, chord)
131
132     ### PRIVATE PROPERTIES ###
133
134     def _get_chord_pitches(self, center_pitch):
135         starting_diatonic_number = \
136             center_pitch.diatonic_pitch_number - (self.staff_space_width // 2)
137         diatonic_numbers = [starting_diatonic_number]
138         for i in range(1, (self.staff_space_width // 2) + 1):
139             step = 2 * i
140             diatonic_number = starting_diatonic_number + step
141             diatonic_numbers.append(diatonic_number)
142         chromatic_numbers = [
143             (12 * (x // 7)) +
144             pitchtools.PitchClass._diatonic_pitch_class_number_to_pitch_class_number[
145                 x % 7]
146             for x in diatonic_numbers
147             ]
148         chord_pitches = [pitchtools.NamedPitch(x)
149                         for x in chromatic_numbers]
150         return chord_pitches
151
152     ### PUBLIC PROPERTIES ###
153
154     @property
155     def arpeggio_direction(self):
156         return self._arpeggio_direction
157
158     @property
159     def include_black_keys(self):
160         return self._include_black_keys

```

```

161
162     @property
163     def include_white_keys(self):
164         return self._include_white_keys
165
166     @property
167     def staff_space_width(self):
168         return self._staff_space_width

```

## A.25 CONSORTTOOLS.LEAFEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import attach
3 from abjad import inspect_
4 from abjad import mutate
5 from abjad import select
6 from abjad.tools import durationtools
7 from abjad.tools import scoretools
8 from abjad.tools import selectiontools
9 from consort.tools.HashCachingObject import HashCachingObject
10
11
12 class LeafExpression(HashCachingObject):
13
14     """ CLASS VARIABLES """
15
16     __slots__ = (
17         '_attachments',
18         '_leaf',
19     )
20
21     """ INITIALIZER """
22
23     def __init__(
24         self,
25         leaf=None,
26         attachments=None,
27     ):
28         HashCachingObject.__init__(self)
29         prototype = scoretools.Leaf
30         if leaf is not None:
31             if isinstance(leaf, prototype):
32                 leaf = mutate(leaf).copy()
33             elif issubclass(leaf, prototype):
34                 leaf = leaf()
35             else:
36                 raise ValueError(leaf)
37         self._leaf = leaf
38         if attachments is not None:
39             attachments = tuple(attachments)
40         self._attachments = attachments
41
42     """ SPECIAL METHODS """
43

```

```

44     def __call__(self, expr):
45         if isinstance(expr, scoretools.Leaf):
46             expr = select(expr)
47         assert isinstance(expr, selectiontools.Selection)
48         for i, old_leaf in enumerate(expr):
49             assert isinstance(old_leaf, scoretools.Leaf)
50             new_leaf = self._make_new_leaf(old_leaf)
51
52             timespan = old_leaf._timespan
53             start_offset = old_leaf._start_offset
54             stop_offset = old_leaf._stop_offset
55             mutate(old_leaf).replace(new_leaf)
56             new_leaf._timespan = timespan
57             new_leaf._start_offset = start_offset
58             new_leaf._stop_offset = stop_offset
59
60             if i == 0 and self.attachments:
61                 for attachment in self.attachments:
62                     attach(attachment, new_leaf)
63
64     ### PRIVATE METHODS ###
65
66     def _make_new_leaf(self, old_leaf):
67         duration = old_leaf.written_duration
68         if isinstance(self.leaf, scoretools.Note):
69             new_leaf = scoretools.Note(self.leaf.written_pitch, duration)
70         elif isinstance(self.leaf, scoretools.Chord):
71             new_leaf = scoretools.Chord(self.leaf.written_pitches, duration)
72         elif isinstance(self.leaf, scoretools.Rest):
73             new_leaf = scoretools.Rest(duration)
74         elif isinstance(self.leaf, scoretools.Skip):
75             new_leaf = scoretools.Skip(duration)
76         prototype = durationtools.Multiplier
77         if inspect_(old_leaf).has_indicator(prototype):
78             multiplier = inspect_(old_leaf).get_indicator(prototype)
79             attach(multiplier, new_leaf)
80         return new_leaf
81
82     ### PUBLIC PROPERTIES ###
83
84     @property
85     def attachments(self):
86         return self._attachments
87
88     @property
89     def leaf(self):
90         return self._leaf

```

## A.26 CONSORTTOOLS.LOGICALTIEEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 import abc
3 from abjad import attach
4 from abjad import detach

```

```

5 from abjad import inspect_
6 from abjad import mutate
7 from abjad.tools import abctools
8 from abjad.tools import scoretools
9
10
11 class LogicalTieExpression(abctools.AbjadValueObject):
12
13     ### CLASS VARIABLES ###
14
15     __slots__ = ()
16
17     ### SPECIAL METHODS ###
18
19     @abc.abstractmethod
20     def __call__(self, logical_tie, pitch_range=None):
21         raise NotImplementedError
22
23     ### PRIVATE METHODS ###
24
25     def _replace(self, old_leaf, new_leaf):
26         grace_containers = inspect_(old_leaf).get_grace_containers('after')
27         if grace_containers:
28             old_grace_container = grace_containers[0]
29             grace_notes = old_grace_container.select_leaves()
30             detach(scoretools.GraceContainer, old_leaf)
31             indicators = inspect_(old_leaf).get_indicators()
32             for indicator in indicators:
33                 detach(indicator, old_leaf)
34
35             timespan = old_leaf._timespan
36             start_offset = old_leaf._start_offset
37             stop_offset = old_leaf._stop_offset
38             mutate(old_leaf).replace(new_leaf)
39             new_leaf._timespan = timespan
40             new_leaf._start_offset = start_offset
41             new_leaf._stop_offset = stop_offset
42
43             if grace_containers:
44                 new_grace_container = scoretools.GraceContainer(
45                     grace_notes,
46                     kind='after',
47                 )
48                 attach(new_grace_container, new_leaf)
49                 for indicator in indicators:
50                     attach(indicator, new_leaf)

```

## A.27 CONSORTTOOLS.MUSICSETTING

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import collections
4 from abjad import inspect_
5 from abjad import new

```

```

6 from abjad.tools import abctools
7 from abjad.tools import systemtools
8 from abjad.tools import timespantools
9
10
11 class MusicSetting(abctools.AbjadValueObject):
12     r'''A music setting.
13
14     :::
15
16     >>> import consort
17     >>> red_setting = consort.MusicSetting(
18         ...     timespan_maker=consort.TaleaTimespanMaker(
19             ...         initial_silence_talea=rhythmmakertools.Talea(
20                 ...             counts=(0, 4),
21                 ...             denominator=16,
22                 ...             ),
23                 ...         playing_talea=rhythmmakertools.Talea(
24                     ...             counts=(4, 8, 4),
25                     ...             denominator=16,
26                     ...             ),
27                     ...             ),
28             ...         viola_rh=consort.tools.MusicSpecifier(),
29             ...         violin_1_rh=consort.tools.MusicSpecifier(),
30             ...         violin_2_rh=consort.tools.MusicSpecifier(),
31             ...         )
32     >>> print(format(red_setting))
33     consort.tools.MusicSetting(
34         timespan_maker=consort.tools.TaleaTimespanMaker(
35             initial_silence_talea=rhythmmakertools.Talea(
36                 counts=(0, 4),
37                 denominator=16,
38                 ),
39                 playing_talea=rhythmmakertools.Talea(
40                     counts=(4, 8, 4),
41                     denominator=16,
42                     ),
43                     playing_groupings=(1,),
44                     repeat=True,
45                     silence_talea=rhythmmakertools.Talea(
46                         counts=(4,),
47                         denominator=16,
48                         ),
49                         step_anchor=Right,
50                         synchronize_groupings=False,
51                         synchronize_step=False,
52                         ),
53                         viola_rh=consort.tools.MusicSpecifier(),
54                         violin_1_rh=consort.tools.MusicSpecifier(),
55                         violin_2_rh=consort.tools.MusicSpecifier(),
56                         )
57
58     :::
59

```

```

60     >>> layer = 1
61     >>> segment_timespan = timespantools.Timespan(1, 2)
62     >>> from abjad.tools import templatetools
63     >>> score_template = consort.StringQuartetScoreTemplate()
64     >>> timespan_inventory = red_setting(
65         ...     layer=layer,
66         ...     score_template=score_template,
67         ...     segment_timespan=segment_timespan,
68         ... )
69
70     :::
71
72     >>> print(format(timespan_inventory))
73     timespantools.TimespanInventory(
74         [
75             consort.tools.PerformedTimespan(
76                 start_offset=durationtools.Offset(1, 1),
77                 stop_offset=durationtools.Offset(5, 4),
78                 layer=1,
79                 musicSpecifier=consort.tools.MusicSpecifier(),
80                 voice_name='Violin 1 Bowing Voice',
81                 ),
82             consort.tools.PerformedTimespan(
83                 start_offset=durationtools.Offset(1, 1),
84                 stop_offset=durationtools.Offset(3, 2),
85                 layer=1,
86                 musicSpecifier=consort.tools.MusicSpecifier(),
87                 voice_name='Viola Bowing Voice',
88                 ),
89             consort.tools.PerformedTimespan(
90                 start_offset=durationtools.Offset(5, 4),
91                 stop_offset=durationtools.Offset(3, 2),
92                 layer=1,
93                 musicSpecifier=consort.tools.MusicSpecifier(),
94                 voice_name='Violin 2 Bowing Voice',
95                 ),
96             consort.tools.PerformedTimespan(
97                 start_offset=durationtools.Offset(3, 2),
98                 stop_offset=durationtools.Offset(2, 1),
99                 layer=1,
100                musicSpecifier=consort.tools.MusicSpecifier(),
101                voice_name='Violin 1 Bowing Voice',
102                ),
103            consort.tools.PerformedTimespan(
104                start_offset=durationtools.Offset(7, 4),
105                stop_offset=durationtools.Offset(2, 1),
106                layer=1,
107                musicSpecifier=consort.tools.MusicSpecifier(),
108                voice_name='Viola Bowing Voice',
109                ),
110            consort.tools.PerformedTimespan(
111                start_offset=durationtools.Offset(7, 4),
112                stop_offset=durationtools.Offset(2, 1),
113                layer=1,

```

```

114     musicSpecifier=consort.tools.MusicSpecifier(),
115     voice_name='Violin 2 Bowing Voice',
116     ),
117   ],
118 )
119 :::
120 :::
121 >>> red_setting = new(
122 ...     red_setting,
123 ...     silenced_contexts=[
124 ...       'viola_lh',
125 ...       'cello',
126 ...     ],
127 ...   )
128 ...
129 >>> timespan_inventory = red_setting(
130 ...     layer=layer,
131 ...     score_template=score_template,
132 ...     segment_timespan=segment_timespan,
133 ...   )
134 >>> print(format(timespan_inventory))
135 timespantools.TimespanInventory(
136 [
137     consort.tools.PerformedTimespan(
138       start_offset=durationtools.Offset(1, 1),
139       stop_offset=durationtools.Offset(5, 4),
140       layer=1,
141       musicSpecifier=consort.tools.MusicSpecifier(),
142       voice_name='Violin 1 Bowing Voice',
143       ),
144     consort.tools.PerformedTimespan(
145       start_offset=durationtools.Offset(1, 1),
146       stop_offset=durationtools.Offset(3, 2),
147       layer=1,
148       musicSpecifier=consort.tools.MusicSpecifier(),
149       voice_name='Viola Bowing Voice',
150       ),
151     consort.tools.SilentTimespan(
152       start_offset=durationtools.Offset(1, 1),
153       stop_offset=durationtools.Offset(2, 1),
154       layer=1,
155       voice_name='Cello Bowing Voice',
156       ),
157     consort.tools.SilentTimespan(
158       start_offset=durationtools.Offset(1, 1),
159       stop_offset=durationtools.Offset(2, 1),
160       layer=1,
161       voice_name='Cello Fingering Voice',
162       ),
163     consort.tools.SilentTimespan(
164       start_offset=durationtools.Offset(1, 1),
165       stop_offset=durationtools.Offset(2, 1),
166       layer=1,
167       voice_name='Viola Fingering Voice',

```

```

168         ),
169         consort.tools.PerformedTimespan(
170             start_offset=durationtools.Offset(5, 4),
171             stop_offset=durationtools.Offset(3, 2),
172             layer=1,
173             music_specifier=consort.tools.MusicSpecifier(),
174             voice_name='Violin 2 Bowing Voice',
175             ),
176         consort.tools.PerformedTimespan(
177             start_offset=durationtools.Offset(3, 2),
178             stop_offset=durationtools.Offset(2, 1),
179             layer=1,
180             music_specifier=consort.tools.MusicSpecifier(),
181             voice_name='Violin 1 Bowing Voice',
182             ),
183         consort.tools.PerformedTimespan(
184             start_offset=durationtools.Offset(7, 4),
185             stop_offset=durationtools.Offset(2, 1),
186             layer=1,
187             music_specifier=consort.tools.MusicSpecifier(),
188             voice_name='Viola Bowing Voice',
189             ),
190         consort.tools.PerformedTimespan(
191             start_offset=durationtools.Offset(7, 4),
192             stop_offset=durationtools.Offset(2, 1),
193             layer=1,
194             music_specifier=consort.tools.MusicSpecifier(),
195             voice_name='Violin 2 Bowing Voice',
196             ),
197         ],
198     )
199     """
200     """
201
202     ### CLASS VARIABLES ###
203
204     __slots__ = (
205         '_music_specifiers',
206         '_silenced_contexts',
207         '_timespan_identifier',
208         '_timespan_maker',
209     )
210
211     ### INITIALIZER ###
212
213     def __init__(
214         self,
215         timespan_identifier=None,
216         timespan_maker=None,
217         silenced_contexts=None,
218         **music_specifiers
219     ):
220         import consort
221         prototype =

```

```

222     consort.CompositeMusicSpecifier,
223     consort.MusicSpecifier,
224     consort.MusicSpecifierSequence,
225     str, # for demonstration purposes only
226     type(None),
227 )
228 for abbreviation, music_specifier in sorted(music_specifiers.items()):
229     if isinstance(music_specifier, prototype):
230         continue
231     elif isinstance(music_specifier, collections.Sequence) and \
232         all(isinstance(x, prototype) for x in music_specifier):
233         music_specifier = consort.MusicSpecifierSequence(
234             music_specifiers=music_specifier,
235         )
236         music_specifiers[abbreviation] = music_specifier
237     else:
238         raise ValueError(music_specifier)
239 self._music_specifiers = music_specifiers
240 if silenced_contexts is not None:
241     silenced_contexts = (str(_) for _ in silenced_contexts)
242     silenced_contexts = set(silenced_contexts)
243 self._silenced_contexts = silenced_contexts
244 if timespan_identifier is not None:
245     prototype = (
246         timespantools.Timespan,
247         timespantools.TimespanInventory,
248         consort.RatioPartsExpression,
249     )
250     if not isinstance(timespan_identifier, prototype):
251         timespan_identifier = \
252             consort.RatioPartsExpression.from_sequence(
253                 timespan_identifier)
254     assert isinstance(timespan_identifier, prototype)
255 self._timespan_identifier = timespan_identifier
256 if timespan_maker is not None:
257     assert isinstance(timespan_maker,
258         consort.TimespanMaker), \
259     timespan_maker
260 else:
261     timespan_maker = consort.FloodedTimespanMaker()
262 self._timespan_maker = timespan_maker
263
264 ### SPECIAL METHODS ###
265
266 def __call__(
267     self,
268     layer=None,
269     score=None,
270     score_template=None,
271     segment_timespan=None,
272     timespan_inventory=None,
273     timespan_quantization=None,
274 ):
275     if score is None:

```

```

276         score = score_template()
277     if timespan_inventory is None:
278         timespan_inventory = timespantools.TimespanInventory()
279     if not self.music_specifiers:
280         return timespan_inventory
281     music_specifiers = self.resolve_music_specifiers(
282         score_template,
283         score=score,
284     )
285     silenced_context_names = self.resolve_silenced_contexts(
286         score_template,
287         score=score,
288     )
289     target_timeespans = self.resolve_target_timeespans(
290         segment_timespan,
291         timespan_quantization,
292     )
293     for i, target_timespan in enumerate(target_timeespans):
294         timespan_maker = self.timespan_maker.rotate(i)
295         timespan_inventory = timespan_maker(
296             layer=layer,
297             music_specifiers=music_specifiers,
298             silenced_context_names=silenced_context_names,
299             target_timespan=target_timespan,
300             timespan_inventory=timespan_inventory,
301         )
302     return timespan_inventory
303
304     def __getattr__(self, item):
305         if item in self.music_specifiers:
306             return self.music_specifiers[item]
307         return object.__getattribute__(self, item)
308
309     ### PRIVATE PROPERTIES ###
310
311     @property
312     def _storage_format_specification(self):
313         manager = systemtools.StorageFormatManager
314         keyword_argument_names = manager.get_keyword_argument_names(self)
315         keyword_argument_names = list(keyword_argument_names)
316         keyword_argument_names.extend(sorted(self.music_specifiers))
317         return systemtools.StorageFormatSpecification(
318             self,
319             keyword_argument_names=keyword_argument_names
320         )
321
322     ### PUBLIC METHODS ###
323
324     def resolve_music_specifiers(
325         self,
326         score_template,
327         score=None,
328     ):
329         import consort

```

```

330     assert score_template is not None
331     if score is None:
332         score = score_template()
333     all_abbreviations = score_template.context_name_abbreviations
334     prototype = (
335         consort.CompositeMusicSpecifier,
336         consort.MusicSpecifierSequence,
337     )
338     triples = []
339     for abbreviation, music_specifier in self.music_specifiers.items():
340         if not isinstance(music_specifier, prototype):
341             music_specifier = consort.MusicSpecifierSequence(
342                 music_specifiers=music_specifier,
343             )
344             context_name = all_abbreviations[abbreviation]
345             context = score[context_name]
346             context_index = inspect_(context).get_parentage().score_index
347             context_name = context.name
348             if isinstance(music_specifier, consort.CompositeMusicSpecifier):
349                 composite_pairs = score_template.composite_context_pairs
350                 one, two = composite_pairs[abbreviation]
351                 primary_voice_name = all_abbreviations[one]
352                 secondary_voice_name = all_abbreviations[two]
353                 music_specifier = new(
354                     music_specifier,
355                     primary_voice_name=primary_voice_name,
356                     secondary_voice_name=secondary_voice_name,
357                 )
358                 triple = (
359                     context_index,
360                     context_name,
361                     music_specifier,
362                 )
363                 triples.append(triple)
364     triples.sort(key=lambda x: x[0])
365     music_specifiers = collections.OrderedDict()
366     for context_index, context_name, music_specifier in triples:
367         music_specifiers[context_name] = music_specifier
368     return music_specifiers
369
370 def resolve_silenced_contexts(
371     self,
372     score_template,
373     score=None,
374 ):
375     assert score_template is not None
376     if score is None:
377         score = score_template()
378     all_abbreviations = score_template.context_name_abbreviations
379     composite_pairs = getattr(
380         score_template,
381         'composite_context_pairs',
382         {},
383     )

```

```

384     silenced_context_names = set()
385     silenced_contexts = self.silenced_contexts or {}
386     for abbreviation in silenced_contexts:
387         if abbreviation in composite_pairs:
388             one, two = composite_pairs[abbreviation]
389             primary_voice_name = all_abbreviations[one]
390             secondary_voice_name = all_abbreviations[two]
391             silenced_context_names.add(primary_voice_name)
392             silenced_context_names.add(secondary_voice_name)
393         elif abbreviation in all_abbreviations:
394             context_name = all_abbreviations[abbreviation]
395             silenced_context_names.add(context_name)
396         else:
397             message = 'Unresolvable context abbreviation: {}'
398             message = message.format(abbreviation)
399             raise Exception(message)
400     return silenced_context_names
401
402 def resolve_target_timepans(
403     self,
404     segment_timespan,
405     timespan_quantization=None,
406 ):
407     import consort
408     assert isinstance(segment_timespan, timespantools.Timespan)
409     timespan_identifier = self.timespan_identifier
410     if timespan_identifier is None:
411         target_timepans = timespantools.TimespanInventory([
412             segment_timespan,
413         ])
414     elif isinstance(self.timespan_identifier, timespantools.Timespan):
415         if timespan_identifier.stop_offset == Infinity:
416             timespan_identifier = new(
417                 timespan_identifier,
418                 stop_offset=segment_timespan.stop_offset,
419             )
420         segment_timespan = timespantools.Timespan(start_offset=0)
421         target_timepans = segment_timespan & timespan_identifier
422     else:
423         if isinstance(timespan_identifier, consort.RatioPartsExpression):
424             mask_timepans = timespan_identifier(segment_timespan)
425         else:
426             mask_timepans = timespan_identifier
427         target_timepans = timespantools.TimespanInventory()
428         for mask_timespan in mask_timepans:
429             available_timepans = segment_timespan & mask_timespan
430             target_timepans.extend(available_timepans)
431     if timespan_quantization is not None:
432         target_timepans.round_offsets(
433             timespan_quantization,
434             must_be_well_formed=True,
435         )
436     return target_timepans
437

```

```

438     """ PUBLIC PROPERTIES """
439
440     @property
441     def music_specifiers(self):
442         return self._music_specifiers
443
444     @property
445     def silenced_contexts(self):
446         return self._silenced_contexts
447
448     @property
449     def timespan_identifier(self):
450         return self._timespan_identifier
451
452     @property
453     def timespan_maker(self):
454         return self._timespan_maker

```

## A.28 CONSORTTOOLS.MUSICSPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 import os
4 from abjad import new
5 from abjad.tools import durationtools
6 from abjad.tools import indicatortools
7 from abjad.tools import instrumenttools
8 from abjad.tools import lilypondfiletools
9 from abjad.tools import markuptools
10 from abjad.tools import pitchtools
11 from abjad.tools import rhythmmakertools
12 from abjad.tools import stringtools
13 from consort.tools.HashCachingObject import HashCachingObject
14
15
16 class MusicSpecifier(HashCachingObject):
17     r'''A music specifier.
18
19     :::
20
21     >>> import consort
22     >>> musicSpecifier = consort.MusicSpecifier()
23     >>> print(format(musicSpecifier))
24     consort.tools.MusicSpecifier()
25
26     .. container:: example
27
28     MusicSpecifier can accept CompositeRhythmMakers in their 'rhythm_maker'
29     slot:
30
31     :::
32
33     >>> musicSpecifier = consort.MusicSpecifier(
34             ...      rhythm_maker=consort.CompositeRhythmMaker(),

```

```

35         ...
36
37     '',
38
39     ### CLASS VARIABLES ###
40
41     __slots__ = (
42         '_attachment_handler',
43         '_color',
44         '_grace_handler',
45         '_instrument',
46         '_labels',
47         '_minimum_phrase_duration',
48         '_pitch_handler',
49         '_rhythm_maker',
50         '_seed',
51     )
52
53     ### INITIALIZER ###
54
55     def __init__(
56         self,
57         attachment_handler=None,
58         color=None,
59         grace_handler=None,
60         instrument=None,
61         labels=None,
62         minimum_phrase_duration=None,
63         pitch_handler=None,
64         rhythm_maker=None,
65         seed=None,
66     ):
67         import consort
68         HashCachingObject.__init__(self)
69         if attachment_handler is not None:
70             assert isinstance(attachment_handler, consort.AttachmentHandler)
71         self._attachment_handler = attachment_handler
72         if color is not None:
73             color = str(color)
74         self._color = color
75         if grace_handler is not None:
76             assert isinstance(grace_handler, consort.GraceHandler)
77         self._grace_handler = grace_handler
78         if instrument is not None:
79             assert isinstance(instrument, instrumenttools.Instrument)
80         self._instrument = instrument
81         if labels is not None:
82             if isinstance(labels, str):
83                 labels = (labels,)
84             labels = tuple(str(_) for _ in labels)
85         self._labels = labels
86         if minimum_phrase_duration is not None:
87             minimum_phrase_duration = \
88                 durationtools.Duration(minimum_phrase_duration)

```

```

89         assert 0 <= minimum_phrase_duration
90     self._minimum_phrase_duration = minimum_phrase_duration
91     if pitch_handler is not None:
92         assert isinstance(pitch_handler, consort.PitchHandler)
93     self._pitch_handler = pitch_handler
94     if rhythm_maker is not None:
95         prototype = (
96             rhythmmakertools.RhythmMaker,
97             consort.CompositeRhythmMaker,
98         )
99         assert isinstance(rhythm_maker, prototype)
100    self._rhythm_maker = rhythm_maker
101    if seed is not None:
102        seed = int(seed)
103    self._seed = seed
104
105    ### SPECIAL METHODS ###
106
107    def __illustrate__(
108        self,
109        annotate=False,
110        verbose=True,
111        package_name=None,
112        **kwargs
113    ):
114        """Illustrates music specifier.
115
116        :::
117
118        >>> piano_glissando_music_specifier = consort.MusicSpecifier(
119            ...     attachment_handler=consort.AttachmentHandler(
120            ...         glissando=spannertools.Glissando(),
121            ...         ),
122            ...     color=None,
123            ...     labels=[],
124            ...     pitch_handler=consort.AbsolutePitchHandler(
125            ...         pitchSpecifier="c' f' c' f' c''' c'' c' c'''",
126            ...         ),
127            ...     rhythm_maker=consort.CompositeRhythmMaker(
128            ...         last=rhythmmakertools.IncisedRhythmMaker(
129            ...             inciseSpecifier=rhythmmakertools.InciseSpecifier(
130            ...                 prefixCounts=[0],
131            ...                 suffixTalea=[1],
132            ...                 suffixCounts=[1],
133            ...                 taleaDenominator=16,
134            ...                 ),
135            ...             ),
136            ...             default=rhythmmakertools.EvenDivisionRhythmMaker(
137            ...                 denominators=(4,),
138            ...                 durationSpellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
139            ...                     decreaseDurationsMonotonically=True,
140            ...                     forbiddenWrittenDuration=(1, 4),
141            ...                     forbidBidirectionalRewriting=True,
142            ...             ),

```

```

143         ... ),
144         ... ),
145         ...
146         >>> illustration = piano_glissando_music_specifier.__illustrate__(  

147             ... annotate=True,  

148             ... verbose=False,  

149             ... )  

150  

151     Returns LilyPond file.  

152     """  

153     import consort  

154     score_template = consort.StringQuartetScoreTemplate(  

155         split=False,  

156         without_instruments=True,  

157         )  

158     segment_maker = consort.SegmentMaker(  

159         desired_duration_in_seconds=10,  

160         discard_final_silence=True,  

161         #annotate_phrasing=True,  

162         annotate_phrasing=annotate,  

163         permitted_time_signatures=[  

164             (3, 8),  

165             (4, 8),  

166             (5, 8),  

167             (6, 8),  

168             (7, 8),  

169             ],  

170         score_template=score_template,  

171         tempo=indicatortools.Tempo((1, 4), 72),  

172         timespan_quantization=(1, 8),  

173         )  

174     timespan_maker = consort.TaleaTimespanMaker(  

175         initial_silence_talea=rhythmmakertools.Talea(  

176             counts=[0, 2, 1],  

177             denominator=8,  

178             ),  

179         playing_talea=rhythmmakertools.Talea(  

180             counts=[3, 2, 1, 4, 5, 3, 1, 2],  

181             denominator=8,  

182             ),  

183         playing_groupings=[2, 1, 2, 3, 1, 3, 4, 1, 2, 3],  

184         repeat=True,  

185         silence_talea=rhythmmakertools.Talea(  

186             counts=[1, 2, 3, 1, 2, 4],  

187             denominator=8,  

188             ),  

189         step_anchor=Right,  

190         synchronize_groupings=False,  

191         synchronize_step=False,  

192         timespanSpecifier=consort.TimespanSpecifier(  

193             minimum_duration=durationtools.Duration(1, 8),  

194             ),  

195         )  

196     segment_maker.add_setting(

```

```

197     timespan_maker=timespan_maker,
198     violin_1=self,
199     violin_2=self,
200     viola=self,
201     cello=self,
202     )
203 segment_metadata = collections.OrderedDict(
204     segment_count=1,
205     segment_number=1,
206     )
207 lilypond_file, segment_metadata = segment_maker(
208     segment_metadata=segment_metadata,
209     verbose=verbose,
210     )
211 consort_stylesheet_path = os.path.join(
212     consort.__path__[0],
213     'stylesheets',
214     'stylesheet.ily',
215     )
216 consort_stylesheet_path = os.path.abspath(consort_stylesheet_path)
217 lilypond_file.file_initial_user_includes[:] = [consort_stylesheet_path]
218 lilypond_file.use_relative_includes = False
219 if package_name is not None:
220     header = lilypondfiletools.Block('header')
221     header.title = stringtools.to_space_delimited_lowercase(
222         package_name).title()
223     header.tagline = markuptools.Markup('""')
224     lilypond_file.items.insert(0, header)
225 return lilypond_file
226
227 ### PUBLIC METHODS ###
228
229 def rotate(self, rotation):
230     seed = self.seed or 0
231     seed = seed + rotation
232     return new(self, seed=seed)
233
234 def transpose(self, expr):
235     r'''Transposes music specifier.
236
237     :::
238
239     >>> musicSpecifier = consort.MusicSpecifier(
240     ...     pitch_handler=consort.AbsolutePitchHandler(
241     ...         pitchSpecifier = consort.PitchSpecifier(
242     ...             pitchSegments=(
243     ...                 "c' e' g'", "fs' gs'", "b",
244     ...                 ),
245     ...             ratio=(1, 2, 3),
246     ...             ),
247     ...             ),
248     ...             ),
249     ...         )
250

```

```

251     >>> transposed_music_specifier = musicSpecifier.transpose('M2')
252     >>> print(format(transposed_musicSpecifier))
253     consort.tools.MusicSpecifier(
254         pitch_handler=consort.tools.AbsolutePitchHandler(
255             pitchSpecifier=consort.tools.PitchSpecifier(
256                 pitch_segments=(
257                     pitchtools.PitchSegment(
258                         (
259                             pitchtools.NamedPitch('bf'),
260                             pitchtools.NamedPitch("d'"),
261                             pitchtools.NamedPitch("f'"),
262                             ),
263                             item_class=pitchtools.NamedPitch,
264                             ),
265                             pitchtools.PitchSegment(
266                                 (
267                                     pitchtools.NamedPitch("e'"),
268                                     pitchtools.NamedPitch("fs'"),
269                                     ),
270                                     item_class=pitchtools.NamedPitch,
271                                     ),
272                                     pitchtools.PitchSegment(
273                                         (
274                                             pitchtools.NamedPitch('a'),
275                                             ),
276                                             item_class=pitchtools.NamedPitch,
277                                             ),
278                                             ),
279                                             ratio=mathtools.Ratio((1, 2, 3)),
280                                             ),
281                                         ),
282                                         )
283
284     Returns new music specifier.
285     """
286     if isinstance(expr, str):
287         try:
288             pitch = pitchtools.NamedPitch(expr)
289             expr = pitchtools.NamedPitch('C4') - pitch
290         except:
291             expr = pitchtools.NamedInterval(expr)
292         pitch_handler = self.pitch_handler
293         if pitch_handler is not None:
294             pitch_handler = pitch_handler.transpose(expr)
295     return new(
296         self,
297         pitch_handler=pitch_handler,
298         )
299
300     ### PUBLIC PROPERTIES ###
301
302     @property
303     def attachment_handler(self):
304         return self._attachment_handler

```

```

305
306     @property
307     def color(self):
308         return self._color
309
310     @property
311     def grace_handler(self):
312         return self._grace_handler
313
314     @property
315     def instrument(self):
316         return self._instrument
317
318     @property
319     def labels(self):
320         return self._labels
321
322     @property
323     def minimum_phrase_duration(self):
324         return self._minimum_phrase_duration
325
326     @property
327     def pitch_handler(self):
328         return self._pitch_handler
329
330     @property
331     def rhythm_maker(self):
332         return self._rhythm_maker
333
334     @property
335     def seed(self):
336         return self._seed

```

## A.29 CONSORTTOOLS.MUSICSPECIFIERSEQUENCE

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad import new
4 from abjad.tools import abctools
5 from abjad.tools import datastructuretools
6 from abjad.tools import mathtools
7 from abjad.tools import rhythmmakertools
8 from abjad.tools import sequencetools
9 from abjad.tools import timespantools
10
11
12 class MusicSpecifierSequence(abctools.AbjadValueObject):
13     r'''A music specifier sequence.
14
15     :::
16
17     >>> import consort
18     >>> sequence_a = consort.MusicSpecifierSequence(
19             ...      music_specifiers='music',

```

```

20     ...
21     )
22     >>> print(format(sequence_a))
23     consort.tools.MusicSpecifierSequence(
24         music_specifiers=datatoolstools.CyclicTuple(
25             ['music']
26         ),
27     )
28 :::
29
30     >>> sequence_b = consort.MusicSpecifierSequence(
31         ...
32             application_rate='phrase',
33             ...
34             music_specifiers=['one', 'two', 'three'],
35             ...
36         )
37     >>> print(format(sequence_b))
38     consort.tools.MusicSpecifierSequence(
39         application_rate='phrase',
40         music_specifiers=datatoolstools.CyclicTuple(
41             ['one', 'two', 'three']
42         ),
43     )
44     ''
45
46     """ CLASS VARIABLES """
47
48     __slots__ = (
49         '_application_rate',
50         '_music_specifiers',
51     )
52
53     """ INITIALIZER """
54
55     def __init__(
56         self,
57         application_rate=None,
58         music_specifiers=None,
59     ):
60         if application_rate is not None:
61             application_rate = application_rate or 'phrase'
62             assert application_rate in ('division', 'phrase')
63         if music_specifiers is not None:
64             if not isinstance(music_specifiers, collections.Sequence) or \
65                 isinstance(music_specifiers, str):
66                 music_specifiers = [music_specifiers]
67             music_specifiers = tuple(music_specifiers)
68             music_specifiers = datatoolstools.CyclicTuple(music_specifiers)
69             assert len(music_specifiers)
70         self._application_rate = application_rate
71         self._music_specifiers = music_specifiers
72
73     """ SPECIAL METHODS """
74
75     def __call__(


```

```

74     self,
75     durations=None,
76     layer=None,
77     output_masks=None,
78     padding=None,
79     seed=None,
80     start_offset=None,
81     timespanSpecifier=None,
82     voice_name=None,
83     ):
84     import consort
85     timespans = timespantools.TimespanInventory()
86     timespanSpecifier = timespanSpecifier or \
87         consort.TimespanSpecifier()
88     seed = seed or 0
89     durations = [_ for _ in durations if _]
90     offsets = mathtools.cumulative_sums(durations, start_offset)
91     if not offsets:
92         return timespans
93
94     offset_pair_count = len(offsets) - 1
95     output_mask_prototype = (
96         type(None),
97         rhythmmakertools.SustainMask,
98         )
99     iterator = sequencetools.iterate_sequence_nwise(offsets)
100    for i, offset_pair in enumerate(iterator):
101        start_offset, stop_offset = offset_pair
102        musicSpecifier = self[seed]
103        timespan = consort.PerformedTimespan(
104            forbid_fusing=timespanSpecifier.forbid_fusing,
105            forbid_splitting=timespanSpecifier.forbid_splitting,
106            layer=layer,
107            minimum_duration=timespanSpecifier.minimum_duration,
108            musicSpecifier=musicSpecifier,
109            start_offset=start_offset,
110            stop_offset=stop_offset,
111            voice_name=voice_name,
112            )
113        if not output_masks:
114            timespans.append(timespan)
115        else:
116            output_mask = output_masks.get_matching_pattern(
117                i, offset_pair_count, rotation=seed)
118            if isinstance(output_mask, output_mask_prototype):
119                timespans.append(timespan)
120            if self.application_rate == 'division':
121                seed += 1
122
123    if padding:
124        silent_timespans = timespantools.TimespanInventory()
125        for shard in timespans.partition(True):
126            silent_timespan_one = consort.SilentTimespan(
127                layer=layer,

```

```

128         start_offset=shard.start_offset - padding,
129         stop_offset=shard.start_offset,
130         voice_name=voice_name,
131         )
132     silent_timepasses.append(silent_timepass_one)
133     silent_timepass_two = consort.SilentTimepass(
134         layer=layer,
135         start_offset=shard.stop_offset,
136         stop_offset=shard.stop_offset + padding,
137         voice_name=voice_name,
138         )
139     silent_timepasses.append(silent_timepass_two)
140     silent_timepasses.compute_logical_or()
141     for timepass in timepasses:
142         silent_timepasses -= timepass
143     timepasses.extend(silent_timepasses)
144     timepasses.sort()
145
146     return timepasses
147
148 def __getitem__(self, item):
149     return self._music_specifiers[item]
150
151 def __len__(self):
152     return len(self._music_specifiers)
153
154 ### PUBLIC METHODS ###
155
156 def transpose(self, expr):
157     music_specifiers = [_.transpose(expr) for _ in self.music_specifiers]
158     return new(
159         self,
160         music_specifiers=music_specifiers,
161         )
162
163 ### PUBLIC PROPERTIES ###
164
165 @property
166 def application_rate(self):
167     return self._application_rate
168
169 @property
170 def music_specifiers(self):
171     return self._music_specifiers

```

## A.30 CONSORTTOOLS.OCTAVATIONEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import attach
4 from abjad import inspect_
5 from abjad.tools import abctools
6 from abjad.tools import indicatortools
7 from abjad.tools import pitchtools

```

```

8 from abjad.tools import scoretools
9 from abjad.tools import spannertools
10
11
12 class OctavationExpression(abctools.AbjadValueObject):
13     r'''An octavation expression.
14     '''
15
16     ### CLASS VARIABLES ###
17
18     __slots__ = ()
19
20     ### INITIALIZER ###
21
22     def __init__(self):
23         pass
24
25     ### SPECIAL METHODS ###
26
27     def __call__(self, music, name=None):
28         leaves = music.select_leaves()
29         weights = []
30         weighted_pitches = []
31         for leaf in leaves:
32             weight = float(inspect_(leaf).get_duration())
33             if isinstance(leaf, scoretools.Note):
34                 pitch = float(leaf.written_pitch)
35                 weighted_pitch = pitch * weight
36                 weights.append(weight)
37                 weighted_pitches.append(weighted_pitch)
38             elif isinstance(leaf, scoretools.Chord):
39                 for pitch in leaf.written_pitches:
40                     pitch = float(pitch)
41                     weighted_pitch = pitch * weight
42                     weighted_pitches.append(weighted_pitch)
43                     weights.append(weight)
44             sum_of_weights = sum(weights)
45             sum_of_weighted_pitches = sum(weighted_pitches)
46             weighted_average = sum_of_weighted_pitches / sum_of_weights
47             #print(music, weighted_average)
48             clef = inspect_(leaves[0]).get_effective(indicatortools.Clef)
49             octavation_spanner = None
50             if clef == indicatortools.Clef('treble'):
51                 if int(pitchtools.NamedPitch('C6')) <= int(weighted_average):
52                     octavation_spanner = spannertools.OctavationSpanner()
53             elif clef == indicatortools.Clef('bass'):
54                 pass
55             if octavation_spanner is not None:
56                 attach(octavation_spanner, music)

```

### A.31 CONSORTTOOLS.PERFORMEDTIMESPAN

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import durationtools

```

```

3 from abjad.tools import markuptools
4 from abjad.tools import mathtools
5 from abjad.tools import timespantools
6
7
8 class PerformedTimespan(timespantools.Timespan):
9     r'''A Consort timespan.
10
11     :::
12
13         >>> import consort
14         >>> timespan = consort.PerformedTimespan()
15         >>> print(format(timespan))
16         consort.tools.PerformedTimespan(
17             start_offset=NegativeInfinity,
18             stop_offset=Infinity,
19             )
20
21     '''
22
23     ### CLASS VARIABLES ###
24
25     __slots__ = (
26         '_forbid_fusing',
27         '_forbid_splitting',
28         '_divisions',
29         '_layer',
30         '_minimum_duration',
31         '_music',
32         '_musicSpecifier',
33         '_original_start_offset',
34         '_original_stop_offset',
35         '_voice_name',
36         )
37
38     ### INITIALIZER ###
39
40     def __init__(
41         self,
42         start_offset=mathtools.NegativeInfinity(),
43         stop_offset=mathtools.Infinity(),
44         divisions=None,
45         forbid_fusing=None,
46         forbid_splitting=None,
47         layer=None,
48         minimum_duration=None,
49         music=None,
50         musicSpecifier=None,
51         original_start_offset=None,
52         original_stop_offset=None,
53         voice_name=None,
54         ):
55         timespantools.Timespan.__init__(
56             self,

```

```

57         start_offset=start_offset,
58         stop_offset=stop_offset,
59     )
60     if divisions is not None:
61         divisions = tuple(durationtools.Duration(_) for _ in divisions)
62         assert sum(divisions) == self.duration
63     self._divisions = divisions
64     if forbid_fusing is not None:
65         forbid_fusing = bool(forbid_fusing)
66     self._forbid_fusing = forbid_fusing
67     if forbid_splitting is not None:
68         forbid_splitting = bool(forbid_splitting)
69     self._forbid_splitting = forbid_splitting
70     if layer is not None:
71         layer = int(layer)
72     self._layer = layer
73     if minimum_duration is not None:
74         minimum_duration = durationtools.Duration(minimum_duration)
75     self._minimum_duration = minimum_duration
76     #if music is not None:
77     #    assert inspect_(music).get_duration() == self.duration
78     self._music = music
79     #if music_specifier is not None:
80     #    assert isinstance(music_specifier, consort.MusicSpecifier), \
81     #        music_specifier
82     self._music_specifier = music_specifier
83     if original_start_offset is not None:
84         original_start_offset = durationtools.Offset(original_start_offset)
85     else:
86         original_start_offset = self.start_offset
87     self._original_start_offset = original_start_offset
88     if original_stop_offset is not None:
89         original_stop_offset = durationtools.Offset(original_stop_offset)
90     else:
91         original_stop_offset = self.stop_offset
92     self._original_stop_offset = original_stop_offset
93     self._voice_name = voice_name
94
95     ### PRIVATE PROPERTIES ###
96
97     @property
98     def _storage_format_specification(self):
99         from abjad.tools import systemtools
100        manager = systemtools.StorageFormatManager
101        keyword_argument_names = list(manager.get_keyword_argument_names(self))
102        if self.original_start_offset == self.start_offset:
103            keyword_argument_names.remove('original_start_offset')
104        if self.original_stop_offset == self.stop_offset:
105            keyword_argument_names.remove('original_stop_offset')
106        return systemtools.StorageFormatSpecification(
107            self,
108            keyword_argument_names=keyword_argument_names,
109        )
110

```

```

111     """ SPECIAL METHODS """
112
113     def __lt__(self, expr):
114         if timespantools.Timespan.__lt__(self, expr):
115             return True
116         if not timespantools.Timespan.__gt__(self, expr):
117             if hasattr(expr, 'voice_name'):
118                 return self.voice_name < expr.voice_name
119             return False
120
121     """ PRIVATE METHODS """
122
123     def _as_postscript(
124         self,
125         postscript_x_offset,
126         postscript_y_offset,
127         postscript_scale,
128     ):
129         start = (float(self.start_offset) * postscript_scale)
130         start -= postscript_x_offset
131         stop = (float(self.stop_offset) * postscript_scale)
132         stop -= postscript_x_offset
133         ps = markuptools.Postscript()
134         ps = ps.moveto(start, postscript_y_offset)
135         ps = ps.lineto(stop, postscript_y_offset)
136         ps = ps.stroke()
137         ps = ps.moveto(start, postscript_y_offset + 0.75)
138         ps = ps.lineto(start, postscript_y_offset - 0.75)
139         ps = ps.stroke()
140         ps = ps.moveto(stop, postscript_y_offset + 0.75)
141         ps = ps.lineto(stop, postscript_y_offset - 0.75)
142         ps = ps.stroke()
143         if self.layer is not None:
144             ps = ps.moveto(start, postscript_y_offset)
145             ps = ps.rmoveto(0.25, 0.5)
146             #ps = ps.scale(0.8, 0.8)
147             ps = ps.show(str(self.layer))
148             #ps = ps.scale(1.25, 1.25)
149         return ps
150
151     """ PUBLIC PROPERTIES """
152
153     @property
154     def divisions(self):
155         return self._divisions
156
157     @property
158     def forbid_fusing(self):
159         return self._forbid_fusing
160
161     @property
162     def forbid_splitting(self):
163         return self._forbid_splitting
164

```

```

165     @property
166     def is_left_broken(self):
167         if self.original_start_offset is not None:
168             if self.original_start_offset != self.start_offset:
169                 return True
170             return False
171
172     @property
173     def is_right_broken(self):
174         if self.original_stop_offset is not None:
175             if self.original_stop_offset != self.stop_offset:
176                 return True
177             return False
178
179     @property
180     def layer(self):
181         return self._layer
182
183     @property
184     def minimum_duration(self):
185         return self._minimum_duration
186
187     @property
188     def music(self):
189         return self._music
190
191     @property
192     def music_specifier(self):
193         return self._music_specifier
194
195     @property
196     def original_start_offset(self):
197         return self._original_start_offset
198
199     @property
200     def original_stop_offset(self):
201         return self._original_stop_offset
202
203     @property
204     def voice_name(self):
205         return self._voice_name

```

## A.32 CONSORTTOOLS.PHRASEDSELECTORCALLBACK

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3 from abjad.tools import durationtools
4 from abjad.tools import selectiontools
5
6
7 class PhrasedSelectorCallback(abctools.AbjadValueObject):
8
9     ### CLASS VARIABLES ###
10

```

```

11     __slots__ = ()
12
13     ### SPECIAL METHODS ###
14
15     def __call__(self, expr):
16         assert isinstance(expr, tuple), repr(tuple)
17         result = []
18         for subexpr in expr:
19             subresult = []
20             for division in subexpr[:-1]:
21                 leaf = division.select_leaves()[0]
22                 selection = selectiontools.Selection(leaf)
23                 subresult.append(selection)
24             leaves = subexpr[-1].select_leaves()
25             if leaves.get_duration() <= durationtools.Duration(1, 8):
26                 subresult.append(leaves[-1])
27             elif 1 == len(leaves):
28                 subresult.append(leaves[0])
29             else:
30                 subresult.append(leaves[0])
31                 subresult.append(leaves[1])
32             subresult = selectiontools.Selection(subresult)
33             result.append(subresult)
34         result = tuple(result)
35     return result

```

### A.33 CONSORTTOOLS.PITCHCLASSPITCHHANDLER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import new
4 from abjad.tools import datastructuretools
5 from abjad.tools import pitchtools
6 from consort.tools.PitchHandler import PitchHandler
7
8
9 class PitchClassPitchHandler(PitchHandler):
10     r'''PitchClass pitch maker.
11
12     :::
13
14     >>> import consort
15     >>> pitch_handler = consort.PitchClassPitchHandler(
16     ...     pitchSpecifier="c' d' e' f",
17     ...     )
18     >>> print(format(pitch_handler))
19     consort.tools.PitchClassPitchHandler(
20         pitchSpecifier=consort.tools.PitchSpecifier(
21             pitchSegments=
22                 pitchtools.PitchSegment(
23                     (
24                         pitchtools.NamedPitch("c"),
25                         pitchtools.NamedPitch("d"),
26                         pitchtools.NamedPitch("e"),

```

```

27             pitchtools.NamedPitch("f'"),
28             ),
29             item_class=pitchtools.NamedPitch,
30             ),
31             ),
32             ratio=mathtools.Ratio((1,)),
33             ),
34         )
35
36     """
37
38     """ CLASS VARIABLES """
39
40     __slots__ = (
41         '_leap_constraint',
42         '_octavations',
43         '_pitch_range',
44         '_register_specifier',
45         '_register_spread',
46     )
47
48     _default_octavations = datastructuretools.CyclicTuple([
49         4, 2, 1, 0, 5, 6, 7, 3,
50         0, 5, 3, 1, 7, 4, 2, 6,
51         2, 1, 5, 3, 4, 0, 7, 6,
52         1, 2, 4, 0, 5, 7, 6, 3,
53         7, 0, 3, 1, 5, 4, 6, 2,
54         6, 1, 2, 0, 7, 5, 3, 4,
55         3, 0, 4, 7, 2, 5, 6, 1,
56         2, 3, 4, 7, 5, 1, 0, 6,
57     ])
58
59     """ INITIALIZER """
60
61     def __init__(
62         self,
63         deviations=None,
64         forbid_repetitions=None,
65         grace_expressions=None,
66         leap_constraint=None,
67         logical_tie_expressions=None,
68         octavations=None,
69         pitch_application_rate=None,
70         pitch_range=None,
71         register_specifier=None,
72         register_spread=None,
73         pitch_operationSpecifier=None,
74         pitchSpecifier=None,
75         pitches_are_nonsemantic=None,
76     ):
77         PitchHandler.__init__(
78             self,
79             deviations=deviations,
80             forbid_repetitions=forbid_repetitions,

```

```

81     grace_expressions=grace_expressions,
82     logical_tie_expressions=logical_tie_expressions,
83     pitch_application_rate=pitch_application_rate,
84     pitch_operationSpecifier=pitch_operationSpecifier,
85     pitchSpecifier=pitchSpecifier,
86     pitchesAreNonsemantic=pitchesAreNonsemantic,
87     )
88 self._initialize_leap_constraint(leap_constraint)
89 self._initialize_octavations(octavations)
90 self._initialize_pitch_range(pitch_range)
91 self._initialize_registerSpecifier(registerSpecifier)
92 self._initialize_registerSpread(registerSpread)

93
94     ### SPECIAL METHODS ###

95
96     def __call__(
97         self,
98         attack_point_signature,
99         logical_tie,
100        musicSpecifier,
101        pitchChoices,
102        previousPitch,
103        seedSession,
104        ):
105     previousPitchClass = pitchtools.NamedPitchClass(previousPitch)
106     instrument = self._get_instrument(logical_tie, musicSpecifier)
107     pitchRange = self._get_pitch_range(
108         instrument,
109         logical_tie,
110         )
111     registration = self._get_registration(
112         attack_point_signature,
113         logical_tie,
114         seedSession.currentTimewisePhraseSeed,
115         )
116     pitchClass = self._get_pitch_class(
117         attack_point_signature,
118         pitchChoices,
119         previousPitchClass,
120         seedSession.currentPhrasedVoicewiseLogicalTieSeed,
121         )
122     pitch = self._get_pitch(
123         pitchClass,
124         registration,
125         seedSession.currentPhrasedVoicewiseLogicalTieSeed,
126         )
127     pitch = self._constrain_interval(
128         pitch,
129         previousPitch,
130         )
131     pitch = self._apply_deviation(
132         pitch,
133         seedSession.currentUnphrasedVoicewiseLogicalTieSeed,
134         )

```

```

135     pitch_range = self.pitch_range or pitch_range
136     if pitch_range is not None:
137         pitch = self._fit_pitch_to_pitch_range(
138             pitch,
139             pitch_range,
140             )
141     return pitch
142
143     ### PRIVATE METHODS ###
144
145     def _constrain_interval(self, current_pitch, previous_pitch):
146         if previous_pitch is None or not self.leap_constraint:
147             return current_pitch
148         maximum_leap = self.leap_constraint.semitones
149         #semitones = float(current_pitch) - float(previous_pitch)
150         semitones = float(previous_pitch) - float(current_pitch)
151         if maximum_leap < semitones: # descent
152             current_pitch = current_pitch.transpose(12)
153         elif semitones < -maximum_leap: # ascent
154             current_pitch = current_pitch.transpose(-12)
155         return current_pitch
156
157     def _fit_pitch_to_pitch_range(self, pitch, pitch_range):
158         while pitch <= pitch_range.start_pitch and \
159             pitch not in pitch_range:
160             pitch = pitch.transpose(12)
161         while pitch_range.stop_pitch <= pitch and \
162             pitch not in pitch_range:
163             pitch = pitch.transpose(-12)
164         assert pitch in pitch_range, \
165             (pitch, pitch.octave_number, pitch_range)
166         return pitch
167
168     def _get_pitch(
169         self,
170         pitch_class,
171         registration,
172         seed,
173         ):
174         octavations = self.octavations or self._default_octavations
175         octave = octavations[seed]
176         pitch = pitchtools.NamedPitch(pitch_class, octave)
177         pitch_range = pitchtools.PitchRange('[C0, C8]')
178         pitch = self._fit_pitch_to_pitch_range(pitch, pitch_range)
179         pitch = registration([pitch])[0]
180         pitch = pitchtools.NamedPitch(pitch)
181     return pitch
182
183     def _get_pitch_class(
184         self,
185         attack_point_signature,
186         pitch_choices,
187         previous_pitch_class,
188         seed,

```

```

189     ):
190     pitch_class = pitch_choices[seed]
191     pitch_class = pitchtools.NamedPitchClass(pitch_class)
192     if pitch_choices and \
193         1 < len(set(pitch_choices)) and \
194         self.forbid_repetitions:
195         if self.pitch_application_rate == 'phrase':
196             if attack_point_signature.is_first_of_phrase:
197                 while float(pitch_class) == float(previous_pitch_class):
198                     seed += 1
199                     pitch_class = pitch_choices[seed]
200                     pitch_class = pitchtools.NamedPitchClass(pitch_class)
201             elif self.pitch_application_rate == 'division':
202                 if attack_point_signature.is_first_of_division:
203                     while float(pitch_class) == float(previous_pitch_class):
204                         seed += 1
205                         pitch_class = pitch_choices[seed]
206                         pitch_class = pitchtools.NamedPitchClass(pitch_class)
207             else:
208                 while float(pitch_class) == float(previous_pitch_class):
209                     seed += 1
210                     pitch_class = pitch_choices[seed]
211                     pitch_class = pitchtools.NamedPitchClass(pitch_class)
212     return pitch_class
213
214     def _get_registration(
215         self,
216         attack_point_signature,
217         logical_tie,
218         phrase_seed,
219     ):
220         import consort
221         registerSpecifier = self.registerSpecifier
222         if registerSpecifier is None:
223             registerSpecifier = consort.RegisterSpecifier()
224         register = registerSpecifier.find_register(
225             attack_point_signature,
226             seed=phrase_seed,
227         )
228         registerSpread = self.registerSpread
229         if registerSpread is None:
230             registerSpread = 6
231         registration = \
232             pitchtools.Registration([
233                 ('[C0, C4)', register),
234                 ('[C4, C8)', register + registerSpread),
235             ])
236     return registration
237
238     def _initialize_leap_constraint(self, leap_constraint):
239         if leap_constraint is not None:
240             leap_constraint = pitchtools.NumberedInterval(leap_constraint)
241             leap_constraint = abs(leap_constraint)
242         self._leap_constraint = leap_constraint

```

```

243     def _initialize_octavations(self, octavations):
244         if octavations is not None:
245             assert octavations
246             assert all(isinstance(x, int) for x in octavations)
247             octavations = datastructuretools.CyclicTuple(octavations)
248         self._octavations = octavations
249
250
251     def _initialize_pitch_classes(self, pitch_classes):
252         pitch_classes = pitchtools.PitchClassSegment(
253             items=pitch_classes,
254             item_class=pitchtools.NumberedPitchClass,
255         )
256         pitch_classes = datastructuretools.CyclicTuple(pitch_classes)
257         self._pitch_classes = pitch_classes
258
259     def _initialize_pitch_range(self, pitch_range):
260         if pitch_range is not None:
261             assert isinstance(pitch_range, pitchtools.PitchRange)
262         self._pitch_range = pitch_range
263
264     def _initialize_register_specifier(self, registerSpecifier):
265         import consort
266         if registerSpecifier is not None:
267             assert isinstance(registerSpecifier, consort.RegisterSpecifier)
268         self._registerSpecifier = registerSpecifier
269
270     def _initialize_register_spread(self, registerSpread):
271         if registerSpread is not None:
272             registerSpread = int(registerSpread)
273             assert 0 <= registerSpread < 12
274         self._registerSpread = registerSpread
275
276     ### PUBLIC METHODS ###
277
278     def transpose(self, expr):
279         pitchSpecifier = self.pitchSpecifier
280         if pitchSpecifier is not None:
281             pitchSpecifier = pitchSpecifier.transpose(expr)
282         registerSpecifier = self.registerSpecifier
283         if registerSpecifier is not None:
284             registerSpecifier = registerSpecifier.transpose(expr)
285         return new(
286             self,
287             pitchSpecifier=pitchSpecifier,
288             registerSpecifier=registerSpecifier,
289         )
290
291     ### PUBLIC PROPERTIES ###
292
293     @property
294     def leap_constraint(self):
295         return self._leap_constraint
296

```

```

297     @property
298     def octavations(self):
299         return self._octavations
300
301     @property
302     def pitch_range(self):
303         return self._pitch_range
304
305     @property
306     def register_specifier(self):
307         return self._register_specifier
308
309     @property
310     def register_spread(self):
311         return self._register_spread

```

## A.34 CONSORTTOOLS.PITCHHANDLER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import abc
4 import collections
5 from abjad import attach
6 from abjad import inspect_
7 from abjad import iterate
8 from abjad import new
9 from abjad.tools import datastructuretools
10 from abjad.tools import durationtools
11 from abjad.tools import instrumenttools
12 from abjad.tools import indicatortools
13 from abjad.tools import pitchtools
14 from abjad.tools import sequencetools
15 from abjad.tools import timespanools
16 from consort.tools.HashCachingObject import HashCachingObject
17
18
19 class PitchHandler(HashCachingObject):
20
21     ### CLASS VARIABLES ###
22
23     __slots__ = (
24         '_deviations',
25         '_forbid_repetitions',
26         '_grace_expressions',
27         '_logical_tie_expressions',
28         '_pitch_application_rate',
29         '_pitch_operationSpecifier',
30         '_pitchSpecifier',
31         '_pitches_are_nonsemantic',
32     )
33
34     ### INITIALIZER ###
35
36     def __init__(


```

```

37     self,
38     deviations=None,
39     forbid_repetitions=None,
40     grace_expressions=None,
41     logical_tie_expressions=None,
42     pitch_application_rate=None,
43     pitch_specifier=None,
44     pitch_operationSpecifier=None,
45     pitches_are_nonsemantic=None,
46     ):
47     HashCachingObject.__init__(self)
48     self._initialize_deviations(deviations)
49     self._initialize_forbid_repetitions(forbid_repetitions)
50     self._initialize_grace_expressions(grace_expressions)
51     self._initialize_logical_tie_expressions(logical_tie_expressions)
52     self._initialize_pitch_application_rate(pitch_application_rate)
53     self._initialize_pitch_specifier(pitch_specifier)
54     self._initialize_pitch_operationSpecifier(pitch_operationSpecifier)
55     if pitches_are_nonsemantic is not None:
56         pitches_are_nonsemantic = bool(pitches_are_nonsemantic)
57     self._pitches_are_nonsemantic = pitches_are_nonsemantic
58
59     ### SPECIAL METHODS ###
60
61     @abc.abstractmethod
62     def __call__(
63         self,
64         timewise_seed,
65         logical_tie,
66         attack_point_signature,
67         phrase_seed,
68         pitch_range,
69         previous_pitch,
70         seed,
71         transposition,
72         ):
73         raise NotImplementedError
74
75     ### PRIVATE METHODS ###
76
77     def _apply_logical_tie_expression(
78         self,
79         logical_tie,
80         pitch_range,
81         seed,
82         ):
83         if self.logical_tie_expressions:
84             logical_tie_expression = self.logical_tie_expressions[seed]
85             if logical_tie_expression is not None:
86                 logical_tie_expression(
87                     logical_tie,
88                     pitch_range=pitch_range,
89                     )
90

```

```

91     def _apply_deviation(
92         self,
93         pitch,
94         seed,
95     ):
96         if self.deviations:
97             deviation = self.deviations[seed]
98             if isinstance(deviation, pitchtools.NumberedInterval):
99                 if deviation != 0:
100                     pitch = pitchtools.NumberedPitch(pitch)
101                     pitch = pitch.transpose(deviation)
102                     pitch = pitchtools.NamedPitch(pitch)
103             elif isinstance(deviation, pitchtools.NamedInterval):
104                 pitch = pitch.transpose(deviation)
105
106         return pitch
107
108     @staticmethod
109     def _get_timewise_seed(
110         timewise_seeds_by_music_specifier,
111         music_specifier,
112     ):
113         if music_specifier in timewise_seeds_by_music_specifier:
114             timewise_seeds_by_music_specifier[music_specifier] += 1
115         else:
116             timewise_seeds_by_music_specifier[music_specifier] = 0
117
118     @staticmethod
119     def _get_grace_logical_ties(logical_tie):
120         logical_ties = []
121         head = logical_tie.head
122         previous_leaf = inspect_(head).get_leaf(-1)
123         if previous_leaf is None:
124             return logical_ties
125         grace_containers = inspect_(previous_leaf).get_grace_containers(
126             'after')
127         if grace_containers:
128             grace_container = grace_containers[0]
129             for logical_tie in iterate(grace_container).by_logical_tie(
130                 pitched=True,
131             ):
132                 logical_ties.append(logical_tie)
133
134     @staticmethod
135     def _get_instrument(logical_tie, music_specifier):
136         if music_specifier.instrument is not None:
137             return music_specifier.instrument
138         component = logical_tie.head
139         prototype = instrumenttools.Instrument
140         instrument = inspect_(component).get_effective(prototype)
141
142         return instrument
143
144     @staticmethod

```

```

145     def _get_phrase_seed(
146         attack_point_signature,
147         music_specifier,
148         phrase_seeds,
149         voice,
150     ):
151         if attack_point_signature.is_first_of_phrase:
152             if (voice, music_specifier) not in phrase_seeds:
153                 phrase_seed = (music_specifier.seed or 0) - 1
154                 phrase_seeds[(voice, music_specifier)] = phrase_seed
155                 phrase_seeds[(voice, music_specifier)] += 1
156             phrase_seed = phrase_seeds[(voice, music_specifier)]
157         return phrase_seed
158
159     def _get_pitch_choices(
160         self,
161         logical_tie,
162         music_specifier,
163         pitch_choice_timepans_by_music_specifier,
164         segment_duration,
165     ):
166         if music_specifier not in pitch_choice_timepans_by_music_specifier:
167             pitch_handler = music_specifier.pitch_handler
168             pitch_specifier = pitch_handler.pitch_specifier
169             operation_specifier = pitch_handler.pitch_operation_specifier
170             pitch_choice_timepans = PitchHandler.get_pitch_choice_timepans(
171                 pitch_specifier=pitch_specifier,
172                 operation_specifier=operation_specifier,
173                 duration=segment_duration,
174             )
175             pitch_choice_timepans_by_music_specifier[music_specifier] = \
176                 pitch_choice_timepans
177             timespans = pitch_choice_timepans_by_music_specifier[
178                 music_specifier]
179             assert len(timespans)
180             start_offset = logical_tie.get_timespan().start_offset
181             # TODO: "overlapping" should include "starting at"
182             found_timepans = \
183                 timespans.find_timepans_overlapping_offset(start_offset)
184             found_timepans += \
185                 timespans.find_timepans_starting_at(start_offset)
186             timespan = found_timepans[0]
187             pitch_choices = timespan.annotation
188         return pitch_choices
189
190     @staticmethod
191     def _get_pitch_range(
192         instrument,
193         logical_tie,
194     ):
195         prototype = pitchtools.PitchRange
196         component = logical_tie.head
197         pitch_range = inspect_(component).get_effective(prototype)
198         if pitch_range is None and instrument is not None:

```

```

199     pitch_range = instrument.pitch_range
200     return pitch_range
201
202     @staticmethod
203     def _get_previous_pitch(
204         music_specifier,
205         previous_pitch_by_music_specifier,
206         voice,
207     ):
208         key = (voice, music_specifier)
209         if key not in previous_pitch_by_music_specifier:
210             previous_pitch_by_music_specifier[key] = None
211         previous_pitch = previous_pitch_by_music_specifier[key]
212         return previous_pitch
213
214     @staticmethod
215     def _get_pitch_seed(
216         attack_point_signature,
217         music_specifier,
218         pitch_application_rate,
219         pitch_seeds_by_music_specifier,
220         pitch_seeds_by_voice,
221         voice,
222     ):
223         if music_specifier not in pitch_seeds_by_music_specifier:
224             seed = (music_specifier.seed or 0) - 1
225             pitch_seeds_by_music_specifier[music_specifier] = seed
226             pitch_seeds_by_voice[voice] = seed
227         if pitch_application_rate == 'phrase':
228             if attack_point_signature.is_first_of_phrase:
229                 pitch_seeds_by_music_specifier[music_specifier] += 1
230                 seed = pitch_seeds_by_music_specifier[music_specifier]
231                 pitch_seeds_by_voice[voice] = seed
232             else:
233                 seed = pitch_seeds_by_voice[voice]
234         elif pitch_application_rate == 'division':
235             if attack_point_signature.is_first_of_division:
236                 pitch_seeds_by_music_specifier[music_specifier] += 1
237                 seed = pitch_seeds_by_music_specifier[music_specifier]
238                 pitch_seeds_by_voice[voice] = seed
239             else:
240                 seed = pitch_seeds_by_voice[voice]
241         else:
242             pitch_seeds_by_music_specifier[music_specifier] += 1
243             seed = pitch_seeds_by_music_specifier[music_specifier]
244         return seed
245
246     @staticmethod
247     def _get_sounding_pitch(
248         instrument,
249         pitch_handler,
250     ):
251         if not instrument:
252             return pitchtools.NamedPitch("c")

```

```

253     sounding_pitch = instrument.sounding_pitch_of_written_middle_c
254     transposition_is_non_octave = sounding_pitch.named_pitch_class != \
255         pitchtools.NamedPitchClass('c')
256     if transposition_is_non_octave:
257         if pitch_handler.pitches_are_nonsensematic:
258             return pitchtools.NamedPitch("c")
259         return sounding_pitch
260     return None
261
262 def _initialize_deviations(self, deviations):
263     if deviations is not None:
264         if not isinstance(deviations, collections.Sequence):
265             deviations = (deviations,)
266         assert len(deviations)
267         intervals = []
268         for interval in deviations:
269             if isinstance(interval, (int, float)):
270                 interval = pitchtools.NumberedInterval(interval)
271             elif isinstance(interval, str):
272                 interval = pitchtools.NamedInterval(interval)
273             elif isinstance(interval, pitchtools.Interval):
274                 pass
275             else:
276                 interval = pitchtools.NumberedInterval(interval)
277             intervals.append(interval)
278         deviations = datastructuretools.CyclicTuple(intervals)
279     self._deviations = deviations
280
281 def _initialize_forbid_repetitions(self, forbid_repetitions):
282     if forbid_repetitions is not None:
283         forbid_repetitions = bool(forbid_repetitions)
284     self._forbid_repetitions = forbid_repetitions
285
286 def _initialize_grace_expressions(self, grace_expressions):
287     import consort
288     if grace_expressions is not None:
289         prototype = consort.LogicalTieExpression
290         assert grace_expressions, grace_expressions
291         assert all(isinstance(_, prototype)
292                  for _ in grace_expressions), \
293                  grace_expressions
294         grace_expressions = datastructuretools.CyclicTuple(
295             grace_expressions,
296             )
297     self._grace_expressions = grace_expressions
298
299 def _initialize_logical_tie_expressions(self, logical_tie_expressions):
300     import consort
301     if logical_tie_expressions is not None:
302         prototype = (consort.LogicalTieExpression, type(None))
303         assert logical_tie_expressions, logical_tie_expressions
304         assert all(isinstance(_, prototype)
305                  for _ in logical_tie_expressions), \
306                  logical_tie_expressions

```

```

307     logical_tie_expressions = datastructuretools.CyclicTuple(
308         logical_tie_expressions,
309         )
310     self._logical_tie_expressions = logical_tie_expressions
311
312     def _initialize_pitch_application_rate(self, pitch_application_rate):
313         assert pitch_application_rate in (
314             None, 'logical_tie', 'division', 'phrase',
315             )
316         self._pitch_application_rate = pitch_application_rate
317
318     def _initialize_pitch_operationSpecifier(self, pitch_operationSpecifier):
319         import consort
320         if pitch_operationSpecifier is not None:
321             prototype = consort.PitchOperationSpecifier
322             if not isinstance(pitch_operationSpecifier, prototype):
323                 pitch_operationSpecifier = consort.PitchOperationSpecifier(
324                     pitch_operations=pitch_operationSpecifier,
325                     )
326             self._pitch_operationSpecifier = pitch_operationSpecifier
327
328     def _initialize_pitchSpecifier(self, pitchSpecifier):
329         import consort
330         if pitchSpecifier is not None:
331             if not isinstance(pitchSpecifier, consort.PitchSpecifier):
332                 pitchSpecifier = consort.PitchSpecifier(pitchSpecifier)
333             self._pitchSpecifier = pitchSpecifier
334
335     def _process_logical_tie(self, logical_tie, pitch, pitch_range, seed):
336         for leaf in logical_tie:
337             leaf.written_pitch = pitch
338             grace_logical_ties = self._get_grace_logical_ties(logical_tie)
339             if str(pitch.accidental) and grace_logical_ties:
340                 leaf.note_head.is_forced = True
341             self._apply_logical_tie_expression(
342                 logical_tie,
343                 seed=seed,
344                 pitch_range=pitch_range,
345                 )
346             for i, grace_logical_tie in enumerate(grace_logical_ties, seed):
347                 for leaf in grace_logical_tie:
348                     leaf.written_pitch = pitch
349                     if self.grace_expressions:
350                         grace_expression = self.grace_expressions[i]
351                         grace_expression(grace_logical_tie)
352
353     @staticmethod
354     def _process_session(segment_maker):
355         import consort
356         maker = consort.SegmentMaker
357         segment_duration = segment_maker.measure_offsets[-1]
358         attack_point_map = segment_maker.attack_point_map
359         pitch_choice_timeSpans_by_music_specifier = {}
360         previous_pitch_by_music_specifier = {}

```

```

361     seed_session = consort.SeedSession()
362     for logical_tie in attack_point_map:
363         music_specifier = maker.logical_tie_to_music_specifier(logical_tie)
364         if not music_specifier or not music_specifier.pitch_handler:
365             continue
366         pitch_handler = music_specifier.pitch_handler
367         attack_point_signature = attack_point_map[logical_tie]
368         application_rate = pitch_handler.pitch_application_rate
369         voice = consort.SegmentMaker.logical_tie_to_voice(logical_tie)
370         seed_session(
371             application_rate,
372             attack_point_signature,
373             music_specifier,
374             voice,
375             )
376         previous_pitch = pitch_handler._get_previous_pitch(
377             music_specifier,
378             previous_pitch_by_music_specifier,
379             voice,
380             )
381         pitch_choices = pitch_handler._get_pitch_choices(
382             logical_tie,
383             music_specifier,
384             pitch_choice_time_spans_by_music_specifier,
385             segment_duration,
386             )
387         pitch = pitch_handler(
388             attack_point_signature,
389             logical_tie,
390             music_specifier,
391             pitch_choices,
392             previous_pitch,
393             seed_session,
394             )
395         pitch_handler._set_previous_pitch(
396             attack_point_signature,
397             music_specifier,
398             pitch,
399             pitch_handler.pitch_application_rate,
400             previous_pitch_by_music_specifier,
401             voice,
402             )
403         pitch_handler._apply_transposition(
404             attack_point_signature,
405             logical_tie,
406             music_specifier,
407             pitch_handler,
408             )
409         instrument = pitch_handler._get_instrument(
410             logical_tie,
411             music_specifier,
412             )
413         pitch_range = pitch_handler._get_pitch_range(
414             instrument,

```

```

415         logical_tie,
416     )
417     pitch_handler._process_logical_tie(
418         logical_tie,
419         pitch,
420         pitch_range,
421         seed_session.current_unphrased_voicewise_logical_tie_seed,
422     )
423
424     @staticmethod
425     def _apply_transposition(
426         attack_point_signature,
427         logical_tie,
428         music_specifier,
429         pitch_handler,
430     ):
431         import consort
432         if not attack_point_signature.is_first_of_phrase:
433             return
434         voice = consort.SegmentMaker.logical_tie_to_voice(logical_tie)
435         instrument = PitchHandler._get_instrument(
436             logical_tie, music_specifier)
437         sounding_pitch = PitchHandler._get_sounding_pitch(
438             instrument, pitch_handler)
439         if pitch_handler and pitch_handler.pitches_are_nonsemantic:
440             sounding_pitch = pitchtools.NamedPitch('C4')
441         if sounding_pitch is None:
442             sounding_pitch = pitchtools.NamedPitch('C4')
443         if sounding_pitch == pitchtools.NamedPitch('C4'):
444             return
445         phrase = consort.SegmentMaker.logical_tie_to_phrase(logical_tie)
446         transposition_command = indicatortools.LilyPondCommand(
447             "transpose {!s} c'".format(sounding_pitch),
448             format_slot='before',
449         )
450         print(
451             '        transposing',
452             voice.name,
453             voice.index(phrase),
454             sounding_pitch,
455         )
456         attach(transposition_command, phrase)
457
458     @staticmethod
459     def _set_previous_pitch(
460         attack_point_signature,
461         music_specifier,
462         pitch,
463         pitch_application_rate,
464         previous_pitch_by_music_specifier,
465         voice,
466     ):
467         key = (voice, music_specifier)
468         if pitch_application_rate == 'phrase':

```

```

469         if attack_point_signature.is_first_of_phrase:
470             previous_pitch_by_music_specifier[key] = pitch
471         elif pitch_application_rate == 'division':
472             if attack_point_signature.is_first_of_division:
473                 previous_pitch_by_music_specifier[key] = pitch
474         else:
475             previous_pitch_by_music_specifier[key] = pitch
476
477     ### PUBLIC METHODS ###
478
479     @staticmethod
480     def get_pitch_choice_timepans(
481         pitch_specifier=None,
482         operationSpecifier=None,
483         duration=1,
484     ):
485         r'''Get pitch expression timepans.
486
487         :::
488
489         >>> import consort
490         >>> pitch_specifier = consort.PitchSpecifier(
491             ...     pitch_segments=(
492             ...     "c' e' g'", ...
493             ...     "fs' g' a'", ...
494             ...     "b' d",
495             ...     ),
496             ...     ratio=(1, 2, 3),
497             ... )
498         >>> operation_specifier = consort.PitchOperationSpecifier(
499             ...     pitch_operations=(
500             ...     pitchtools.PitchOperation(
501                 ...     pitchtools.Rotation(1),
502                 ...     pitchtools.Transposition(1),
503                 ... )),
504             ...     None,
505             ...     pitchtools.PitchOperation(
506                 ...     pitchtools.Rotation(-1),
507                 ...     pitchtools.Transposition(-1),
508                 ... )),
509             ...     ),
510             ...     ratio=(1, 2, 1),
511             ... )
512         >>> timepans = consort.PitchHandler.get_pitch_choice_timepans(
513             ...     pitch_specifier=pitch_specifier,
514             ...     operation_specifier=operation_specifier,
515             ...     duration=12,
516             ... )
517         >>> print(format(timepans))
518         consort.tools.TimespanCollection(
519             [
520                 timespantools.AnnotatedTimespan(
521                     start_offset=durationtools.Offset(0, 1),
522                     stop_offset=durationtools.Offset(2, 1),

```

```

523     annotation=datastructuretools.CyclicTuple(
524         [
525             pitchtools.NamedPitch("df'"),
526             pitchtools.NamedPitch('gf'),
527             pitchtools.NamedPitch('bf'),
528             ]
529         ),
530     ),
531     timespan=timespantools.AnnotatedTimespan(
532         start_offset=durationtools.Offset(2, 1),
533         stop_offset=durationtools.Offset(3, 1),
534         annotation=datastructuretools.CyclicTuple(
535             [
536                 pitchtools.NamedPitch("g'"),
537                 pitchtools.NamedPitch("e'"),
538                 pitchtools.NamedPitch("f'"),
539                 ]
540             ),
541         ),
542         timespan=timespantools.AnnotatedTimespan(
543             start_offset=durationtools.Offset(3, 1),
544             stop_offset=durationtools.Offset(6, 1),
545             annotation=datastructuretools.CyclicTuple(
546                 [
547                     pitchtools.NamedPitch("fs'"),
548                     pitchtools.NamedPitch("g'"),
549                     pitchtools.NamedPitch("a'"),
550                     ]
551                 ),
552             ),
553             timespan=timespantools.AnnotatedTimespan(
554                 start_offset=durationtools.Offset(6, 1),
555                 stop_offset=durationtools.Offset(9, 1),
556                 annotation=datastructuretools.CyclicTuple(
557                     [
558                         pitchtools.NamedPitch('b'),
559                         pitchtools.NamedPitch('d'),
560                         ]
561                     ),
562                 ),
563                 timespan=timespantools.AnnotatedTimespan(
564                     start_offset=durationtools.Offset(9, 1),
565                     stop_offset=durationtools.Offset(12, 1),
566                     annotation=datastructuretools.CyclicTuple(
567                         [
568                             pitchtools.NamedPitch('as'),
569                             pitchtools.NamedPitch("fss"),
570                             ]
571                         ),
572                     ),
573                 ],
574             )
575         )
576 Returns timespans.

```

```

577     ''
578     import consort
579     duration = durationtools.Duration(duration)
580     pitchSpecifier = pitchSpecifier or consort.PitchSpecifier()
581     operationSpecifier = operationSpecifier or \
582         consort.PitchOperationSpecifier()
583     pitchChoiceTimespans = consort.TimespanCollection()
584     pitchTimespans = pitchSpecifier.getTimespans(duration)
585     operationTimespans = operationSpecifier.getTimespans(duration)
586     offsets = set()
587     offsets.update(pitchTimespans.allOffsets)
588     offsets.update(operationTimespans.allOffsets)
589     offsets = tuple(sorted(offsets))
590     for startOffset, stopOffset in sequencetools.iterateSequenceNwise(
591         offsets):
592         timespan = timespantools.Timespan(
593             startOffset=startOffset,
594             stopOffset=stopOffset,
595             )
596         pitchTimespan = \
597             pitchTimespans.findTimespansIntersectingTimespan(
598                 timespan)[0]
599         pitches = pitchTimespan.annotation
600         operationTimespan = \
601             operationTimespans.findTimespansIntersectingTimespan(
602                 timespan)[0]
603         operation = operationTimespan.annotation
604         if operation is not None:
605             pitches = operation(pitches)
606             pitches = datastructuretools.CyclicTuple(pitches)
607             pitchChoiceTimespan = timespantools.AnnotatedTimespan(
608                 annotation=pitches,
609                 startOffset=startOffset,
610                 stopOffset=stopOffset,
611                 )
612             pitchChoiceTimespans.insert(pitchChoiceTimespan)
613     return pitchChoiceTimespans
614
615     def transpose(self, expr):
616         import consort
617         pitchSpecifier = self.pitchSpecifier or consort.PitchSpecifier(
618             pitchSegments='C4',
619             )
620         pitchSpecifier = pitchSpecifier.transpose(expr)
621         return new(self, pitchSpecifier=pitchSpecifier)
622
623     ### PUBLIC PROPERTIES ###
624
625     @property
626     def deviations(self):
627         return self._deviations
628
629     @property
630     def forbidRepetitions(self):

```

```

631     return self._forbid_repetitions
632
633     @property
634     def grace_expressions(self):
635         return self._grace_expressions
636
637     @property
638     def logical_tie_expressions(self):
639         return self._logical_tie_expressions
640
641     @property
642     def pitch_application_rate(self):
643         return self._pitch_application_rate
644
645     @property
646     def pitch_operation_specifier(self):
647         return self._pitch_operation_specifier
648
649     @property
650     def pitchSpecifier(self):
651         return self._pitchSpecifier
652
653     @property
654     def pitches_are_nonsemantic(self):
655         return self._pitches_are_nonsemantic

```

## A.35 CONSORTTOOLS.PITCHOPERATIONSPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad import new
4 from abjad.tools import abctools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import sequencetools
8 from abjad.tools import timespantools
9
10
11 class PitchOperationSpecifier(abctools.AbjadValueObject):
12     r'''A operation specifier.
13
14     .. container:: example
15
16         ::

17
18         >>> import consort
19         >>> pitch_operationSpecifier = consort.PitchOperationSpecifier(
20             ...     pitch_operations=(
21             ...         pitchtools.PitchOperation(
22             ...             pitchtools.Rotation(1),
23             ...             pitchtools.Transposition(1),
24             ...             )),
25             ...         None,
26             ...         pitchtools.PitchOperation(

```

```

27         ...           pitchtools.Rotation(-1),
28         ...           pitchtools.Transposition(-1),
29         ...           ))
30         ...           ),
31         ...           ratio=(1, 2, 1),
32         ...           )
33 >>> print(format(pitch_operationSpecifier))
34 consort.tools.PitchOperationSpecifier(
35     pitch_operations=
36         pitchtools.PitchOperation(
37             operators=
38                 pitchtools.Rotation(
39                     index=1,
40                     transpose=True,
41                     ),
42                     pitchtools.Transposition(
43                         index=1,
44                         ),
45                     ),
46                     ),
47 None,
48         pitchtools.PitchOperation(
49             operators=
50                 pitchtools.Rotation(
51                     index=-1,
52                     transpose=True,
53                     ),
54                     pitchtools.Transposition(
55                         index=-1,
56                         ),
57                     ),
58                     ),
59                     ),
60             ratio=mathtools.Ratio((1, 2, 1)),
61         )
62
63     ...
64
65     ### CLASS VARIABLES ###
66
67     __slots__ = (
68         '_is_cumulative',
69         '_pitch_operations',
70         '_ratio',
71     )
72
73     ### INITIALIZER ###
74
75     def __init__(
76         self,
77         pitch_operations=None,
78         ratio=None,
79         is_cumulative=None,
80     ):

```

```

81     if pitch_operations is not None:
82         if not isinstance(pitch_operations, collections.Sequence):
83             pitch_operations = (pitch_operations,)
84         prototype = (
85             pitchtools.PitchOperation,
86             type(None),
87         )
88         coerced_pitch_operations = []
89         for x in pitch_operations:
90             if not isinstance(x, prototype):
91                 x = pitchtools.PitchOperation(x)
92             coerced_pitch_operations.append(x)
93         pitch_operations = tuple(coerced_pitch_operations)
94         assert len(pitch_operations)
95
96     if pitch_operations and not ratio:
97         ratio = [1] * len(pitch_operations)
98
99     if ratio is not None:
100        ratio = mathtools.Ratio([abs(_) for _ in ratio])
101        assert len(ratio) == len(pitch_operations)
102
103    if is_cumulative is not None:
104        is_cumulative = bool(is_cumulative)
105
106    self._is_cumulative = is_cumulative
107    self._pitch_operations = pitch_operations
108    self._ratio = ratio
109
110    ### PUBLIC METHODS ###
111
112    def get_timespans(self, stop_offset):
113        '''Gets pitch expr timespans.
114
115        .. container:: example
116
117            ::

118            >>> timespans = pitch_operationSpecifier.get_timespans(10)
119            >>> print(format(timespans))
120            consort.tools.TimespanCollection(
121                [
122                    timespantools.AnnotatedTimespan(
123                        start_offset=durationtools.Offset(0, 1),
124                        stop_offset=durationtools.Offset(5, 2),
125                        annotation=pitchtools.PitchOperation(
126                            operators=(
127                                pitchtools.Rotation(
128                                    index=1,
129                                    transpose=True,
130                                ),
131                                pitchtools.Transposition(
132                                    index=1,
133                                ),

```

```

135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
        ),
        ),
        ),
        timespantools.AnnotatedTimespan(
            start_offset=durationtools.Offset(5, 2),
            stop_offset=durationtools.Offset(15, 2),
            ),
            timespantools.AnnotatedTimespan(
                start_offset=durationtools.Offset(15, 2),
                stop_offset=durationtools.Offset(10, 1),
                annotation=pitchtools.PitchOperation(
                    operators=(
                        pitchtools.Rotation(
                            index=-1,
                            transpose=True,
                            ),
                            pitchtools.Transposition(
                                index=-1,
                                ),
                                ),
                                ),
                                ),
                                ],
)
... container:: example
:::
>>> pitch_operationSpecifier = consort.PitchOperationSpecifier(
...     is_cumulative=True,
...     pitch_operations=(
...         pitchtools.Rotation(1),
...         pitchtools.Transposition(1),
...         pitchtools.Inversion(),
...         None,
...         ),
...         ratio=(1, 2, 3, 1),
...         )
...
>>> timespans = pitch_operationSpecifier.get_timespans(10)
>>> print(format(timespans))
consort.tools.TimespanCollection(
[
    timespantools.AnnotatedTimespan(
        start_offset=durationtools.Offset(0, 1),
        stop_offset=durationtools.Offset(10, 7),
        annotation=pitchtools.PitchOperation(
            operators=(
                pitchtools.Rotation(
                    index=1,
                    transpose=True,
                    ),
                    ),
                    ),
                    ),
                    ),

```

```

189         ),
190         timespanTools.AnnotatedTimespan(
191             start_offset=durationtools.Offset(10, 7),
192             stop_offset=durationtools.Offset(30, 7),
193             annotation=pitchtools.PitchOperation(
194                 operators=(
195                     pitchtools.Rotation(
196                         index=1,
197                         transpose=True,
198                         ),
199                     pitchtools.Transposition(
200                         index=1,
201                         ),
202                     ),
203                     ),
204             ),
205             timespanTools.AnnotatedTimespan(
206                 start_offset=durationtools.Offset(30, 7),
207                 stop_offset=durationtools.Offset(60, 7),
208                 annotation=pitchtools.PitchOperation(
209                     operators=(
210                         pitchtools.Rotation(
211                             index=1,
212                             transpose=True,
213                             ),
214                         pitchtools.Transposition(
215                             index=1,
216                             ),
217                         pitchtools.Inversion(),
218                         ),
219                         ),
220             ),
221             timespanTools.AnnotatedTimespan(
222                 start_offset=durationtools.Offset(60, 7),
223                 stop_offset=durationtools.Offset(10, 1),
224                 annotation=pitchtools.PitchOperation(
225                     operators=(
226                         pitchtools.Rotation(
227                             index=1,
228                             transpose=True,
229                             ),
230                         pitchtools.Transposition(
231                             index=1,
232                             ),
233                         pitchtools.Inversion(),
234                         ),
235                         ),
236                     ),
237                 ],
238             )
239
240     Returns timespan collection.
241     """
242     import consort

```

```

243     timespans = consort.TimespanCollection()
244     if not self.ratio or not self.pitch_operations:
245         annotated_timespan = timespantools.AnnotatedTimespan(
246             start_offset=0,
247             stop_offset=stop_offset,
248         ),
249         timespans.insert(annotated_timespan)
250     else:
251         target_timespan = timespantools.Timespan(
252             start_offset=0,
253             stop_offset=stop_offset,
254         )
255         divided_timespans = target_timespan.divide_by_ratio(self.ratio)
256         pitch_operation = pitchtools.PitchOperation()
257         for i, timespan in enumerate(divided_timespans):
258             current_pitch_operation = self._pitch_operations[i]
259             if self.is_cumulative:
260                 if current_pitch_operation:
261                     pitch_operation = pitchtools.PitchOperation(
262                         (pitch_operation.operators or ()) +
263                         current_pitch_operation.operators
264                     )
265             else:
266                 pitch_operation = current_pitch_operation
267             annotated_timespan = timespantools.AnnotatedTimespan(
268                 annotation=pitch_operation,
269                 start_offset=timespan.start_offset,
270                 stop_offset=timespan.stop_offset,
271             )
272             timespans.insert(annotated_timespan)
273     return timespans
274
275     def rotate(self, rotation):
276         r'''Rotates pitch operation specifier.
277
278         :::
279
280         >>> pitch_operationSpecifier = consort.PitchOperationSpecifier(
281             ...     pitch_operations=(
282             ...         pitchtools.PitchOperation((
283             ...             pitchtools.Rotation(1),
284             ...             pitchtools.Transposition(1),
285             ...             )),
286             ...             None,
287             ...             pitchtools.PitchOperation((
288             ...                 pitchtools.Rotation(-1),
289             ...                 pitchtools.Transposition(-1),
290             ...                 )),
291             ...             ),
292             ...             ratio=(1, 2, 1),
293             ...         )
294         >>> rotatedSpecifier = pitch_operationSpecifier.rotate(1)
295         >>> print(format(rotatedSpecifier))
296         consort.tools.PitchOperationSpecifier(

```

```

297     pitch_operations=(
298         pitchtools.PitchOperation(
299             operators=(
300                 pitchtools.Rotation(
301                     index=-1,
302                     transpose=True,
303                     ),
304                 pitchtools.Transposition(
305                     index=-1,
306                     ),
307                     ),
308                 ),
309         pitchtools.PitchOperation(
310             operators=(
311                 pitchtools.Rotation(
312                     index=1,
313                     transpose=True,
314                     ),
315                 pitchtools.Transposition(
316                     index=1,
317                     ),
318                     ),
319                     ),
320             None,
321             ),
322         ratio=mathtools.Ratio((1, 1, 2)),
323     )
324
325     Returns new pitch specifier.
326     """
327     rotation = int(rotation)
328     pitch_operations = sequencetools.rotate_sequence(
329         self.pitch_operations, rotation)
330     ratio = sequencetools.rotate_sequence(self.ratio, rotation)
331     return new(
332         self,
333         pitch_operations=pitch_operations,
334         ratio=ratio,
335     )
336
337     ### PUBLIC PROPERTIES ###
338
339     @property
340     def is_cumulative(self):
341         return self._is_cumulative
342
343     @property
344     def ratio(self):
345         return self._ratio
346
347     @property
348     def pitch_operations(self):
349         return self._pitch_operations

```

## A.36 CONSORTTOOLS.PITCHSPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import abctools
4 from abjad.tools import mathtools
5 from abjad.tools import pitchtools
6 from abjad.tools import sequencetools
7 from abjad.tools import timespantools
8
9
10 class PitchSpecifier(abctools.AbjadValueObject):
11     r'''A pitch specifier.
12
13     :::
14
15         >>> import consort
16         >>> pitchSpecifier = consort.PitchSpecifier(
17             ...     pitch_segments=(
18                 ...     "c' e' g''",
19                 ...     "fs' gs''",
20                 ...     "b",
21                 ...     ),
22                 ...     ratio=(1, 2, 3),
23                 ...     )
24         >>> print(format(pitchSpecifier))
25         consort.tools.PitchSpecifier(
26             pitch_segments=(
27                 pitchtools.PitchSegment(
28                     (
29                         pitchtools.NamedPitch("c'"),
30                         pitchtools.NamedPitch("e'"),
31                         pitchtools.NamedPitch("g'"),
32                         ),
33                         item_class=pitchtools.NamedPitch,
34                         ),
35                 pitchtools.PitchSegment(
36                     (
37                         pitchtools.NamedPitch("fs'"),
38                         pitchtools.NamedPitch("gs'"),
39                         ),
40                         item_class=pitchtools.NamedPitch,
41                         ),
42                 pitchtools.PitchSegment(
43                     (
44                         pitchtools.NamedPitch('b'),
45                         ),
46                         item_class=pitchtools.NamedPitch,
47                         ),
48                     ),
49                     ratio=mathtools.Ratio((1, 2, 3)),
50                     )
51
52     Pitch specifiers can be instantiated from a string of pitch names:
```

```

53
54 :::
55
56     >>> pitchSpecifier = consort.PitchSpecifier("c' e' g' a'")
57     >>> print(format(pitchSpecifier))
58     consort.tools.PitchSpecifier(
59         pitch_segments=(
60             pitchtools.PitchSegment(
61                 (
62                     pitchtools.NamedPitch("c'"),
63                     pitchtools.NamedPitch("e'"),
64                     pitchtools.NamedPitch("g'"),
65                     pitchtools.NamedPitch("a'"),
66                 ),
67                 item_class=pitchtools.NamedPitch,
68             ),
69         ),
70         ratio=mathtools.Ratio((1,)),
71     )
72
73 Pitch specifiers can be instantiated from a single pitch:
74 :::
75
76
77     >>> pitchSpecifier = consort.PitchSpecifier(NamedPitch("ds'"))
78     >>> print(format(pitchSpecifier))
79     consort.tools.PitchSpecifier(
80         pitch_segments=(
81             pitchtools.PitchSegment(
82                 (
83                     pitchtools.NamedPitch("ds'"),
84                 ),
85                 item_class=pitchtools.NamedPitch,
86             ),
87         ),
88         ratio=mathtools.Ratio((1,)),
89     )
90
91     """
92
93     ### CLASS VARIABLES ###
94
95     __slots__ = (
96         '_pitch_segments',
97         '_ratio',
98     )
99
100    ### INITIALIZER ###
101
102    def __init__(
103        self,
104        pitch_segments=None,
105        ratio=None,
106    ):

```

```

107     if pitch_segments is not None:
108         if isinstance(pitch_segments, pitchtools.Pitch):
109             pitch_segments = pitchtools.PitchSegment([pitch_segments])
110         elif isinstance(pitch_segments, str):
111             pitch_segments = pitchtools.PitchSegment(pitch_segments)
112         if isinstance(pitch_segments, pitchtools.PitchSegment):
113             pitch_segments = [pitch_segments]
114         coerced_pitch_segments = []
115         for pitch_segment in pitch_segments:
116             pitch_segment = pitchtools.PitchSegment(
117                 pitch_segment,
118                 item_class=pitchtools.NamedPitch,
119                 )
120             if not pitch_segment:
121                 pitch_segment = pitchtools.PitchSegment("c")
122             coerced_pitch_segments.append(pitch_segment)
123         pitch_segments = tuple(coerced_pitch_segments)
124         assert len(pitch_segments)
125
126     if pitch_segments and not ratio:
127         ratio = [1] * len(pitch_segments)
128
129     if ratio is not None:
130         ratio = mathtools.Ratio([abs(x) for x in ratio])
131         assert len(ratio) == len(pitch_segments)
132
133     self._pitch_segments = pitch_segments
134     self._ratio = ratio
135
136     ### PUBLIC METHODS ###
137
138     def get_timespans(self, stop_offset):
139         '''Gets pitch segment timespans.
140
141         :::
142
143         >>> pitchSpecifier = consort.PitchSpecifier(
144             ...     pitch_segments=(
145             ...         "c' e' g",
146             ...         "fs' g",
147             ...         "b",
148             ...         ),
149             ...         ratio=(1, 2, 3),
150             ...         )
151         >>> timespans = pitchSpecifier.get_timespans(stop_offset=10)
152         >>> print(format(timespans))
153         consort.tools.TimespanCollection(
154             [
155                 timespantools.AnnotatedTimespan(
156                     start_offset=durationtools.Offset(0, 1),
157                     stop_offset=durationtools.Offset(5, 3),
158                     annotation=pitchtools.PitchSegment(
159                         (
160                             pitchtools.NamedPitch("c"),

```

```

161             pitchtools.NamedPitch("e'"),
162             pitchtools.NamedPitch("g'"),
163             ),
164             item_class=pitchtools.NamedPitch,
165             ),
166             ),
167             timespantools.AnnotatedTimespan(
168                 start_offset=durationtools.Offset(5, 3),
169                 stop_offset=durationtools.Offset(5, 1),
170                 annotation=pitchtools.PitchSegment(
171                     (
172                         pitchtools.NamedPitch("fs'"),
173                         pitchtools.NamedPitch("g'"),
174                         ),
175                         item_class=pitchtools.NamedPitch,
176                         ),
177                         ),
178                         timespantools.AnnotatedTimespan(
179                             start_offset=durationtools.Offset(5, 1),
180                             stop_offset=durationtools.Offset(10, 1),
181                             annotation=pitchtools.PitchSegment(
182                                 (
183                                     pitchtools.NamedPitch('b'),
184                                     ),
185                                     item_class=pitchtools.NamedPitch,
186                                     ),
187                                     ),
188                                     ],
189                                     )
190                                     )
191                                     """
192                                     import consort
193                                     timespans = consort.TimespanCollection()
194                                     if not self.ratio or not self.pitch_segments:
195                                         pitch_segment = pitchtools.PitchSegment("c")
196                                         annotated_timespan = timespantools.AnnotatedTimespan(
197                                             annotation=pitch_segment,
198                                             start_offset=0,
199                                             stop_offset=stop_offset,
200                                             ),
201                                         timespans.insert(annotated_timespan)
202                                     else:
203                                         target_timespan = timespantools.Timespan(
204                                             start_offset=0,
205                                             stop_offset=stop_offset,
206                                             )
207                                         divided_timespans = target_timespan.divide_by_ratio(self.ratio)
208                                         for i, timespan in enumerate(divided_timespans):
209                                             pitch_segment = self._pitch_segments[i]
210                                             annotated_timespan = timespantools.AnnotatedTimespan(
211                                                 annotation=pitch_segment,
212                                                 start_offset=timespan.start_offset,
213                                                 stop_offset=timespan.stop_offset,
214                                                 )

```

```

215         timespans.insert(annotated_timespan)
216     return timespans
217
218     def rotate(self, rotation):
219         r'''Rotates pitch specifier.
220
221         :::
222
223         >>> pitchSpecifier = consort.PitchSpecifier(
224             ...     pitchSegments=(
225                 ...         "c' e' g'",
226                 ...         "fs' gs'",
227                 ...         "b",
228                 ...         ),
229                 ...     ratio=(1, 2, 3),
230                 ... )
231
232         >>> rotatedPitchSpecifier = pitchSpecifier.rotate(1)
233         >>> print(format(rotatedPitchSpecifier))
234         consort.tools.PitchSpecifier(
235             pitchSegments=(
236                 pitchtools.PitchSegment(
237                     (
238                         pitchtools.NamedPitch('b'),
239                         ),
240                         item_class=pitchtools.NamedPitch,
241                         ),
242                         pitchtools.PitchSegment(
243                             (
244                                 pitchtools.NamedPitch("c'"),
245                                 pitchtools.NamedPitch('f'),
246                                 pitchtools.NamedPitch('a'),
247                                 ),
248                                 item_class=pitchtools.NamedPitch,
249                                 ),
250                                 pitchtools.PitchSegment(
251                                     (
252                                         pitchtools.NamedPitch("fs'"),
253                                         pitchtools.NamedPitch("e'"),
254                                         ),
255                                         item_class=pitchtools.NamedPitch,
256                                         ),
257                                         ),
258                                         ratio=mathtools.Ratio((3, 1, 2)),
259                                         )
260
261     Returns new pitch specifier.
262     '''
263     rotation = int(rotation)
264     pitchSegments = tuple(
265         _ .rotate(rotation, transpose=True)
266         for _ in self.pitchSegments
267     )
268     pitchSegments = sequencetools.rotate_sequence(
269         pitchSegments, rotation)

```

```

269     ratio = sequencetools.rotate_sequence(self.ratio, rotation)
270     return new(
271         self,
272         pitch_segments=pitch_segments,
273         ratio=ratio,
274     )
275
276     def transpose(self, expr=0):
277         r'''Transposes pitch specifier.
278
279         :::
280
281         >>> pitchSpecifier = consort.PitchSpecifier(
282             ...     pitch_segments=(
283             ...         "c' e' g'",
284             ...         "fs' gs'",
285             ...         "b",
286             ...         ),
287             ...         ratio=(1, 2, 3),
288             ...         )
289         >>> transposed_pitchSpecifier = pitchSpecifier.transpose('M2')
290         >>> print(format(transposed_pitchSpecifier))
291         consort.tools.PitchSpecifier(
292             pitch_segments=(
293                 pitchtools.PitchSegment(
294                     (
295                         pitchtools.NamedPitch("d"),
296                         pitchtools.NamedPitch("fs"),
297                         pitchtools.NamedPitch("a"),
298                         ),
299                         item_class=pitchtools.NamedPitch,
300                         ),
301                         pitchtools.PitchSegment(
302                             (
303                                 pitchtools.NamedPitch("gs"),
304                                 pitchtools.NamedPitch("as"),
305                                 ),
306                                 item_class=pitchtools.NamedPitch,
307                                 ),
308                                 pitchtools.PitchSegment(
309                                     (
310                                         pitchtools.NamedPitch("cs"),
311                                         ),
312                                         item_class=pitchtools.NamedPitch,
313                                         ),
314                                         ),
315                                         ratio=mathtools.Ratio((1, 2, 3)),
316                                         )
317
318     Returns new pitch specifier.
319     '''
320     pitch_segments = (_.transpose(expr) for _ in self.pitch_segments)
321     return new(self, pitch_segments=pitch_segments)
322

```

```

323     """ PUBLIC PROPERTIES """
324
325     @property
326     def ratio(self):
327         return self._ratio
328
329     @property
330     def pitch_segments(self):
331         return self._pitch_segments

```

## A.37 CONSORTTOOLS.PROPORTIONS

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import datastructuretools
3 from abjad.tools import durationtools
4 from abjad.tools import sequencetools
5
6
7 class Proportions(datastructuretools.TypedList):
8
9     """ CLASS VARIABLES """
10
11     __slots__ = ()
12
13     """ INITIALIZER """
14
15     def __init__(self, items=None):
16         datastructuretools.TypedList.__init__(
17             self,
18             items=items,
19         )
20
21     """ PRIVATE PROPERTIES """
22
23     @property
24     def _attribute_manifest(self):
25         from abjad.tools import systemtools
26         return systemtools.AttributeManifest()
27
28     """ PUBLIC METHODS """
29
30     def get_segment_desired_duration_in_seconds(self, segment_index, total_seconds):
31         segment_proportions = self[segment_index]
32         segment_total = sum(sequencetools.flatten_sequence(
33             segment_proportions))
34         ratio = durationtools.Multiplier(segment_total, self.total)
35         desired_duration_in_seconds = ratio * total_seconds
36         return desired_duration_in_seconds
37
38     """ PUBLIC PROPERTIES """
39
40     @property
41     def total(self):
42         return sum(sequencetools.flatten_sequence(self))

```

## A.38 CONSORTTOOLS.RATIOPARTSexpression

```
1 # -*- encoding: utf -*-
2 from abjad.tools import abctools
3 from abjad.tools import mathtools
4 from abjad.tools import timespantools
5
6
7 class RatioPartsExpression(abctools.AbjadObject):
8     r'''Ratio parts expression.
9
10    .. container:: example
11
12        :::
13
14            >>> import consort
15            >>> expression = consort.RatioPartsExpression(
16                ...     ratio=(1, 2, 1),
17                ...     parts=(0, 2),
18                ... )
19            >>> print(format(expression))
20            consort.tools.RatioPartsExpression(
21                parts=(0, 2),
22                ratio=mathtools.Ratio((1, 2, 1)),
23                )
24
25        :::
26
27
28        >>> timespan = timespantools.Timespan(
29            ...     start_offset=Duration(1, 2),
30            ...     stop_offset=Duration(3, 2),
31            ... )
32        >>> for x in expression(timespan):
33            ...     x
34            ...
35        Timespan(start_offset=Offset(1, 2), stop_offset=Offset(3, 4))
36        Timespan(start_offset=Offset(5, 4), stop_offset=Offset(3, 2))
37
38    .. container:: example
39
40        :::
41
42        >>> expression = consort.RatioPartsExpression(
43            ...     ratio=(1, 2, 1),
44            ...     parts=(0, 2),
45            ...     mask_timespan=timespantools.Timespan(
46            ...         start_offset=(1, 4),
47            ...         ),
48            ...     )
49
50        :::
51
52        >>> timespan = timespantools.Timespan(0, 4)
```

```

53         >>> for x in expression(timespan):
54             ...
55             ...
56             Timespan(start_offset=Offset(1, 4), stop_offset=Offset(1, 1))
57             Timespan(start_offset=Offset(3, 1), stop_offset=Offset(4, 1))
58
59     ...
60
61     ### CLASS VARIABLES ###
62
63     __slots__ = (
64         '_parts',
65         '_ratio',
66         '_mask_timespan',
67     )
68
69     ### INITIALIZER ###
70
71     def __init__(
72         self,
73         parts=0,
74         ratio=(1, 1),
75         mask_timespan=None,
76     ):
77         if not isinstance(ratio, mathtools.Ratio):
78             ratio = mathtools.Ratio(ratio)
79         self._ratio = ratio
80         if isinstance(parts, int):
81             parts = (parts,)
82         assert all(0 <= _ < len(ratio) for _ in parts)
83         parts = tuple(sorted(set(parts)))
84         self._parts = parts
85         if mask_timespan is not None:
86             assert isinstance(mask_timespan, timespantools.Timespan)
87         self._mask_timespan = mask_timespan
88
89     ### SPECIAL METHODS ###
90
91     def __call__(self, timespan):
92         assert isinstance(timespan, timespantools.Timespan)
93         divided_timespan = timespan.divide_by_ratio(self.ratio)
94         timespans = timespantools.TimespanInventory()
95         for part in self.parts:
96             timespans.append(divided_timespan[part])
97         if self.mask_timespan is not None:
98             timespans & self.mask_timespan
99         return timespans
100
101    ### PUBLIC METHODS ###
102
103    @staticmethod
104    def from_sequence(sequence):
105        r'''Creates a ratio parts expression from 'sequence'.
106

```

```

107      :::
108
109      >>> ratio = [-1, 2, -1, 1, -1]
110      >>> expression = consort.RatioPartsExpression.from_sequence(ratio)
111      >>> print(format(expression))
112      consort.tools.RatioPartsExpression(
113          parts=(1, 3),
114          ratio=mathtools.Ratio((1, 2, 1, 1, 1)),
115      )
116
117      Returns new ratio parts expression.
118      '',
119      assert all(sequence)
120      assert len(sequence)
121      ratio = []
122      parts = []
123      for i, x in enumerate(sequence):
124          if 0 < x:
125              parts.append(i)
126              ratio.append(abs(x))
127      result = RatioPartsExpression(
128          parts=parts,
129          ratio=ratio,
130      )
131      return result
132
133  ### PUBLIC PROPERTIES ###
134
135  @property
136  def mask_timespan(self):
137      return self._mask_timespan
138
139  @property
140  def parts(self):
141      return self._parts
142
143  @property
144  def ratio(self):
145      return self._ratio

```

## A.39 CONSORTTOOLS.REGISTERINFLECTION

```

1 # -*- encoding: utf-8 -*-
2 import bisect
3 from abjad import new
4 from abjad.tools import abctools
5 from abjad.tools import durationtools
6 from abjad.tools import mathtools
7 from abjad.tools import pitchtools
8 from abjad.tools import sequencetools
9
10
11 class RegisterInflection(abctools.AbjadValueObject):
12     r'''A pitch curve.

```

```

13
14 :::
15
16     >>> import consort
17     >>> register_inflection = consort.RegisterInflection(
18         ...     inflections=(-6, 0, 9),
19         ...     ratio=(2, 1),
20         ... )
21     >>> print(format(register_inflection))
22     consort.tools.RegisterInflection(
23         inflections=pitchtools.IntervalSegment(
24             (
25                 pitchtools.NumberedInterval(-6),
26                 pitchtools.NumberedInterval(0),
27                 pitchtools.NumberedInterval(9),
28                 ),
29                 item_class=pitchtools.NumberedInterval,
30                 ),
31                 ratio=mathtools.Ratio((2, 1)),
32                 )
33
34 :::
35
36     >>> register_inflection(0)
37     NumberedInterval(-6)
38
39 :::
40
41     >>> register_inflection((1, 3))
42     NumberedInterval(-3)
43
44 :::
45
46     >>> register_inflection((1, 2))
47     NumberedInterval(-1)
48
49 :::
50
51     >>> register_inflection((2, 3))
52     NumberedInterval(0)
53
54 :::
55
56     >>> register_inflection(1)
57     NumberedInterval(9)
58
59     """
60
61     ### CLASS VARIABLES ###
62
63     __slots__ = (
64         '_inflections',
65         '_ratio',
66         )

```

```

67
68     """INITIALIZER"""
69
70     def __init__(
71         self,
72         inflections=(0, 0),
73         ratio=(1,),
74     ):
75         if isinstance(inflections, type(self)):
76             expr = inflections
77             self._inflections = expr.inflections
78             self._ratio = expr.ratio
79             return
80         ratio = mathtools.Ratio([abs(x) for x in ratio])
81         self._ratio = ratio
82         inflections = pitchtools.IntervalSegment(
83             inflections,
84             item_class=pitchtools.NumberedInterval,
85         )
86         self._inflections = inflections
87         assert len(inflections) == len(ratio) + 1
88
89     """SPECIAL METHODS"""
90
91     def __call__(self, position):
92         position = durationtools.Offset(position)
93         if position < 0:
94             position = durationtools.Offset(0)
95         if 1 < position:
96             position = durationtools.Offset(1)
97         if position == 0:
98             return self.inflections[0]
99         elif position == 1:
100             return self.inflections[-1]
101         ratio_sum = sum(self.ratio)
102         positions = [durationtools.Offset(x) / ratio_sum
103             for x in mathtools.cumulative_sums(self.ratio)]
104         index = bisect.bisect(positions, position)
105         position = float(position)
106         x0 = float(positions[index - 1])
107         x1 = float(positions[index])
108         y0 = float(self.inflections[index - 1])
109         y1 = float(self.inflections[index])
110         dx = x1 - x0
111         dy = y1 - y0
112         m = float(dy) / float(dx)
113         b = y0 - (m * x0)
114         result = (position * m) + b
115         result = pitchtools.NumberedInterval(int(result))
116         return result
117
118     """PUBLIC METHODS"""
119
120     def align(self):

```

```

121     r'''Aligns all inflections to a minimum interval of zero.
122
123     :::
124
125     >>> import consort
126     >>> inflection = consort.RegisterInflection.zigzag().align()
127     >>> print(format(inflection))
128     consort.tools.RegisterInflection(
129         inflections=pitchtools.IntervalSegment(
130             (
131                 pitchtools.NumberedInterval(0),
132                 pitchtools.NumberedInterval(9),
133                 pitchtools.NumberedInterval(3),
134                 pitchtools.NumberedInterval(12),
135             ),
136             item_class=pitchtools.NumberedInterval,
137         ),
138         ratio=mathtools.Ratio((1, 1, 1)),
139     )
140
141     Emits new register inflection.
142     '''
143     minimum = sorted(self.inflections, key=lambda x: x.semitones)[0]
144     inflections = (_ - minimum for _ in self.inflections)
145     return new(self,
146             inflections=inflections,
147         )
148
149     @staticmethod
150     def ascending(width=12):
151         r'''Creates an ascending register inflection.
152
153         :::
154
155         >>> import consort
156         >>> inflection = consort.RegisterInflection.ascending()
157         >>> print(format(inflection))
158         consort.tools.RegisterInflection(
159             inflections=pitchtools.IntervalSegment(
160                 (
161                     pitchtools.NumberedInterval(-6),
162                     pitchtools.NumberedInterval(6),
163                 ),
164                     item_class=pitchtools.NumberedInterval,
165                 ),
166                 ratio=mathtools.Ratio((1,)),
167             )
168
169         Emits new register inflection.
170         '''
171         import consort
172         half_width = abs(int(width / 2))
173         return consort.RegisterInflection(
174             inflections=(0 - half_width, half_width),

```

```

175         ratio=(1,),
176     )
177
178     @staticmethod
179     def descending(width=12):
180         r'''Creates a descending register inflection.
181
182         :::
183
184         >>> import consort
185         >>> inflection = consort.RegisterInflection.descending()
186         >>> print(format(inflection))
187         consort.tools.RegisterInflection(
188             inflections=pitchtools.IntervalSegment(
189                 (
190                     pitchtools.NumberedInterval(6),
191                     pitchtools.NumberedInterval(-6),
192                     ),
193                     item_class=pitchtools.NumberedInterval,
194                     ),
195                     ratio=mathtools.Ratio((1,)),
196                     )
197
198         Emits new register inflection.
199         '''
200
201         import consort
202         return consort.RegisterInflection.ascending(width=width).invert()
203
204     def invert(self):
205         r'''Inverts register inflection
206
207         :::
208
209         >>> import consort
210         >>> inflection = consort.RegisterInflection.triangle().invert()
211         >>> print(format(inflection))
212         consort.tools.RegisterInflection(
213             inflections=pitchtools.IntervalSegment(
214                 (
215                     pitchtools.NumberedInterval(6),
216                     pitchtools.NumberedInterval(-6),
217                     pitchtools.NumberedInterval(6),
218                     ),
219                     item_class=pitchtools.NumberedInterval,
220                     ),
221                     ratio=mathtools.Ratio((1, 1)),
222                     )
223
224         Emits new register inflection.
225         '''
226
227         return new(self,
228             inflections=(-x for x in self.inflections),
229             )

```

```

229     def reverse(self):
230         r'''Reverses register inflection.
231
232         :::
233
234         >>> import consort
235         >>> inflection = consort.RegisterInflection.zigzag().reverse()
236         >>> print(format(inflection))
237         consort.tools.RegisterInflection(
238             inflections=pitchtools.IntervalSegment(
239                 (
240                     pitchtools.NumberedInterval(6),
241                     pitchtools.NumberedInterval(-3),
242                     pitchtools.NumberedInterval(3),
243                     pitchtools.NumberedInterval(-6),
244                     ),
245                     item_class=pitchtools.NumberedInterval,
246                     ),
247                     ratio=mathtools.Ratio((1, 1, 1)),
248                     )
249
250         Emits new register inflection.
251         '''
252
253         return new(self,
254             inflections=reversed(self.inflections),
255             ratio=reversed(self.ratio),
256             )
257
258     def rotate(self, n=1):
259         r'''Rotates register inflection by ‘n’.
260
261         :::
262
263         >>> import consort
264         >>> inflection = consort.RegisterInflection.zigzag().rotate(1)
265         >>> print(format(inflection))
266         consort.tools.RegisterInflection(
267             inflections=pitchtools.IntervalSegment(
268                 (
269                     pitchtools.NumberedInterval(6),
270                     pitchtools.NumberedInterval(-6),
271                     pitchtools.NumberedInterval(3),
272                     pitchtools.NumberedInterval(-3),
273                     ),
274                     item_class=pitchtools.NumberedInterval,
275                     ),
276                     ratio=mathtools.Ratio((1, 1, 1)),
277                     )
278
279         Emits new register inflection.
280         '''
281
282         return new(self,
283             inflections=sequencetools.rotate_sequence(self.inflections, n),
284             ratio=sequencetools.rotate_sequence(self.ratio, n),

```

```

283         )
284
285     @staticmethod
286     def triangle(width=12):
287         r'''Creates a triangular register inflection.
288
289         :::
290
291         >>> import consort
292         >>> inflection = consort.RegisterInflection.triangle()
293         >>> print(format(inflection))
294         consort.tools.RegisterInflection(
295             inflections=pitchtools.IntervalSegment(
296                 (
297                     pitchtools.NumberedInterval(-6),
298                     pitchtools.NumberedInterval(6),
299                     pitchtools.NumberedInterval(-6),
300                     ),
301                     item_class=pitchtools.NumberedInterval,
302                     ),
303                     ratio=mathtools.Ratio((1, 1)),
304                     )
305
306         :::
307
308         >>> import consort
309         >>> inflection = consort.RegisterInflection.triangle(width=6)
310         >>> print(format(inflection))
311         consort.tools.RegisterInflection(
312             inflections=pitchtools.IntervalSegment(
313                 (
314                     pitchtools.NumberedInterval(-3),
315                     pitchtools.NumberedInterval(3),
316                     pitchtools.NumberedInterval(-3),
317                     ),
318                     item_class=pitchtools.NumberedInterval,
319                     ),
320                     ratio=mathtools.Ratio((1, 1)),
321                     )
322
323         Emits new register inflection.
324         '''
325
326         import consort
327         half_width = int(width / 2)
328         return consort.RegisterInflection(
329             inflections=(-half_width, half_width, -half_width),
330             ratio=(1, 1),
331             )
332
333     @staticmethod
334     def zigzag(width=12):
335         r'''Creates a zigzag register inflection.
336         :::

```

```

337
338     >>> import consort
339     >>> inflection = consort.RegisterInflection.zigzag()
340     >>> print(format(inflection))
341     consort.tools.RegisterInflection(
342         inflections=pitchtools.IntervalSegment(
343             (
344                 pitchtools.NumberedInterval(-6),
345                 pitchtools.NumberedInterval(3),
346                 pitchtools.NumberedInterval(-3),
347                 pitchtools.NumberedInterval(6),
348                 ),
349                 item_class=pitchtools.NumberedInterval,
350                 ),
351                 ratio=mathtools.Ratio((1, 1, 1)),
352             )
353
354     :::
355
356     >>> import consort
357     >>> inflection = consort.RegisterInflection.zigzag(width=8)
358     >>> print(format(inflection))
359     consort.tools.RegisterInflection(
360         inflections=pitchtools.IntervalSegment(
361             (
362                 pitchtools.NumberedInterval(-4),
363                 pitchtools.NumberedInterval(2),
364                 pitchtools.NumberedInterval(-2),
365                 pitchtools.NumberedInterval(4),
366                 ),
367                 item_class=pitchtools.NumberedInterval,
368                 ),
369                 ratio=mathtools.Ratio((1, 1, 1)),
370             )
371
372     Emits new register inflection.
373     """
374     import consort
375     half_width = int(width / 2)
376     quarter_width = int(width / 4)
377     return consort.RegisterInflection(
378         inflections=(
379             -half_width,
380             quarter_width,
381             -quarter_width,
382             half_width,
383             ),
384             ratio=(1, 1, 1),
385         )
386
387     ### PUBLIC PROPERTIES ###
388
389     @property
390     def inflections(self):

```

```

391     return self._inflections
392
393     @property
394     def ratio(self):
395         return self._ratio

```

## A.40 CONSORTTOOLS.REGISTERINFLECTIONINVENTORY

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import datastructuretools
3
4
5 class RegisterInflectionInventory(datastructuretools.TypedList):
6
7     ### CLASS VARIABLES ###
8
9     __slots__ = ()
10
11    ### PRIVATE PROPERTIES ###
12
13    @property
14    def _attribute_manifest(self):
15        from abjad.tools import systemtools
16        return systemtools.AttributeManifest()
17
18    @property
19    def _item_callable(self):
20        import consort
21        return pitchtools.RegisterInflection

```

## A.41 CONSORTTOOLS.REGISTERSPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad import new
4 from abjad.tools import abctools
5 from abjad.tools import pitchtools
6
7
8 class RegisterSpecifier(abctools.AbjadValueObject):
9     r'''A register specifier.
10
11     :::
12
13     >>> import consort
14     >>> register_specifier = consort.RegisterSpecifier(
15     ...     base_pitch=12,
16     ...     division_inflections=(
17     ...         consort.RegisterInflection(
18     ...             inflections=(-6, 3, 6),
19     ...             ratio=(1, 1),
20     ...         ),
21     ...     ),

```

```

22     ...     phrase_inflections=(
23     ...         consort.RegisterInflection(
24     ...             inflections=(3, -3),
25     ...             ratio=(1,),
26     ...             ),
27     ...             ),
28     ...     segment_inflections=(
29     ...         consort.RegisterInflection(
30     ...             inflections=(-12, -9, 0, 12),
31     ...             ratio=(3, 2, 1),
32     ...             ),
33     ...             ),
34     ...         )
35 >>> print(format(registerSpecifier))
36 consort.tools.RegisterSpecifier(
37     base_pitch=pitchtools.NumberedPitch(12),
38     division_inflections=consort.tools.RegisterInflectionInventory(
39         [
40             consort.tools.RegisterInflection(
41                 inflections=pitchtools.IntervalSegment(
42                     (
43                         pitchtools.NumberedInterval(-6),
44                         pitchtools.NumberedInterval(3),
45                         pitchtools.NumberedInterval(6),
46                         ),
47                         item_class=pitchtools.NumberedInterval,
48                         ),
49                         ratio=mathtools.Ratio((1, 1)),
50                         ),
51                     ],
52                 ),
53     phrase_inflections=consort.tools.RegisterInflectionInventory(
54         [
55             consort.tools.RegisterInflection(
56                 inflections=pitchtools.IntervalSegment(
57                     (
58                         pitchtools.NumberedInterval(3),
59                         pitchtools.NumberedInterval(-3),
60                         ),
61                         item_class=pitchtools.NumberedInterval,
62                         ),
63                         ratio=mathtools.Ratio((1,)),
64                         ),
65                     ],
66                 ),
67     segment_inflections=consort.tools.RegisterInflectionInventory(
68         [
69             consort.tools.RegisterInflection(
70                 inflections=pitchtools.IntervalSegment(
71                     (
72                         pitchtools.NumberedInterval(-12),
73                         pitchtools.NumberedInterval(-9),
74                         pitchtools.NumberedInterval(0),
75                         pitchtools.NumberedInterval(12),

```

```

76
77
78
79
80
81
82
83
84
85 :::
86
87     >>> attack_point_signature = consort.AttackPointSignature(
88     ...     division_position=0,
89     ...     phrase_position=(1, 2),
90     ...     segment_position=(4, 5),
91     ... )
92     >>> registerSpecifier.find_register(attack_point_signature)
93     NumberedPitch(6)
94
95     '',
96
97     ### CLASS VARIABLES ###
98
99     __slots__ = (
100         '_base_pitch',
101         '_division_inflections',
102         '_phrase_inflections',
103         '_segment_inflections',
104     )
105
106     ### INITIALIZER ###
107
108     def __init__(
109         self,
110         base_pitch=None,
111         division_inflections=None,
112         phrase_inflections=None,
113         segment_inflections=None,
114     ):
115         from consort.tools import RegisterInflectionInventory
116         if isinstance(base_pitch, type(self)):
117             expr = base_pitch
118             self._base_pitch = expr.base_pitch
119             self._division_inflections = expr.division_inflections
120             self._phrase_inflections = expr.phrase_inflections
121             self._segment_inflections = expr.segment_inflections
122             return
123         if base_pitch is not None:
124             base_pitch = pitchtools.NumberedPitch(base_pitch)
125             self._base_pitch = base_pitch
126         if division_inflections is not None:
127             if not isinstance(division_inflections, collections.Sequence):
128                 division_inflections = [division_inflections]
129             division_inflections = RegisterInflectionInventory(

```

```

130             division_inflections)
131     self._division_inflections = division_inflections
132     if phrase_inflections is not None:
133         if not isinstance(phrase_inflections, collections.Sequence):
134             phrase_inflections = [phrase_inflections]
135             phrase_inflections = RegisterInflectionInventory(
136                 phrase_inflections)
137     self._phrase_inflections = phrase_inflections
138     if segment_inflections is not None:
139         if not isinstance(segment_inflections, collections.Sequence):
140             segment_inflections = [segment_inflections]
141             segment_inflections = RegisterInflectionInventory(
142                 segment_inflections)
143     self._segment_inflections = segment_inflections
144
145     ### PUBLIC METHODS ###
146
147     def find_register(
148         self,
149         attack_point_signature,
150         seed=0,
151     ):
152         division_position = attack_point_signature.division_position
153         phrase_position = attack_point_signature.phrase_position
154         segment_position = attack_point_signature.segment_position
155         seed = int(seed)
156         register = self.base_pitch
157         if register is None:
158             register = pitchtools.NumberedPitch(0)
159         if self.division_inflections:
160             index = seed % len(self.division_inflections)
161             inflection = self.division_inflections[index]
162             deviation = inflection(division_position)
163             register = register.transpose(deviation)
164         if self.phrase_inflections:
165             index = seed % len(self.phrase_inflections)
166             inflection = self.phrase_inflections[index]
167             deviation = inflection(phrase_position)
168             register = register.transpose(deviation)
169         if self.segment_inflections:
170             index = seed % len(self.segment_inflections)
171             inflection = self.segment_inflections[index]
172             deviation = inflection(segment_position)
173             register = register.transpose(deviation)
174         return register
175
176     def transpose(self, transposition):
177         r'''Transposes register specifier.
178
179         :::
180
181         >>> registerSpecifier = consort.RegisterSpecifier(
182             ...     base_pitch=12,
183             ...     division_inflections=

```

```

184     ...
185         consort.RegisterInflection(
186             ...
187                 inflections=(-6, 3, 6),
188                 ratio=(1, 1),
189             ),
190             ...
191                 phrase_inflections=(
192                     ...
193                         consort.RegisterInflection(
194                             ...
195                                 inflections=(3, -3),
196                                 ratio=(1, ),
197                             ),
198                             ...
199                                 ),
200                             ...
201                         ),
202             ...
203             ...
204             ...
205             ...
206             ...
207             ...
208             ...
209             ...
210             ...
211             ...
212             ...
213             ...
214             ...
215             ...
216             ...
217             ...
218             ...
219             ...
220             ...
221             ...
222             ...
223             ...
224             ...
225             ...
226             ...
227             ...
228             ...
229             ...
230             ...
231             ...
232             ...
233             ...
234             ...
235             ...
236             ...
237             ...

```

```

238         inflections=pitchtools.IntervalSegment(
239             (
240                 pitchtools.NumberedInterval(-12),
241                 pitchtools.NumberedInterval(-9),
242                 pitchtools.NumberedInterval(0),
243                 pitchtools.NumberedInterval(12),
244             ),
245             item_class=pitchtools.NumberedInterval,
246             ),
247             ratio=mathtools.Ratio((3, 2, 1)),
248             ),
249         ],
250     ),
251 )
252 """
253 base_pitch = self.base_pitch or pitchtools.NamedPitch('C4')
254 base_pitch = base_pitch.transpose(transposition)
255 return new(self, base_pitch=base_pitch)
256
257 ### PUBLIC PROPERTIES ###
258
259 @property
260 def base_pitch(self):
261     return self._base_pitch
262
263 @property
264 def division_inflections(self):
265     return self._division_inflections
266
267 @property
268 def phrase_inflections(self):
269     return self._phrase_inflections
270
271 @property
272 def segment_inflections(self):
273     return self._segment_inflections
274

```

## A.42 CONSORTTOOLS.REGISTERSPECIFIERINVENTORY

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import datastructuretools
3
4
5 class RegisterSpecifierInventory(datastructuretools.TypedList):
6
7     """ CLASS VARIABLES """
8
9     __slots__ = ()
10
11    """ PRIVATE PROPERTIES """
12
13    @property
14    def _attribute_manifest(self):

```

```

15     from abjad.tools import systemtools
16     return systemtools.AttributeManifest()
17
18     @property
19     def _item_callable(self):
20         import consort
21         return consort.RegisterSpecifier

```

## A.43 CONSORTTOOLS.SCORETEMPLATE

```

1 # -*- encoding: utf-8 -*-
2 import abc
3 from abjad import attach
4 from abjad.tools import abctools
5 from abjad.tools import indicatortools
6 from abjad.tools import instrumenttools
7 from abjad.tools import scoretools
8 from abjad.tools import stringtools
9
10
11 class ScoreTemplate(abctools.AbjadValueObject):
12
13     ### CLASS VARIABLES ###
14
15     __slots__ = (
16         '_composite_context_pairs',
17         '_context_name_abbreviations',
18     )
19
20     _is_populated = False
21
22     ### INITIALIZER ###
23
24     def __init__(self):
25         self._context_name_abbreviations = {}
26         self._composite_context_pairs = {}
27
28         if not type(self)._is_populated:
29             self()
30             type(self)._is_populated = True
31
32     ### SPECIAL METHODS ###
33
34     @abc.abstractmethod
35     def __call__(self):
36         raise NotImplementedError
37
38     ### PRIVATE METHODS ###
39
40     def _attach_tag(self, label, context):
41         label = stringtools.to_dash_case(label)
42         tag = indicatortools.LilyPondCommand(
43             name="tag #'{}".format(label),
44             format_slot='before',

```

```

45         )
46     attach(tag, context)
47
48     def _make_voice(self, name, abbreviation=None, context_name=None):
49         name = name.title()
50         abbreviation = abbreviation or name
51         abbreviation = stringtools.to_snake_case(abbreviation)
52         voice_name = '{} Voice'.format(name)
53         voice = scoretools.Voice(
54             name=voice_name,
55             context_name=context_name,
56             )
57         self._context_nameAbbreviations[abbreviation] = voice.name
58         return voice
59
60     def _make_staff(
61         self,
62         name,
63         clef,
64         abbreviation=None,
65         context_name=None,
66         instrument=None,
67         tag=None,
68         ):
69         name = name.title()
70         staff_name = '{} Staff'.format(name)
71         context_name = context_name or staff_name
72         context_name = context_name.replace(' ', '')
73         abbreviation = abbreviation or name
74         abbreviation = stringtools.to_snake_case(abbreviation)
75         voice = self._make_voice(name, abbreviation=abbreviation)
76         staff = scoretools.Staff(
77             [voice],
78             context_name=context_name,
79             name=staff_name
80             )
81         if not isinstance(clef, indicatortools.Clef):
82             clef = indicatortools.Clef(clef)
83         attach(clef, staff)
84         if tag:
85             self._attach_tag(tag, staff)
86         if instrument:
87             assert isinstance(instrument, instrumenttools.Instrument)
88             attach(instrument, staff)
89         return staff
90
91     def _populate(self):
92         if not type(self).is_populated:
93             self()
94             type(self).is_populated = True
95
96     ### PUBLIC PROPERTIES ###
97
98     @property

```

```

99     def all_voice_names(self):
100         self._populate()
101         result = []
102         for name in self.context_name_abbreviations:
103             if name in self.composite_context_pairs:
104                 continue
105             result.append(name)
106         return tuple(result)
107
108     @property
109     def composite_context_pairs(self):
110         self._populate()
111         return self._composite_context_pairs
112
113     @property
114     def context_name_abbreviations(self):
115         self._populate()
116         return self._context_name_abbreviations

```

#### A.44 CONSORTTOOLS.SCORETEMPLATEMANAGER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import attach
3 from abjad import set_
4 from abjad.tools import abctools
5 from abjad.tools import indicatortools
6 from abjad.tools import markuptools
7 from abjad.tools import scoretools
8 from abjad.tools import stringtools
9
10
11 class ScoreTemplateManager(abctools.AbjadObject):
12
13     """PUBLIC METHODS"""
14
15     @staticmethod
16     def attach_tag(label, context):
17         label = stringtools.to_dash_case(label)
18         tag = indicatortools.LilyPondCommand(
19             name="tag #'{}".format(label),
20             format_slot='before',
21         )
22         attach(tag, context)
23
24     @staticmethod
25     def make_auxiliary_staff(
26         primary_instrument=None,
27         secondary_instrument=None,
28         score_template=None,
29     ):
30         name = '{} {}'.format(
31             primary_instrument.instrument_name.title(),
32             secondary_instrument.instrument_name.title(),
33         )

```

```

34     voice = scoretools.Voice(
35         name='{} Voice'.format(name),
36         )
37     context_name = ScoreTemplateManager.make_staff_name(
38         secondary_instrument.instrument_name.title(),
39         )
40     staff = scoretools.Staff(
41         [voice],
42         name='{} Staff'.format(name),
43         context_name=context_name,
44         )
45     abbreviation = stringtools.to_snake_case(name)
46     score_template._context_name_abbreviations[abbreviation] = voice.name
47     return staff
48
49     @staticmethod
50     def make_column_markup(string, space):
51         string_parts = string.split()
52         if len(string_parts) == 1:
53             markup = markuptools.Markup(string_parts[0]).hcenter_in(space)
54         else:
55             markups = [markuptools.Markup(_) for _ in string_parts]
56             markup = markuptools.Markup.center_column(markups, direction=None)
57             markup = markup.hcenter_in(space)
58         return markup
59
60     @staticmethod
61     def make_ensemble_group(
62         label=None,
63         name=None,
64         performer_groups=None,
65         ):
66         ensemble_group = scoretools.StaffGroup(
67             performer_groups,
68             name=name,
69             context_name='EnsembleGroup',
70             )
71         if label is not None:
72             ScoreTemplateManager.attach_tag(label, ensemble_group)
73         return ensemble_group
74
75     @staticmethod
76     def make_performer_group(
77         context_name=None,
78         instrument=None,
79         label=None,
80         ):
81         context_name = context_name or 'PerformerGroup'
82         name = '{} Performer Group'.format(instrument.instrument_name.title())
83         performer_group = scoretools.StaffGroup(
84             context_name=context_name,
85             name=name,
86             )
87         performer_group.is_simultaneous = True

```

```

88     if label is not None:
89         ScoreTemplateManager.attach_tag(label, performer_group)
90     attach(
91         instrument,
92         performer_group,
93         scope=context_name,
94         is_annotation=True,
95     )
96     manager = set_(performer_group)
97     manager.instrument_name = instrument.instrument_name_markup
98     manager.short_instrument_name = instrument.short_instrument_name_markup
99     return performer_group
100
101 @staticmethod
102 def make_single_basic_performer(
103     clef=None,
104     context_name=None,
105     instrument=None,
106     label=None,
107     score_template=None,
108 ):
109     performer_group = ScoreTemplateManager.make_performer_group(
110         context_name=context_name,
111         instrument=instrument,
112         label=label,
113     )
114     name = instrument.instrument_name.title()
115     context_name = ScoreTemplateManager.make_staff_name(name)
116     voice = scoretools.Voice(
117         name='{} Voice'.format(name),
118     )
119     staff = scoretools.Staff(
120         [voice],
121         context_name=context_name,
122         name='{} Staff'.format(name),
123     )
124     performer_group.append(staff)
125     attach(clef, voice)
126     abbreviation = stringtools.to_snake_case(name)
127     score_template._context_name_abbreviations[abbreviation] = voice.name
128     return performer_group
129
130 @staticmethod
131 def make_single_piano_performer(
132     instrument=None,
133     score_template=None,
134 ):
135     performer_group = ScoreTemplateManager.make_performer_group(
136         context_name='PianoStaff',
137         instrument=instrument,
138         label=stringtools.to_dash_case(instrument.instrument_name),
139     )
140     name = instrument.instrument_name.title()
141     upper_voice = scoretools.Voice(

```

```

142         name='{} Upper Voice'.format(name),
143     )
144     upper_staff = scoretools.Staff(
145         [upper_voice],
146         context_name='PianoUpperStaff',
147         name='{} Upper Staff'.format(name),
148     )
149     dynamics = scoretools.Voice(
150         context_name='Dynamics',
151         name='{} Dynamics'.format(name),
152     )
153     lower_voice = scoretools.Voice(
154         name='{} Lower Voice'.format(name),
155     )
156     lower_staff = scoretools.Staff(
157         [lower_voice],
158         context_name='PianoLowerStaff',
159         name='{} Lower Staff'.format(name),
160     )
161     pedals = scoretools.Voice(
162         context_name='Dynamics',
163         name='{} Pedals'.format(name),
164     )
165     performer_group.extend((
166         upper_staff,
167         dynamics,
168         lower_staff,
169         pedals,
170     ))
171     attach(indicatortools.Clef('treble'), upper_voice)
172     attach(indicatortools.Clef('bass'), lower_voice)
173     score_template._context_nameAbbreviations.update(
174         piano_dynamics=dynamics.name,
175         piano_lh=lower_voice.name,
176         piano_pedals=pedals.name,
177         piano_rh=upper_voice.name,
178     )
179     return performer_group
180
181     @staticmethod
182     def make_single_string_performer(
183         abbreviation=None,
184         clef=None,
185         instrument=None,
186         score_template=None,
187         split=True,
188     ):
189         performer_group = ScoreTemplateManager.make_performer_group(
190             context_name='StringPerformerGroup',
191             instrument=instrument,
192             label=stringtools.to_dash_case(instrument.instrument_name),
193         )
194         name = instrument.instrument_name.title()
195         abbreviation = abbreviation or \

```

```

196     stringtools.to_snake_case(name)
197     if split:
198         right_hand_voice = scoretools.Voice(
199             name='{} Bowing Voice'.format(name),
200             )
201         right_hand_staff = scoretools.Staff(
202             [right_hand_voice],
203             context_name='BowingStaff',
204             name='{} Bowing Staff'.format(name),
205             )
206         left_hand_voice = scoretools.Voice(
207             name='{} Fingering Voice'.format(name),
208             )
209         left_hand_staff = scoretools.Staff(
210             [left_hand_voice],
211             context_name='FingeringStaff',
212             name='{} Fingering Staff'.format(name),
213             )
214         performer_group.append(right_hand_staff)
215         performer_group.append(left_hand_staff)
216         attach(clef, left_hand_staff)
217         attach(indicatortools.Clef('percussion'), right_hand_staff)
218         right_hand_abbreviation = '{}_rh'.format(abbreviation)
219         left_hand_abbreviation = '{}_lh'.format(abbreviation)
220         score_template._context_name_abbreviations.update({
221             abbreviation: performer_group.name,
222             right_hand_abbreviation: right_hand_voice.name,
223             left_hand_abbreviation: left_hand_voice.name,
224             })
225         score_template._composite_context_pairs[abbreviation] = (
226             right_hand_abbreviation,
227             left_hand_abbreviation,
228             )
229     else:
230         voice = scoretools.Voice(
231             name='{} Voice'.format(name),
232             )
233         staff = scoretools.Staff(
234             [voice],
235             context_name='StringStaff',
236             name='{} Staff'.format(name),
237             )
238         performer_group.append(staff)
239         attach(clef, voice)
240         score_template._context_name_abbreviations[abbreviation] = \
241             voice.name
242     return performer_group
243
244     @staticmethod
245     def make_single_wind_performer(
246         abbreviation=None,
247         clef=None,
248         instrument=None,
249         score_template=None,

```

```

250     ):
251     performer_group = ScoreTemplateManager.make_performer_group(
252         instrument=instrument,
253         label=stringtools.to_dash_case(instrument.instrument_name),
254         )
255     name = instrument.instrument_name.title()
256     context_name = ScoreTemplateManager.make_staff_name(name)
257     voice = scoretools.Voice(
258         name='{} Voice'.format(name),
259         )
260     staff = scoretools.Staff(
261         [voice],
262         context_name=context_name,
263         name='{} Staff'.format(name),
264         )
265     performer_group.append(staff)
266     attach(clef, voice)
267     abbreviation = abbreviation or \
268         stringtools.to_snake_case(name)
269     score_template._context_name_abbreviations[abbreviation] = voice.name
270     return performer_group
271
272     @staticmethod
273     def make_staff_name(name):
274         name = ''.join(x for x in name if x.isalpha())
275         name = '{}Staff'.format(name)
276         return name
277
278     @staticmethod
279     def make_voice_name(name):
280         name = ''.join(x for x in name if x.isalpha())
281         name = '{}Voice'.format(name)
282         return name
283
284     @staticmethod
285     def make_time_signature_context():
286         time_signature_context = scoretools.Context(
287             context_name='TimeSignatureContext',
288             name='Time Signature Context',
289             )
290         label = 'time'
291         ScoreTemplateManager.attach_tag(label, time_signature_context)
292         return time_signature_context

```

## A.45 CONSORTTOOLS.SEEDSESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import abctools
3
4
5 class SeedSession(abctools.AbjadObject):
6
7     ### CLASS VARIABLES ###
8

```

```

9     __slots__ = (
10        '_current_timewise_logical_tie_seed',
11        '_current_timewise_phrase_seed',
12        '_current_phrased_voicewise_logical_tie_seed',
13        '_current_unphrased_voicewise_logical_tie_seed',
14        '_current_timewise_music_specifier_seed',
15        '_timewise_logical_tie_seeds',
16        '_timewise_music_specifier_seeds',
17        '_timewise_phrase_seeds',
18        '_phrased_voicewise_logical_tie_seeds',
19        '_unphrased_voicewise_logical_tie_seeds',
20    )
21
22    ### INITIALIZER ###
23
24    def __init__(self):
25        self._current_timewise_logical_tie_seed = 0
26        self._current_timewise_phrase_seed = 0
27        self._current_phrased_voicewise_logical_tie_seed = 0
28        self._current_timewise_music_specifier_seed = 0
29        self._timewise_music_specifier_seeds = {}
30        self._timewise_logical_tie_seeds = {}
31        self._timewise_phrase_seeds = {}
32        self._phrased_voicewise_logical_tie_seeds = {}
33        self._unphrased_voicewise_logical_tie_seeds = {}
34
35    ### SPECIAL METHODS ###
36
37    def __call__(
38        self,
39        application_rate,
40        attack_point_signature,
41        music_specifier,
42        voice,
43    ):
44        phrased_voicewise_logical_tie_seed = self._get_phrased_voicewise_logical_tie_seed(
45            attack_point_signature,
46            music_specifier,
47            application_rate,
48            voice,
49        )
50        unphrased_voicewise_logical_tie_seed = self._get_unphrased_voicewise_logical_tie_seed(
51            music_specifier,
52            voice,
53        )
54        timewise_phrase_seed = self._get_timewise_phrase_seed(
55            attack_point_signature,
56            music_specifier,
57            voice,
58        )
59        timewise_music_specifier_seed = \
60            self._get_timewise_music_specifier_seed(
61                music_specifier,
62            )

```

```

63     self._current_timewise_phrase_seed = timewise_phrase_seed
64     self._current_phrased_voicewise_logical_tie_seed = \
65         phrased_voicewise_logical_tie_seed
66     self._current_unphrased_voicewise_logical_tie_seed = \
67         unphrased_voicewise_logical_tie_seed
68     self._current_timewise_music_specifier_seed = \
69         timewise_music_specifier_seed
70
71     ### PRIVATE METHODS ###
72
73     def _get_timewise_music_specifier_seed(
74         self,
75         music_specifier,
76     ):
77         if music_specifier not in self._timewise_music_specifier_seeds:
78             self._timewise_music_specifier_seeds[music_specifier] = 0
79         seed = self._timewise_music_specifier_seeds[music_specifier]
80         self._timewise_music_specifier_seeds[music_specifier] += 1
81         return seed
82
83     def _get_timewise_phrase_seed(
84         self,
85         attack_point_signature,
86         music_specifier,
87         voice,
88     ):
89         key = (voice, music_specifier)
90         if attack_point_signature.is_first_of_phrase:
91             if key not in self._timewise_phrase_seeds:
92                 phrase_seed = (music_specifier.seed or 0) - 1
93                 self._timewise_phrase_seeds[key] = phrase_seed
94             self._timewise_phrase_seeds[key] += 1
95         phrase_seed = self._timewise_phrase_seeds[key]
96         return phrase_seed
97
98     def _get_phrased_voicewise_logical_tie_seed(
99         self,
100        attack_point_signature,
101        music_specifier,
102        application_rate,
103        voice,
104    ):
105        if music_specifier not in self._timewise_logical_tie_seeds:
106            seed = (music_specifier.seed or 0) - 1
107            self._timewise_logical_tie_seeds[music_specifier] = seed
108            self._phrased_voicewise_logical_tie_seeds[voice] = seed
109        if application_rate == 'phrase':
110            if attack_point_signature.is_first_of_phrase:
111                self._timewise_logical_tie_seeds[music_specifier] += 1
112                seed = self._timewise_logical_tie_seeds[music_specifier]
113                self._phrased_voicewise_logical_tie_seeds[voice] = seed
114            else:
115                seed = self._phrased_voicewise_logical_tie_seeds[voice]
116        elif application_rate == 'division':

```

```

117     if attack_point_signature.is_first_of_division:
118         self._timewise_logical_tie_seeds[music_specifier] += 1
119         seed = self._timewise_logical_tie_seeds[music_specifier]
120         self._phrased_voicewise_logical_tie_seeds[voice] = seed
121     else:
122         seed = self._phrased_voicewise_logical_tie_seeds[voice]
123     else:
124         self._timewise_logical_tie_seeds[music_specifier] += 1
125         seed = self._timewise_logical_tie_seeds[music_specifier]
126     return seed
127
128 def _get_unphrased_voicewise_logical_tie_seed(
129     self,
130     music_specifier,
131     voice,
132 ):
133     if voice not in self._unphrased_voicewise_logical_tie_seeds:
134         self._unphrased_voicewise_logical_tie_seeds[voice] = {}
135     if music_specifier not in self._unphrased_voicewise_logical_tie_seeds[voice]:
136         self._unphrased_voicewise_logical_tie_seeds[voice][music_specifier] = \
137             (music_specifier.seed or 0) - 1
138     self._unphrased_voicewise_logical_tie_seeds[voice][music_specifier] += 1
139     return self._unphrased_voicewise_logical_tie_seeds[voice][music_specifier]
140
141 ### PUBLIC PROPERTIES ###
142
143 @property
144 def current_timewise_music_specifier_seed(self):
145     return self._current_timewise_music_specifier_seed
146
147 @property
148 def current_timewise_phrase_seed(self):
149     return self._current_timewise_phrase_seed
150
151 @property
152 def current_phrased_voicewise_logical_tie_seed(self):
153     return self._current_phrased_voicewise_logical_tie_seed
154
155 @property
156 def current_unphrased_voicewise_logical_tie_seed(self):
157     return self._current_unphrased_voicewise_logical_tie_seed

```

## A.46 CONSORTTOOLS.SEGMENTMAKER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import collections
4 import importlib
5 import itertools
6 import os
7 from abjad import attach
8 from abjad import detach
9 from abjad import inspect_
10 from abjad import iterate

```

```

11 from abjad import mutate
12 from abjad import override
13 from abjad import new
14 from abjad import set_
15 from abjad.tools import datastructuretools
16 from abjad.tools import durationtools
17 from abjad.tools import indicatortools
18 from abjad.tools import instrumenttools
19 from abjad.tools import lilypondfiletools
20 from abjad.tools import markuptools
21 from abjad.tools import mathtools
22 from abjad.tools import metertools
23 from abjad.tools import rhythmmakertools
24 from abjad.tools import scoretools
25 from abjad.tools import selectiontools
26 from abjad.tools import spannertools
27 from abjad.tools import systemtools
28 from abjad.tools import timespan-tools
29 from experimental.tools import makertools
30
31
32 class SegmentMaker(makertools.SegmentMaker):
33     r'''A Consort segment-maker.
34
35     :::
36
37     >>> import consort
38     >>> score_template = templatetools.StringOrchestraScoreTemplate(
39         ...     violin_count=2,
40         ...     viola_count=1,
41         ...     cello_count=1,
42         ...     contrabass_count=0,
43         ... )
44
45     :::
46
47     >>> segment_maker = consort.SegmentMaker(
48         ...     score_template=score_template,
49         ...     settings=(
50             ...         consort.MusicSetting(
51                 ...             timespan_maker=consort.TaleaTimespanMaker(),
52                 ...             violin_1_bowing_voice=consort.MusicSpecifier(),
53                 ...             violin_2_bowing_voice=consort.MusicSpecifier(),
54                 ...             ),
55                 ...             ),
56                 ...             desired_duration_in_seconds=2,
57                 ...             tempo=indicatortools.Tempo((1, 4), 72),
58                 ...             permitted_time_signatures=(
59                     ...             (5, 8),
60                     ...             (7, 16),
61                     ...             ),
62                     ...             )
63     >>> print(format(segment_maker))
64     consort.tools.SegmentMaker(

```

```

65     desired_duration_in_seconds=durationtools.Duration(2, 1),
66     permitted_time_signatures=indicatortools.TimeSignatureInventory(
67         [
68             indicatortools.TimeSignature((5, 8)),
69             indicatortools.TimeSignature((7, 16)),
70         ]
71     ),
72     score_template=templatetools.StringOrchestraScoreTemplate(
73         violin_count=2,
74         viola_count=1,
75         cello_count=1,
76         contrabass_count=0,
77         split_hands=True,
78         use_percussion_clefs=False,
79     ),
80     settings=(
81         consort.tools.MusicSetting(
82             timespan_maker=consort.tools.TaleaTimespanMaker(
83                 playing_talea=rhythmmakertools.Talea(
84                     counts=(4,),
85                     denominator=16,
86                 ),
87                 playing_groupings=(1,),
88                 repeat=True,
89                 silence_talea=rhythmmakertools.Talea(
90                     counts=(4,),
91                     denominator=16,
92                 ),
93                 step_anchor=Right,
94                 synchronize_groupings=False,
95                 synchronize_step=False,
96             ),
97             violin_1_bowing_voice=consort.tools.MusicSpecifier(),
98             violin_2_bowing_voice=consort.tools.MusicSpecifier(),
99         ),
100    ),
101    tempo=indicatortools.Tempo(
102        duration=durationtools.Duration(1, 4),
103        units_per_minute=72,
104    ),
105),
106
107 :::
108
109 >>> lilypond_file = segment_maker() # doctest: +SKIP
110 Performing rhythmic interpretation:
111     populating independent timespans:
112         populated timespans: ...
113         found meters: ...
114         demultiplexed timespans: ...
115         split timespans: ...
116         pruned malformed timespans: ...
117         consolidated timespans: ...
118         inscribed timespans: ...

```

```

119     multiplexed timespans: ...
120     pruned short timespans: ...
121     pruned meters: ...
122     total: ...
123     populating dependent timespans:
124         populated timespans: ...
125         demultiplexed timespans: ...
126         split timespans: ...
127         pruned short timespans: ...
128         pruned malformed timespans: ...
129         consolidated timespans: ...
130         inscribed timespans: ...
131         total: ...
132         populated silent timespans: ...
133         validated timespans: ...
134     rewriting meters:
135         rewriting Cello Bowing Voice: 2
136         rewriting Cello Fingering Voice: 2
137         rewriting Viola Bowing Voice: 2
138         rewriting Viola Fingering Voice: 2
139         rewriting Violin 1 Bowing Voice: 3
140         rewriting Violin 1 Fingering Voice: 2
141         rewriting Violin 2 Bowing Voice: 3
142         rewriting Violin 2 Fingering Voice: 2
143         total: 0.169489145279
144     populated score: ...
145     total: ...
146     Performing non-rhythmic interpretation:
147         collected attack points: ...
148         handled graces: ...
149         handled pitches: ...
150         handled attachments: ...
151         total: ...
152     Checking for well-formedness violations:
153         [] 24 check_beamed_quarter_notes
154         [] 18 check_discontiguous_spanners
155         [] 80 check_duplicate_ids
156         [] 0 check_intermarked_hairpins
157         [] 2 check_misdurated_measures
158         [] 2 check_misfilled_measures
159         [] 4 check_mispitched_ties
160         [] 24 check_misrepresented_flags
161         [] 80 check_missing_parents
162         [] 2 check_nested_measures
163         [] 0 check_overlapping_beams
164         [] 0 check_overlapping_glissandi
165         [] 0 check_overlapping_octavation_spanners
166         [] 0 check_short_hairpins
167         total: ...
168     """
169
170     ### CLASS VARIABLES ###
171
172

```

```

173     __slots__ = (
174         '_annotate_colors',
175         '_annotate_phrasing',
176         '_annotate_timespans',
177         '_attack_point_map',
178         '_desired_duration_in_seconds',
179         '_discard_final_silence',
180         '_lilypond_file',
181         '_maximum_meter_run_length',
182         '_meters',
183         '_name',
184         '_omit_stylesheets',
185         '_permitted_time_signatures',
186         '_previous_segment_metadata',
187         '_repeat',
188         '_score',
189         '_score_template',
190         '_segment_metadata',
191         '_settings',
192         '_tempo',
193         '_timespan_quantization',
194         '_voice_names',
195         '_voicewise_timespans',
196     )
197
198     ### INITIALIZER ###
199
200     def __init__(
201         self,
202         annotate_colors=None,
203         annotate_phrasing=None,
204         annotate_timespans=None,
205         desired_duration_in_seconds=None,
206         discard_final_silence=None,
207         maximum_meter_run_length=None,
208         name=None,
209         omit_stylesheets=None,
210         permitted_time_signatures=None,
211         repeat=None,
212         score_template=None,
213         settings=None,
214         tempo=None,
215         timespan_quantization=None,
216     ):
217         makertools.SegmentMaker.__init__(
218             self,
219         )
220         self.name = name
221         self.annotate_colors = annotate_colors
222         self.annotate_phrasing = annotate_phrasing
223         self.annotate_timespans = annotate_timespans
224         self.discard_final_silence = discard_final_silence
225         self.desired_duration_in_seconds = desired_duration_in_seconds
226         self.maximum_meter_run_length = maximum_meter_run_length

```

```

227     self.omit_stylesheets = omit_stylesheets
228     self.permitted_time_signatures = permitted_time_signatures
229     self.score_template = score_template
230     self.tempo = tempo
231     self.timespan_quantization = timespan_quantization
232     self.settings = settings
233     self.repeat = repeat
234     self._reset()
235
236     ### SPECIAL METHODS ###
237
238     def __call__(
239         self,
240         annotate=None,
241         verbose=True,
242         segment_metadata=None,
243         previous_segment_metadata=None,
244     ):
245         import consort
246         self._reset()
247
248         self._annotate_phrasing = self._annotate_phrasing or annotate
249         self._segment_metadata = segment_metadata or \
250             datastructuretools.TypedOrderedDict()
251         self._previous_segment_metadata = previous_segment_metadata or \
252             datastructuretools.TypedOrderedDict()
253         self._score = self.score_template()
254         self._voice_names = tuple(
255             voice.name for voice in
256             iterate(self.score).by_class(scoretools.Voice)
257         )
258
259         with systemtools.Timer(
260             '    total:',
261             'Performing rhythmic interpretation:',
262             verbose=verbose,
263         ):
264             self.interpret_rhythms(verbose=verbose)
265         with systemtools.Timer(
266             '    total:',
267             'Performing non-rhythmic interpretation:',
268             verbose=verbose,
269         ):
270             with systemtools.Timer(
271                 '        collected attack points:',
272                 verbose=verbose,
273             ):
274                 attack_point_map = self.collect_attack_points(self.score)
275             self._attack_point_map = attack_point_map
276             with systemtools.ForbidUpdate(self.score, update_on_exit=True):
277                 with systemtools.Timer(
278                     '        handled graces:',
279                     verbose=verbose,
280                 ):

```

```

281         consort.GraceHandler._process_session(self)
282     with systemtools.ForbidUpdate(self.score, update_on_exit=True):
283         with systemtools.Timer(
284             '      total:',
285             '      handling pitches:',
286             verbose=verbose,
287         ):
288             consort.PitchHandler._process_session(self)
289     with systemtools.ForbidUpdate(self.score, update_on_exit=True):
290         with systemtools.Timer(
291             '      total:',
292             '      handling attachments:',
293             verbose=verbose,
294         ):
295             consort.AttachmentHandler._process_session(
296                 self,
297                 verbose=verbose,
298             )
299             self.configure_score()
300             self.configure_lilypond_file()
301     with systemtools.Timer(
302         enter_message='Checking for well-formedness violations:',
303         exit_message='      total:',
304         verbose=verbose,
305     ):
306         self.validate_score(self.score, verbose=verbose)
307
308     self.update_segment_metadata()
309
310     return self.lilypond_file, self._segment_metadata
311
312     ### PRIVATE METHODS ###
313
314     def _reset(self):
315         self._attack_point_map = None
316         self._lilypond_file = None
317         self._meters = None
318         self._score = None
319         self._voice_names = None
320         self._voicewise_timesteps = None
321         self._segment_metadata = None
322         self._previous_segment_metadata = None
323
324     ### PRIVATE PROPERTIES ###
325
326     @property
327     def _storage_format_specification(self):
328         from abjad.tools import systemtools
329         manager = systemtools.StorageFormatManager
330         keyword_argument_names = manager.get_keyword_argument_names(self)
331         keyword_argument_names = list(keyword_argument_names)
332         if not self.settings:
333             keyword_argument_names.remove('settings')
334         return systemtools.StorageFormatSpecification(

```

```

335         self,
336         keyword_argument_names=keyword_argument_names
337     )
338
339     ### PUBLIC METHODS ###
340
341     def get_end_instruments(self):
342         result = datastructuretools.TypedOrderedDict()
343         staves = iterate(self._score).by_class(scoretools.Staff)
344         staves = list(staves)
345         staves.sort(key=lambda x: x.name)
346         prototype = (instrumenttools.Instrument,)
347         for staff in staves:
348             last_leaf = inspect_(staff).get_leaf(-1)
349             instrument = inspect_(last_leaf).get_effective(prototype)
350             if instrument:
351                 result[staff.name] = instrument.instrument_name
352             else:
353                 result[staff.name] = None
354
355         return result
356
357     def get_end_tempo_indication(self):
358         prototype = indicatortools.Tempo
359         context = self._score['Time Signature Context']
360         last_leaf = inspect_(context).get_leaf(-1)
361         effective_tempo = inspect_(last_leaf).get_effective(prototype)
362         if effective_tempo is not None:
363             duration = effective_tempo.duration.pair
364             units_per_minute = effective_tempo.units_per_minute
365             effective_tempo = (duration, units_per_minute)
366
367         return effective_tempo
368
369     def get_end_time_signature(self):
370         prototype = indicatortools.TimeSignature
371         context = self._score['Time Signature Context']
372         last_measure = context[-1]
373         time_signature = inspect_(last_measure).get_effective(prototype)
374         if not time_signature:
375             return
376         pair = time_signature.pair
377         return pair
378
379     def add_time_signature_context(self):
380         import consort
381         if 'Time Signature Context' not in self.score:
382             time_signature_context = \
383                 consort.ScoreTemplateManager.make_time_signature_context()
384             self.score.insert(0, time_signature_context)
385         context = self.score['Time Signature Context']
386         time_signatures = [_.implied_time_signature for _ in self.meters]
387         iterator = itertools.groupby(time_signatures, lambda x: x)
388         measures = []
389         for time_signature, group in iterator:
390             count = len(tuple(group))

```

```

389     skip = scoretools.Skip(1)
390     multiplier = durationtools.Multiplier(time_signature) * count
391     attach(multiplier, skip)
392     attach(time_signature, skip, scope=scoretools.Score)
393     measure = scoretools.Container([skip])
394     measures.append(measure)
395     context.extend(measures)
396
397     def add_setting(
398         self,
399         silenced_contexts=None,
400         timespan_identifier=None,
401         timespan_maker=None,
402         **music_specifiers
403     ):
404         import consort
405         setting = consort.MusicSetting(
406             silenced_contexts=silenced_contexts,
407             timespan_identifier=timespan_identifier,
408             timespan_maker=timespan_maker,
409             **music_specifiers
410         )
411         self._settings.append(setting)
412
413     def attach_initial_bar_line(self):
414         segment_number = self._segment_metadata.get('segment_number', 1)
415         if self.repeat:
416             if segment_number != 1:
417                 command = indicatortools.LilyPondCommand('break', 'opening')
418                 attach(command, self.score['Time Signature Context'])
419             return
420         elif self._previous_segment_metadata.get('is_repeated'):
421             return
422         elif segment_number == 1:
423             return
424         bar_line = indicatortools.LilyPondCommand('bar "||"', 'opening')
425         for staff in iterate(self.score).by_class(scoretools.Staff):
426             attach(bar_line, staff)
427
428     def attach_final_bar_line(self):
429         segment_number = self._segment_metadata.get('segment_number', 1)
430         segment_count = self._segment_metadata.get('segment_count', 1)
431         if self.repeat:
432             repeat = indicatortools.Repeat()
433             for staff in iterate(self.score).by_class(scoretools.Staff):
434                 attach(repeat, staff)
435                 attach(repeat, self.score['Time Signature Context'])
436         elif segment_number == segment_count:
437             self.score.add_final_bar_line(
438                 abbreviation='|.',
439                 to_each_voice=True,
440             )
441         if segment_number == segment_count and self.final_markup is not None:
442             self.score.add_final_markup(self.final_markup)

```

```

443
444     def get_rehearsal_letter(self):
445         segment_number = self._segment_metadata.get('segment_number', 1)
446         if segment_number == 1:
447             return ''
448         segment_index = segment_number - 1
449         rehearsal_ordinal = ord('A') - 1 + segment_index
450         rehearsal_letter = chr(rehearsal_ordinal)
451         return rehearsal_letter
452
453     def attach_rehearsal_mark(self):
454         markup_a, markup_b = None, None
455         first_leaf = self.score['Time Signature Context'].select_leaves()[0]
456         rehearsal_letter = self.get_rehearsal_letter()
457         if rehearsal_letter:
458             markup_a = markuptools.Markup(rehearsal_letter)
459             markup_a = markup_a.caps().pad_around(0.5).box()
460         if self.name:
461             markup_b = markuptools.Markup("{}".format(self.name or ' '))
462             markup_b = markup_b.fontsize(-3)
463         if markup_a and markup_b:
464             markup = markuptools.Markup.concat([markup_a, ', ', markup_b])
465         else:
466             markup = markup_a or markup_b
467         if markup:
468             rehearsal_mark = indicatortools.RehearsalMark(markup=markup)
469             attach(rehearsal_mark, first_leaf)
470
471     def attach_tempo(self):
472         first_leaf = self.score['Time Signature Context'].select_leaves()[0]
473         if self.tempo is not None:
474             attach(self.tempo, first_leaf)
475
476     def configure_lilypond_file(self):
477         lilypond_file = lilypondfiletools.LilyPondFile()
478         if not self.omit_stylesheets:
479             lilypond_file.use_relative_includes = True
480             path = os.path.join(
481                 '..',
482                 '..',
483                 'stylesheets',
484                 'stylesheet.ily',
485             )
486             lilypond_file.file_initial_user_includes.append(path)
487             if 1 < self._segment_metadata.get('segment_number', 1):
488                 path = os.path.join(
489                     '..',
490                     '..',
491                     'stylesheets',
492                     'nonfirst-segment.ily',
493                 )
494                 lilypond_file.file_initial_user_includes.append(path)
495         lilypond_file.file_initial_system_comments[:] = []
496         score_block = lilypondfiletools.Block(name='score')

```

```

497     score_block.items.append(self.score)
498     lilypond_file.items.append(score_block)
499     lilypond_file.score = self.score
500     self._lilypond_file = lilypond_file
501
502     def configure_score(self):
503         self.add_time_signature_context()
504         self.attach_tempo()
505         self.attach_rehearsal_mark()
506         self.attach_initial_bar_line()
507         self.attach_final_bar_line()
508         self.set_bar_number()
509         self.postprocess_grace_containers()
510         self.postprocess_ties()
511         self.postprocess_staff_lines_spanners()
512         self.postprocess_multimeasure_rests()
513         self.attach_bar_number_comments()
514         self.apply_annotations()
515
516     def apply_annotations(self):
517         import consort
518         if self.annotate_phrasing:
519             consort.annotate(self.score, nonsilence=True)
520         if self.annotate_timespans:
521             context = self.score['Time Signature Context']
522             for leaf in iterate(context).by_class(scoretools.Leaf):
523                 timespan = inspect_(leaf).get_timespan()
524                 start_fraction = markuptools.Markup.fraction(
525                     timespan.start_offset)
526                 stop_fraction = markuptools.Markup.fraction(
527                     timespan.stop_offset)
528                 markup_contents = [start_fraction, ' : ', stop_fraction]
529                 markup = markuptools.Markup.concat(markup_contents)
530                 markup = markuptools.Markup(markup, Up)
531                 markup = markup.pad_around(0.5).box()
532                 attach(markup, leaf)
533         if self.annotate_colors:
534             for voice in iterate(self.score).by_class(scoretools.Voice):
535                 for phrase in voice:
536                     music_specifier = inspect_(phrase).get_indicator(
537                         consort.MusicSpecifier)
538                     if music_specifier is None:
539                         continue
540                     color = music_specifier.color
541                     if color is None:
542                         continue
543                     override(phrase).beam.color = color
544                     override(phrase).dots.color = color
545                     override(phrase).flag.color = color
546                     override(phrase).note_head.color = color
547                     override(phrase).stem.color = color
548
549     def postprocess_multimeasure_rests(self):
550         def division_to_meter(division):

```

```

551     offset = inspect_(division).get_timespan().start_offset
552     timespan = meter_timepans.find_timepans_starting_at(offset)[0]
553     meter = timespan.annotation
554     return meter
555
556 import consort
557 silentSpecifier = consort.MusicSpecifier()
558 meter_timepans = self.meters_to_timepans(self.meters)
559 with systemtools.ForbidUpdate(self.score):
560     for voice in iterate(self.score).by_class(scoretools.Voice):
561         for phrase in voice:
562             musicSpecifier = inspect_(phrase).get_indicator(
563                 consort.MusicSpecifier)
564             if musicSpecifier != silentSpecifier:
565                 continue
566             divisions = [_ for _ in phrase
567                         if isinstance(_, scoretools.MultimeasureRest)]
568             iterator = itertools.groupby(divisions, division_to_meter)
569             for meter, groupedDivisions in iterator:
570                 groupedDivisions = list(groupedDivisions)
571                 count = len(groupedDivisions)
572                 if count == 1:
573                     continue
574                 for division in groupedDivisions[1:]:
575                     phrase.remove(division)
576                 rest = groupedDivisions[0][0]
577                 multiplier = inspect_(rest).get_indicator(
578                     durationtools.Multiplier)
579                 detach(multiplier, rest)
580                 multiplier = multiplier * count
581                 attach(multiplier, rest)
582
583 def postprocessStaffLinesSpanners(self):
584     segmentNumber = self._segment_metadata.get('segment_number', 1)
585     segmentCount = self._segment_metadata.get('segment_count', 1)
586     if segmentNumber != segmentCount:
587         return
588     for voice in iterate(self.score).by_class(scoretools.Voice):
589         for leaf in iterate(voice).by_class(scoretools.Leaf, reverse=True):
590             if not isinstance(leaf, scoretools.MultimeasureRest):
591                 break
592             prototype = spannertools.StaffLinesSpanner
593             if not inspect_(leaf).hasSpanner(prototype):
594                 continue
595             staffLinesSpanner = inspect_(leaf).getSpanner(prototype)
596             components = staffLinesSpanner.components
597             detach(staffLinesSpanner)
598             staffLinesSpanner = new(
599                 staffLinesSpanner,
600                 forbidRestarting=True,
601                 )
602             attach(
603                 staffLinesSpanner,
604                 components,

```

```

605             name='staff_lines_spacer',
606         )
607     break
608
609 def attach_bar_number_comments(self):
610     first_bar_number = self._segment_metadata.get('first_bar_number', 1)
611     measure_offsets = self.measure_offsets
612     for voice in iterate(self.score).by_class(scoretools.Voice):
613         voice_name = voice.name
614         for phrase in voice:
615             for division in phrase:
616                 timespan = inspect_(division).get_timespan()
617                 start_offset = timespan.start_offset
618                 matched = False
619                 for bar_number, measure_offset in enumerate(
620                     measure_offsets, first_bar_number):
621                     if measure_offset == start_offset:
622                         matched = True
623                         break
624                     if not matched:
625                         continue
626                     string = '[{}]\nMeasure {}'.format(
627                         voice_name,
628                         bar_number,
629                     )
630                     comment = indicatortools.LilyPondComment(string)
631                     attach(comment, division)
632
633     def postprocess_ties(self):
634         for component in iterate(self.score).depth_first():
635             if not inspect_(component).has_spacer(spannertools.Tie):
636                 continue
637             tie = inspect_(component).get_spacer(spannertools.Tie)
638             if component != tie[0]:
639                 continue
640             components = tie.components
641             detach(tie)
642             tie = spannertools.Tie(use_messiaen_style_ties=True)
643             attach(tie, components)
644
645     def set_bar_number(self):
646         first_bar_number = self._segment_metadata.get('first_bar_number')
647         if first_bar_number is not None:
648             set_(self.score).current_bar_number = first_bar_number
649         else:
650             #    override(self.score).bar_number.transparent = True
651
652     def copy_voice(
653         self,
654         voice,
655         attachment_names=None,
656         new_voice_name=None,
657         new_context_name=None,
658         remove_grace_containers=False,

```

```

659     remove_ties=False,
660     replace_rests_with_skips=False,
661     ):
662     new_voice = mutate(voice).copy()
663     if new_voice_name:
664         new_voice.name = new_voice_name
665     if new_context_name:
666         new_voice.context_name = new_context_name
667     rests = []
668     for component in iterate(new_voice).depth_first(capped=True):
669         agent = inspect_(component)
670         indicators = agent.get_indicators(unwrap=False)
671         spanners = agent.get_spanners()
672         for x in indicators:
673             if not x.name:
674                 continue
675             if attachment_names and \
676                 not any(x.name.startswith(_) for _ in attachment_names):
677                 x._detach()
678             for x in spanners:
679                 if remove_ties and isinstance(x, spanertools.Tie):
680                     x._detach()
681                 if not x.name:
682                     continue
683                 elif attachment_names and \
684                     not any(x.name.startswith(_) for _ in attachment_names):
685                     x._detach()
686                 if replace_rests_with_skips and \
687                     isinstance(component, scoretools.Rest):
688                     rests.append(component)
689                 grace_containers = agent.get_grace_containers()
690                 if grace_containers and remove_grace_containers:
691                     for grace_container in grace_containers:
692                         grace_container._detach()
693             if replace_rests_with_skips:
694                 for rest in rests:
695                     indicators = inspect_(rest).get_indicators(
696                         durationtools.Multiplier,
697                         )
698                     skip = scoretools.Skip(rest)
699                     if indicators:
700                         attach(indicators[0], skip)
701                         mutate(rest).replace(skip)
702     return new_voice
703
704     @staticmethod
705     def logical_tie_to_music_specifier(logical_tie):
706         import consort
707         parentage = inspect_(logical_tie.head).get_parentage()
708         musicSpecifier = None
709         prototype = consort.MusicSpecifier
710         for parent in parentage:
711             if not inspect_(parent).has_indicator(prototype):
712                 continue

```

```

713     musicSpecifier = inspect_(parent).get_indicator(prototype)
714     return musicSpecifier
715
716     @staticmethod
717     def logical_tie_to_division(logical_tie):
718         import consort
719         parentage = inspect_(logical_tie.head).get_parentage()
720         prototype = consort.MusicSpecifier
721         for i, parent in enumerate(parentage):
722             if inspect_(parent).has_indicator(prototype):
723                 break
724         return parentage[i - 1]
725
726     @staticmethod
727     def logical_tie_to_phrase(logical_tie):
728         import consort
729         parentage = inspect_(logical_tie.head).get_parentage()
730         prototype = consort.MusicSpecifier
731         for parent in parentage:
732             if inspect_(parent).has_indicator(prototype):
733                 return parent
734
735     @staticmethod
736     def logical_tie_to_voice(logical_tie):
737         parentage = inspect_(logical_tie.head).get_parentage()
738         voice = None
739         for parent in parentage:
740             if isinstance(parent, scoretools.Voice):
741                 voice = parent
742                 break
743         return voice
744
745     @staticmethod
746     def logical_tie_to_staff(logical_tie):
747         parentage = inspect_(logical_tie.head).get_parentage()
748         staff = None
749         for parent in parentage:
750             if isinstance(parent, scoretools.Staff):
751                 staff = parent
752                 break
753         return staff
754
755     def postprocess_grace_containers(self):
756         import consort
757         score = self.score
758         stop_trill_span = consort.StopTrillSpan()
759         for leaf in iterate(score).by_class(scoretools.Leaf):
760             agent = inspect_(leaf)
761             spanners = agent.get_spanners(consort.ConsortTrillSpanner)
762             if not spanners:
763                 continue
764             after_graces = agent.get_grace_containers('after')
765             if not after_graces:
766                 continue

```

```

767     after_grace = after_graces[0]
768     leaf = after_grace[0]
769     attach(stop_trill_span, leaf)
770
771     @staticmethod
772     def validate_score(score, verbose=True):
773         import consort
774         manager = systemtools.WellformednessManager(expr=score)
775         triples = manager()
776         for current_violators, current_total, current_check in triples:
777             if verbose:
778                 print('{0} {1} {2} {3}'.format(
779                     current_violators,
780                     current_total,
781                     current_check,
782                     ))
783         if current_violators:
784             raise AssertionError
785         for voice in iterate(score).by_class(scoretools.Voice):
786             #print(voice.name)
787             voice_name = voice.name
788             for phrase in voice:
789                 #print('PHRASE:', phrase)
790                 music_specifier = inspect_(phrase).get_indicator(
791                     consort.MusicSpecifier)
792                 if music_specifier is None:
793                     #print('\tNO MUSIC SPECIFIER')
794                     continue
795                 pitch_handler = music_specifier.pitch_handler
796                 if pitch_handler is not None:
797                     if pitch_handler.pitches_are_nonsemantic:
798                         #print('\tPITCHES ARE NONSEMANTIC')
799                         continue
800                 instrument = music_specifier.instrument
801                 if instrument is None:
802                     instrument = inspect_(phrase).get_effective(
803                         instrumenttools.Instrument)
804                 if instrument is None:
805                     #print('\tNO INSTRUMENT')
806                     continue
807                 pitch_range = instrument.pitch_range
808                 for leaf in iterate(phrase).by_class((
809                     scoretools.Note, scoretools.Chord,
810                     )):
811                     timespan = inspect_(leaf).get_timespan()
812                     #print('\t{!r}'.format(leaf))
813                     if isinstance(leaf, scoretools.Note):
814                         note_head = leaf.note_head
815                         #print('\t\t', note_head)
816                         if note_head.written_pitch not in pitch_range:
817                             override(leaf).note_head.color = 'red'
818                             message = '{0}Out of range: {1} {!r} {!s} {!s}{2}'
819                             message = message.format(
820                               '\x033[91m',

```

```

821             voice_name,
822             timespan,
823             pitch_range,
824             leaf,
825             '\033[0m',
826             )
827             print(message)
828     elif isinstance(leaf, scoretools.Chord):
829         for note_head in leaf.note_heads:
830             #print('\t\t', note_head)
831             if note_head.written_pitch not in pitch_range:
832                 note_head.tweak.color = 'red'
833                 message = '    {}Out of range: {} {!r} {!s} {!s} {!s}{}'
834                 message = message.format(
835                     '\033[91m',
836                     voice_name,
837                     timespan,
838                     pitch_range,
839                     leaf,
840                     note_head,
841                     '\033[0m',
842                     )
843             print(message)
844
845     @staticmethod
846     def can_rewrite_meter(inscribed_timespan):
847         r'''Is true if containers to be inscribed into 'inscribed_timespan' can
848         undergo meter rewriting. Otherwise false.
849
850         Returns boolean.
851         '''
852         import consort
853         musicSpecifier = inscribed_timespan.musicSpecifier
854         if musicSpecifier is None:
855             return True
856         rhythmMaker = musicSpecifier.rhythmMaker
857         if rhythmMaker is None:
858             return True
859         if isinstance(rhythmMaker, consort.CompositeRhythmMaker):
860             specifier = rhythmMaker.default.duration_spellingSpecifier
861         else:
862             specifier = rhythmMaker.duration_spellingSpecifier
863         if specifier is None:
864             return True
865         if specifier.forbid_meter_rewriting:
866             return False
867         return True
868
869     @staticmethod
870     def cleanup_logical_ties(music):
871         for logical_tie in iterate(music).by_logical_tie(
872             nontrivial=True, pitched=True, reverse=True):
873             if len(logical_tie) != 2:
874                 continue

```

```

875     if not logical_tie.all_leaves_are_in_same_parent:
876         continue
877     if logical_tie.written_duration == \
878         durationtools.Duration(1, 8):
879         mutate(logical_tie).replace([scoretools.Note("c'8")])
880     elif logical_tie.written_duration == \
881         durationtools.Duration(1, 16):
882         mutate(logical_tie).replace([scoretools.Note("c'16")])
883
884     @staticmethod
885     def collect_attack_points(score):
886         import consort
887         attack_point_map = collections.OrderedDict()
888         iterator = iterate(score).by_timeline(component_class=scoretools.Note)
889         for note in iterator:
890             logical_tie = inspect_(note).get_logical_tie()
891             if note is not logical_tie.head:
892                 continue
893             attack_point_signature = \
894                 consort.AttackPointSignature.from_logical_tie(logical_tie)
895             attack_point_map[logical_tie] = attack_point_signature
896         return attack_point_map
897
898     @staticmethod
899     def consolidate_demultiplexed_timeespans(demultiplexed_maquette):
900         for voice_name in demultiplexed_maquette:
901             timeespans = demultiplexed_maquette[voice_name]
902             consolidated_timeespans = SegmentMaker.consolidate_timeespans(
903                 timeespans)
904             demultiplexed_maquette[voice_name] = consolidated_timeespans
905
906     @staticmethod
907     def consolidate_rests(music):
908         """Consolidates non-tupletted rests into separate containers in
909         'music'.
910
911         :::
912
913         >>> import consort
914
915         :::
916
917         >>> music = scoretools.Container(r'''
918             ... { r4 c'8 }
919             ... \times 2/3 { d'4 r8 }
920             ... { r4 e'4 f'4 r4 }
921             ... { r4 g8 r8 }
922             ... { r4 }
923             ... { r4 }
924             ... { a'4 \times 2/3 { b'4 r8 } }
925             ... { c''4 r8 }
926             ...
927             >>> print(format(music))
928             {

```

```

929      {
930          r4
931          c'8
932      }
933      \times 2/3 {
934          d'4
935          r8
936      }
937      {
938          r4
939          e'4
940          f'4
941          r4
942      }
943      {
944          r4
945          g8
946          r8
947      }
948      {
949          r4
950      }
951      {
952          r4
953      }
954      {
955          a'4
956          \times 2/3 {
957              b'4
958              r8
959          }
960      }
961      {
962          c''4
963          r8
964      }
965  }
966
967  :::
968
969      >>> music = consort.SegmentMaker.consolidate_rests(music)
970      >>> print(format(music))
971      {
972          {
973              r4
974          }
975          {
976              c'8
977          }
978          \times 2/3 {
979              d'4
980              r8
981          }
982      }

```

```

983         r4
984     }
985     {
986         e'4
987         f'4
988     }
989     {
990         r4
991         r4
992     }
993     {
994         g8
995     }
996     {
997         r8
998         r4
999         r4
1000    }
1001    {
1002        a'4
1003        \times 2/3 {
1004            b'4
1005            r8
1006        }
1007    }
1008    {
1009        c''4
1010    }
1011    {
1012        r8
1013    }
1014}
1015
1016 Returns 'music'.
1017 """
1018 prototype = (
1019     scoretools.Rest,
1020     scoretools.MultimeasureRest,
1021 )
1022 initial_music_duration = inspect_(music).get_duration()
1023 initial_leaves = music.select_leaves()
1024 if not isinstance(music[0], scoretools.Tuplet):
1025     leading_silence = scoretools.Container()
1026     while music[0] and isinstance(music[0][0], prototype):
1027         leading_silence.append(music[0].pop(0))
1028     if leading_silence:
1029         music.insert(0, leading_silence)
1030     if not isinstance(music[-1], scoretools.Tuplet):
1031         tailing_silence = scoretools.Container()
1032         while music[-1] and isinstance(music[-1][-1], prototype):
1033             tailing_silence.insert(0, music[-1].pop())
1034         if tailing_silence:
1035             music.append(tailing_silence)
1036 if len(music) < 2:

```

```

1037         return music
1038     indices = reversed(range(len(music) - 1))
1039     for index in indices:
1040         division = music[index]
1041         next_division = music[index + 1]
1042         silence = scoretools.Container()
1043         if not isinstance(division, scoretools.Tuplet):
1044             while division and isinstance(division[-1], prototype):
1045                 silence.insert(0, division.pop())
1046         if not isinstance(next_division, scoretools.Tuplet):
1047             while next_division and \
1048                 isinstance(next_division[0], prototype):
1049                 silence.append(next_division.pop(0))
1050         if silence:
1051             music.insert(index + 1, silence)
1052         if not division:
1053             music.remove(division)
1054         if not next_division:
1055             music.remove(next_division)
1056     for division in music[:]:
1057         if not division:
1058             music.remove(division)
1059     assert inspect_(music).get_duration() == initial_music_duration
1060     assert music.select_leaves() == initial_leaves
1061     return music
1062
1063     @staticmethod
1064     def consolidate_timepans(timepans, allow_silences=False):
1065         r'''Consolidates contiguous performed timepans by music specifier.
1066
1067         :::
1068
1069         >>> import consort
1070
1071         :::
1072
1073         >>> timepans = timespantools.TimespanInventory([
1074             ...     consort.PerformedTimespan(
1075             ...         start_offset=0,
1076             ...         stop_offset=10,
1077             ...         musicSpecifier='foo',
1078             ...         ),
1079             ...     consort.PerformedTimespan(
1080             ...         start_offset=10,
1081             ...         stop_offset=20,
1082             ...         musicSpecifier='foo',
1083             ...         ),
1084             ...     consort.PerformedTimespan(
1085             ...         start_offset=20,
1086             ...         stop_offset=25,
1087             ...         musicSpecifier='bar',
1088             ...         ),
1089             ...     consort.PerformedTimespan(
1090             ...         start_offset=40,

```

```

1091     ...         stop_offset=50,
1092     ...         musicSpecifier='bar',
1093     ...         ),
1094     ...     consort.PerformedTimespan(
1095     ...         start_offset=50,
1096     ...         stop_offset=58,
1097     ...         musicSpecifier='bar',
1098     ...         ),
1099     ...     ],
1100 >>> print(format(timespans))
1101 timespantools.TimespanInventory(
1102 [
1103     consort.tools.PerformedTimespan(
1104         start_offset=durationtools.Offset(0, 1),
1105         stop_offset=durationtools.Offset(10, 1),
1106         musicSpecifier='foo',
1107         ),
1108     consort.tools.PerformedTimespan(
1109         start_offset=durationtools.Offset(10, 1),
1110         stop_offset=durationtools.Offset(20, 1),
1111         musicSpecifier='foo',
1112         ),
1113     consort.tools.PerformedTimespan(
1114         start_offset=durationtools.Offset(20, 1),
1115         stop_offset=durationtools.Offset(25, 1),
1116         musicSpecifier='bar',
1117         ),
1118     consort.tools.PerformedTimespan(
1119         start_offset=durationtools.Offset(40, 1),
1120         stop_offset=durationtools.Offset(50, 1),
1121         musicSpecifier='bar',
1122         ),
1123     consort.tools.PerformedTimespan(
1124         start_offset=durationtools.Offset(50, 1),
1125         stop_offset=durationtools.Offset(58, 1),
1126         musicSpecifier='bar',
1127         ),
1128     ],
1129 )
1130 :::
1131 :::
1132
1133 >>> timespans = consort.SegmentMaker.consolidate_timespans(
1134     ...     timespans)
1135 >>> print(format(timespans))
1136 timespantools.TimespanInventory(
1137 [
1138     consort.tools.PerformedTimespan(
1139         start_offset=durationtools.Offset(0, 1),
1140         stop_offset=durationtools.Offset(20, 1),
1141         divisions=(
1142             durationtools.Duration(10, 1),
1143             durationtools.Duration(10, 1),
1144             ),

```

```

1145         musicSpecifier='foo',
1146         ),
1147         consort.tools.PerformedTimespan(
1148             startOffset=durationtools.Offset(20, 1),
1149             stopOffset=durationtools.Offset(25, 1),
1150             divisions=(
1151                 durationtools.Duration(5, 1),
1152                 ),
1153                 musicSpecifier='bar',
1154                 ),
1155                 consort.tools.PerformedTimespan(
1156                     startOffset=durationtools.Offset(40, 1),
1157                     stopOffset=durationtools.Offset(58, 1),
1158                     divisions=(
1159                         durationtools.Duration(10, 1),
1160                         durationtools.Duration(8, 1),
1161                         ),
1162                         musicSpecifier='bar',
1163                         ),
1164                     ],
1165                 )
1166             )
1167
1168     Returns new timespan inventory.
1169     """
1170     consolidated_timespans = timespantools.TimespanInventory()
1171     for musicSpecifier, grouped_timespans in \
1172         SegmentMaker.group_timespans(timespans):
1173         if musicSpecifier is None and not allow_silences:
1174             continue
1175         if hasattr(musicSpecifier, 'minimum_phrase_duration'):
1176             duration = musicSpecifier.minimum_phrase_duration
1177             if duration and grouped_timespans.duration < duration:
1178                 continue
1179             divisions = tuple(_.duration for _ in grouped_timespans)
1180             first_timespan = grouped_timespans[0]
1181             last_timespan = grouped_timespans[-1]
1182             consolidated_timespan = new(
1183                 first_timespan,
1184                 divisions=divisions,
1185                 stopOffset=last_timespan.stopOffset,
1186                 originalStopOffset=last_timespan.originalStopOffset,
1187                 )
1188             consolidated_timespans.append(consolidated_timespan)
1189             consolidated_timespans.sort()
1190             return consolidated_timespans
1191
1192     @staticmethod
1193     def debug_timespans(timespans):
1194         import consort
1195         if not timespans:
1196             consort.debug('No timespans found.')
1197         else:
1198             consort.debug('DEBUG: Dumping timespans:')
1199             if isinstance(timespans, dict):

```

```

1199     for voice_name in timespans:
1200         consort.debug('\t' + voice_name)
1201         for timespan in timespans[voice_name]:
1202             divisions = timespan.divisions or []
1203             divisions = ', '.join(str(_) for _ in divisions)
1204             consort.debug('\t\t{}: [{!s} ... {!s}] [{!s}] [{!s}] {}'.format(
1205                 type(timespan).__name__,
1206                 timespan.start_offset,
1207                 timespan.stop_offset,
1208                 timespan.duration,
1209                 divisions,
1210                 timespan.music,
1211                 ))
1212     else:
1213         for timespan in timespans:
1214             consort.debug('\t({}) {}: [{!s} to {!s}]'.format(
1215                 timespan.voice_name,
1216                 type(timespan).__name__,
1217                 timespan.start_offset,
1218                 timespan.stop_offset,
1219                 ))
1220
1221     @staticmethod
1222     def resolve_maquette(multiplexed_timespans):
1223         import consort
1224         demultiplexed_maquette = consort.TimespanInventoryMapping()
1225         for timespan in multiplexed_timespans:
1226             voice_name, layer = timespan.voice_name, timespan.layer
1227             if voice_name not in demultiplexed_maquette:
1228                 demultiplexed_maquette[voice_name] = {}
1229             if layer not in demultiplexed_maquette[voice_name]:
1230                 demultiplexed_maquette[voice_name][layer] = \
1231                     timespantools.TimespanInventory()
1232                 demultiplexed_maquette[voice_name][layer].append(
1233                     timespan)
1234                 demultiplexed_maquette[voice_name][layer]
1235             for voice_name in demultiplexed_maquette:
1236                 for layer, timespans in demultiplexed_maquette[voice_name].items():
1237                     cleaned_layer = SegmentMaker.cleanup_maquette_layer(timespans)
1238                     demultiplexed_maquette[voice_name][layer] = cleaned_layer
1239             for voice_name in demultiplexed_maquette:
1240                 timespan_inventories = demultiplexed_maquette[voice_name]
1241                 timespan_inventory = \
1242                     SegmentMaker.resolve_timespan_inventories(
1243                         timespan_inventories)
1244                 demultiplexed_maquette[voice_name] = timespan_inventory
1245         return demultiplexed_maquette
1246
1247     @staticmethod
1248     def cleanup_maquette_layer(timespans):
1249         import consort
1250         performed_timespans = timespantools.TimespanInventory()
1251         silent_timespans = timespantools.TimespanInventory()
1252         for timespan in timespans:

```

```

1253     if isinstance(timespan, consort.PerformedTimespan):
1254         performed_timespans.append(timespan)
1255     elif isinstance(timespan, consort.SilentTimespan):
1256         silent_timespans.append(timespan)
1257     else:
1258         raise ValueError(timespan)
1259     silent_timespans.compute_logical_or()
1260     for performed_timespan in performed_timespans:
1261         silent_timespans -= performed_timespan
1262     performed_timespans.extend(silent_timespans)
1263     performed_timespans.sort()
1264     return performed_timespans
1265
1266 @staticmethod
1267 def division_is_silent(division):
1268     r'''Is true when division only contains rests, at any depth.
1269
1270     :::
1271
1272     >>> import consort
1273
1274     :::
1275
1276     >>> division = scoretools.Container("c'4 d'4 e'4 f'4")
1277     >>> consort.SegmentMaker.division_is_silent(division)
1278     False
1279
1280     :::
1281
1282     >>> division = scoretools.Container('r4 r8 r16 r32')
1283     >>> consort.SegmentMaker.division_is_silent(division)
1284     True
1285
1286     :::
1287
1288     >>> division = scoretools.Container(
1289     ...     r"c'4 \times 2/3 { d'8 r8 e'8 } f'4")
1290     >>> consort.SegmentMaker.division_is_silent(division)
1291     False
1292
1293     :::
1294
1295     >>> division = scoretools.Container(
1296     ...     r"\times 2/3 { r4 \times 2/3 { r8. } }")
1297     >>> consort.SegmentMaker.division_is_silent(division)
1298     True
1299
1300     Returns boolean.
1301     '''
1302     rest_prototype = (
1303         scoretools.Rest,
1304         scoretools.MultimeasureRest,
1305     )
1306     leaves = division.select_leaves()

```

```

1307     return all(isinstance(leaf, rest_prototype) for leaf in leaves)
1308
1309     def interpret_rhythms(
1310         self,
1311         verbose=True,
1312     ):
1313         multiplexed_timepans = timespanutils.TimespanInventory()
1314
1315         with systemtools.Timer(
1316             enter_message='    populating independent timepans:',
1317             exit_message='        total:',
1318             verbose=verbose,
1319         ):
1320             meters, measure_offsets, multiplexed_timepans = \
1321                 self.populate_independent_timepans(
1322                     self.discard_final_silence,
1323                     multiplexed_timepans,
1324                     self.permitted_time_signatures,
1325                     self.score,
1326                     self.score_template,
1327                     self.settings or (),
1328                     self.desired_duration,
1329                     self.timespan_quantization,
1330                     verbose=verbose,
1331                 )
1332             self._meters = meters
1333
1334         with systemtools.Timer(
1335             enter_message='    populating dependent timepans:',
1336             exit_message='        total:',
1337             verbose=verbose,
1338         ):
1339             demultiplexed_maquette = \
1340                 self.populate_dependent_timepans(
1341                     self.measure_offsets,
1342                     multiplexed_timepans,
1343                     self.score,
1344                     self.score_template,
1345                     self.settings or (),
1346                     self.desired_duration,
1347                     verbose=verbose,
1348                 )
1349
1350         with systemtools.Timer(
1351             '    populated silent timepans:',
1352             verbose=verbose,
1353         ):
1354             demultiplexed_maquette = self.populate_silent_timepans(
1355                 demultiplexed_maquette,
1356                 self.measure_offsets,
1357                 self.voice_names,
1358             )
1359
1360         with systemtools.Timer(

```

```

1361     '    validated timespans:',  

1362     verbose=verbose,  

1363     ):  

1364     self.validate_timespans(demultiplexed_maquette)  

1365  

1366     with systemtools.Timer(  

1367         enter_message='    rewriting meters:',  

1368         exit_message='        total:',  

1369         verbose=verbose,  

1370         ):  

1371         #expr = 'self.rewrite_meters(demultiplexed_maquette, self.meters)'  

1372         #systemtools.IOManager.profile_expr(  

1373             #    expr,  

1374             #    global_context=globals(),  

1375             #    local_context=locals(),  

1376             #    )  

1377         self.rewrite_meters(  

1378             demultiplexed_maquette,  

1379             self.meters,  

1380             self.score,  

1381             verbose=verbose,  

1382             )  

1383  

1384     with systemtools.Timer(  

1385         '    populated score:',  

1386         verbose=verbose,  

1387         ):  

1388         self.populate_score(  

1389             demultiplexed_maquette,  

1390             self.score,  

1391             )  

1392  

1393     self._voicewise_timespans = demultiplexed_maquette  

1394  

1395 def find_meters(  

1396     self,  

1397     permitted_time_signatures=None,  

1398     desired_duration=None,  

1399     timespan_inventory=None,  

1400     ):  

1401     import consort  

1402     offset_counter = metertools.OffsetCounter()  

1403     for timespan in timespan_inventory:  

1404         if isinstance(timespan, consort.SilentTimespan):  

1405             continue  

1406         offset_counter[timespan.start_offset] += 2  

1407         offset_counter[timespan.stop_offset] += 1  

1408     maximum = 1  

1409     if offset_counter:  

1410         maximum = int(max(offset_counter.values()))  

1411     offset_counter[desired_duration] = maximum * 2  

1412     maximum_meter_run_length = self.maximum_meter_run_length  

1413     meters = metertools.Meter.fit_meters_to_expr(  

1414         expr=offset_counter,

```

```

1415         meters=permitted_time_signatures,
1416         maximum_run_length=maximum_meter_run_length,
1417     )
1418     return tuple(meters)
1419
1420     @staticmethod
1421     def get_rhythm_maker(musicSpecifier):
1422         import consort
1423         beamSpecifier = rhythmmakertools.BeamSpecifier(
1424             beam_each_division=False,
1425             beam_divisions_together=False,
1426         )
1427         if musicSpecifier is None:
1428             rhythmMaker = rhythmmakertools.NoteRhythmMaker(
1429                 beamSpecifier=beamSpecifier,
1430                 output_masks=[rhythmmakertools.silence_all()],
1431             )
1432         elif musicSpecifier.rhythm_maker is None:
1433             rhythmMaker = rhythmmakertools.NoteRhythmMaker(
1434                 beamSpecifier=beamSpecifier,
1435                 tieSpecifier=rhythmmakertools.TieSpecifier(
1436                     tie_across_divisions=True,
1437                 ),
1438             )
1439         elif isinstance(musicSpecifier.rhythm_maker,
1440                         consort.CompositeRhythmMaker):
1441             rhythmMaker = musicSpecifier.rhythm_maker.new(
1442                 beamSpecifier=beamSpecifier,
1443             )
1444         else:
1445             rhythmMaker = musicSpecifier.rhythm_maker
1446             beamSpecifier = rhythmMaker.beamSpecifier or beamSpecifier
1447             beamSpecifier = new(
1448                 beamSpecifier,
1449                 beam_each_division=False,
1450                 beam_divisions_together=False,
1451             )
1452             rhythmMaker = new(
1453                 rhythmMaker,
1454                 beamSpecifier=beamSpecifier,
1455             )
1456             assert rhythmMaker is not None
1457             return rhythmMaker
1458
1459     @staticmethod
1460     def group_nonsilent_divisions(music):
1461         r'''Groups non-silent divisions together.
1462
1463         Yields groups in reverse order.
1464
1465         :::
1466
1467         >>> import consort
1468

```

```

1469      ::

1470
1471      >>> divisions = []
1472      >>> divisions.append(scoretools.Container('r4'))
1473      >>> divisions.append(scoretools.Container("c'4"))
1474      >>> divisions.append(scoretools.Container('r4 r4'))
1475      >>> divisions.append(scoretools.Container("d'4 d'4"))
1476      >>> divisions.append(scoretools.Container("e'4 e'4 e'4"))
1477      >>> divisions.append(scoretools.Container('r4 r4 r4'))
1478      >>> divisions.append(scoretools.Container("f'4 f'4 f'4 f'4"))

1479      ::

1480
1481      >>> for group in consort.SegmentMaker.group_nonsilent_divisions(
1482          ...     divisions):
1483          ...     print(group)
1484          (Container("f'4 f'4 f'4 f'4"),)
1485          (Container("d'4 d'4"), Container("e'4 e'4 e'4"))
1486          (Container("c'4"),)

1487
1488      Returns generator.

1489      ''
1490
1491      group = []
1492      for division in tuple(reversed(music)):
1493          if SegmentMaker.division_is_silent(division):
1494              if group:
1495                  yield tuple(reversed(group))
1496                  group = []
1497              else:
1498                  group.append(division)
1499          if group:
1500              yield tuple(reversed(group))

1501
1502      @staticmethod
1503      def group_timeespans(timeespans):
1504          def grouper(timespan):
1505              music_specifier = None
1506              if isinstance(timespan, consort.PerformedTimespan):
1507                  music_specifier = timespan.music_specifier
1508                  if music_specifier is None:
1509                      music_specifier = consort.MusicSpecifier()
1510                  forbid_fusing = timespan.forbid_fusing
1511                  return music_specifier, forbid_fusing
1512
1513          import consort
1514          for partitioned_timeespans in timeespans.partition(
1515              include_tangent_timeespans=True):
1516              for key, grouped_timeespans in itertools.groupby(
1517                  partitioned_timeespans, grouper):
1518                  music_specifier, forbid_fusing = key
1519                  if forbid_fusing:
1520                      for timespan in grouped_timeespans:
1521                          group = timespantools.TimespanInventory([timespan])
1522                          yield music_specifier, group
1523                  else:

```

```

1523         group = timespanools.TimespanInventory(
1524             grouped_timepans)
1525             yield music_specifier, group
1526
1527     @staticmethod
1528     def inscribe_demultiplexed_timepans(
1529         demultiplexed_maquette,
1530         score,
1531     ):
1532         counter = collections.Counter()
1533         voice_names = demultiplexed_maquette.keys()
1534         voice_names = SegmentMaker.sort_voice_names(score, voice_names)
1535         for voice_name in voice_names:
1536             inscribed_timepans = timespanools.TimespanInventory()
1537             uninscribed_timepans = demultiplexed_maquette[voice_name]
1538             for timespan in uninscribed_timepans:
1539                 if timespan.music is None:
1540                     music_specifier = timespan.music_specifier
1541                     if music_specifier not in counter:
1542                         if music_specifier is None:
1543                             seed = 0
1544                         else:
1545                             seed = music_specifier.seed or 0
1546                             counter[music_specifier] = seed
1547                             seed = counter[music_specifier]
1548                             result = SegmentMaker.inscribe_timepans(
1549                                 timespan,
1550                                 seed=seed,
1551                             )
1552                             inscribed_timepans.extend(result)
1553                             # Negative rotation mimics advancing through a series.
1554                             counter[music_specifier] -= 1
1555                         else:
1556                             inscribed_timepans.append(timespan)
1557                             demultiplexed_maquette[voice_name] = inscribed_timepans
1558
1559     @staticmethod
1560     def inscribe_timepans(timespan, seed=None):
1561         r'''Inscribes 'timespan'.
1562
1563         :::
1564
1565         >>> import consort
1566         >>> music_specifier = consort.MusicSpecifier(
1567             ...     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
1568             ...         output_masks=[
1569             ...             rhythmmakertools.SilenceMask(
1570             ...                 indices=[0],
1571             ...                 period=3,
1572             ...                 ),
1573             ...             ],
1574             ...         ),
1575             ...     )
1576

```

```

1577      ::

1578
1579      >>> timespan = consort.PerformedTimespan(
1580          ...      divisions=[durationtools.Duration(1, 4)] * 7,
1581          ...      start_offset=0,
1582          ...      stop_offset=(7, 4),
1583          ...      music_specifier=music_specifier,
1584          ...      )
1585      >>> print(format(timespan))
1586      consort.tools.PerformedTimespan(
1587          start_offset=durationtools.Offset(0, 1),
1588          stop_offset=durationtools.Offset(7, 4),
1589          divisions=
1590              durationtools.Duration(1, 4),
1591              durationtools.Duration(1, 4),
1592              durationtools.Duration(1, 4),
1593              durationtools.Duration(1, 4),
1594              durationtools.Duration(1, 4),
1595              durationtools.Duration(1, 4),
1596              durationtools.Duration(1, 4),
1597              ),
1598          music_specifier=consort.tools.MusicSpecifier(
1599              rhythm_maker=rhythmmakertools.NoteRhythmMaker(
1600                  output_masks=rhythmmakertools.BooleanPatternInventory(
1601                      (
1602                          rhythmmakertools.SilenceMask(
1603                              indices=(0,),
1604                              period=3,
1605                              ),
1606                          ),
1607                      ),
1608                      ),
1609                      ),
1610                  )
1611
1612      ::

1613
1614      >>> result = consort.SegmentMaker.inscribe_timespan(timespan)
1615      >>> print(format(result))
1616      timespantools.TimespanInventory(
1617          [
1618              consort.tools.PerformedTimespan(
1619                  start_offset=durationtools.Offset(1, 4),
1620                  stop_offset=durationtools.Offset(3, 4),
1621                  music=scoretools.Container(
1622                      "{ c'4 } { c'4 }"
1623                      ),
1624                  music_specifier=consort.tools.MusicSpecifier(
1625                      rhythm_maker=rhythmmakertools.NoteRhythmMaker(
1626                          output_masks=rhythmmakertools.BooleanPatternInventory(
1627                              (
1628                                  rhythmmakertools.SilenceMask(
1629                                      indices=(0,),
1630                                      period=3,

```

```

1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663 Returns timespan inventory.
1664 """
1665 inscribed_timespans = timespantools.TimespanInventory()
1666 rhythm_maker = SegmentMaker.get_rhythm_maker(timespan.musicSpecifier)
1667 durations = timespan.divisions[:]
1668 music = SegmentMaker.make_music(
1669     rhythm_maker,
1670     durations,
1671     seed,
1672 )
1673 assert inspect_(music).get_duration() == timespan.duration
1674 for container, duration in zip(music, durations):
1675     assert inspect_(container).get_duration() == duration
1676 music = SegmentMaker.consolidate_rests(music)
1677 assert inspect_(music).get_duration() == timespan.duration
1678 for group in SegmentMaker.group_nonsilent_divisions(music):
1679     start_offset = inspect_(group[0]).get_timespan().start_offset
1680     stop_offset = inspect_(group[-1]).get_timespan().stop_offset
1681     start_offset += timespan.start_offset
1682     stop_offset += timespan.start_offset
1683     container = scoretools.Container()
1684     container.extend(group)

```

```

1685 #         beam = spanertools.GeneralizedBeam(
1686 #             durations=[division._get_duration() for division in music],
1687 #             include_long_duration_notes=False,
1688 #             include_long_duration_rests=False,
1689 #             isolated_nib_direction=None,
1690 #             use_stemlets=False,
1691 #             )
1692 #         attach(beam, container, name='beam')
1693 for division in container:
1694     durations = [division._get_duration()]
1695     beam = spanertools.GeneralizedBeam(
1696         durations=durations,
1697         include_long_duration_notes=False,
1698         include_long_duration_rests=False,
1699         isolated_nib_direction=None,
1700         use_stemlets=True,
1701         )
1702     attach(beam, division)
1703     attach(timespan.musicSpecifier, container, scope=scoretools.Voice)
1704     inscribed_timespan = new(
1705         timespan,
1706         divisions=None,
1707         music=container,
1708         start_offset=start_offset,
1709         stop_offset=stop_offset,
1710         )
1711     assert inspect_(container).get_duration() == \
1712         inscribed_timespan.duration
1713     assert inspect_(container).get_timespan().start_offset == 0
1714     assert inspect_(container[0]).get_timespan().start_offset == 0
1715     inscribed_timespans.append(inscribed_timespan)
1716     inscribed_timespans.sort()
1717 return inscribed_timespans
1718
1719 @staticmethod
1720 def leaf_is_tied(leaf):
1721     prototype = spanertools.Tie
1722     leaf_tie = None
1723     if inspect_(leaf).get_spanners(prototype):
1724         leaf_tie = inspect_(leaf).get_spinner(prototype)
1725     else:
1726         return False
1727     next_leaf = inspect_(leaf).get_leaf(1)
1728     if next_leaf is not None:
1729         if inspect_(next_leaf).get_spanners(prototype):
1730             next_leaf_tie = inspect_(next_leaf).get_spinner(prototype)
1731             if leaf_tie is next_leaf_tie:
1732                 return True
1733     return False
1734
1735 @staticmethod
1736 def make_music(rhythm_maker, durations, seed=0):
1737     music = rhythm_maker(durations, rotation=seed)
1738     for i, division in enumerate(music):

```

```

1739     if (
1740         len(division) == 1 and
1741         isinstance(division[0], scoretools.Tuplet)
1742     ):
1743         music[i] = division[0]
1744     else:
1745         music[i] = scoretools.Container(division)
1746     music = scoretools.Container(music)
1747     prototype = rhythmmakertools.AccelerandoRhythmMaker
1748     if not isinstance(rhythm_maker, prototype):
1749         for division in music[:]:
1750             if (
1751                 isinstance(division, scoretools.Tuplet) and
1752                 division.multiplier == 1
1753             ):
1754                 mutate(division).swap(scoretools.Container())
1755
1756     return music
1757
1758 @staticmethod
1759 def meters_to_offsets(meters):
1760     r'''Converts 'meters' to offsets.
1761
1762     :::
1763
1764     >>> import consort
1765
1766     :::
1767
1768     >>> meters = [
1769         ...     metertools.Meter((3, 4)),
1770         ...     metertools.Meter((2, 4)),
1771         ...     metertools.Meter((6, 8)),
1772         ...     metertools.Meter((5, 16)),
1773         ...
1774     :::
1775
1776     >>> offsets = consort.SegmentMaker.meters_to_offsets(meters)
1777     >>> for x in offsets:
1778         ...
1779         x
1780         ...
1781         Offset(0, 1)
1782         Offset(3, 4)
1783         Offset(5, 4)
1784         Offset(2, 1)
1785         Offset(37, 16)
1786
1787     Returns tuple of offsets.
1788     '''
1789     durations = [_.duration for _ in meters]
1790     offsets = mathtools.cumulative_sums(durations)
1791     offsets = [durationtools.Offset(_) for _ in offsets]
1792     return tuple(offsets)

```

```

1793     @staticmethod
1794     def meters_to_time spans(meters):
1795         '''Convert 'meters' into a collection of annotated time spans.
1796
1797         :::
1798
1799         >>> import consort
1800
1801         :::
1802
1803         >>> meters = [
1804             ...     metertools.Meter((3, 4)),
1805             ...     metertools.Meter((2, 4)),
1806             ...     metertools.Meter((6, 8)),
1807             ...     metertools.Meter((5, 16)),
1808             ...
1809
1810         :::
1811
1812         >>> time spans = consort.SegmentMaker.meters_to_time spans(meters)
1813         >>> print(format(time spans))
1814         consort.tools.TimespanCollection(
1815             [
1816                 timespantools.AnnotatedTimespan(
1817                     start_offset=durationtools.Offset(0, 1),
1818                     stop_offset=durationtools.Offset(3, 4),
1819                     annotation=metertools.Meter(
1820                         '(3/4 (1/4 1/4 1/4))'
1821                         ),
1822                         ),
1823                         timespantools.AnnotatedTimespan(
1824                             start_offset=durationtools.Offset(3, 4),
1825                             stop_offset=durationtools.Offset(5, 4),
1826                             annotation=metertools.Meter(
1827                                 '(2/4 (1/4 1/4))'
1828                                 ),
1829                                 ),
1830                                 timespantools.AnnotatedTimespan(
1831                                     start_offset=durationtools.Offset(5, 4),
1832                                     stop_offset=durationtools.Offset(2, 1),
1833                                     annotation=metertools.Meter(
1834                                         '(6/8 ((3/8 (1/8 1/8 1/8)) (3/8 (1/8 1/8 1/8))))'
1835                                         ),
1836                                         ),
1837                                         timespantools.AnnotatedTimespan(
1838                                             start_offset=durationtools.Offset(2, 1),
1839                                             stop_offset=durationtools.Offset(37, 16),
1840                                             annotation=metertools.Meter(
1841                                                 '(5/16 ((3/16 (1/16 1/16 1/16)) (2/16 (1/16 1/16))))'
1842                                                 ),
1843                                                 ),
1844             ]
1845         )
1846

```

```

1847     Returns timespan collections.
1848     '',
1849     import consort
1850     timespans = consort.TimespanCollection()
1851     offsets = SegmentMaker.meters_to_offsets(meters)
1852     for i, meter in enumerate(meters):
1853         start_offset = offsets[i]
1854         stop_offset = offsets[i + 1]
1855         timespan = timespantools.AnnotatedTimespan(
1856             annotation=meter,
1857             start_offset=start_offset,
1858             stop_offset=stop_offset,
1859             )
1860         timespans.insert(timespan)
1861     return timespans
1862
1863 @staticmethod
1864 def multiplex_timespans(demultiplexed_maquette):
1865     r'''Multiplexes 'demultiplexed_maquette' into a single timespan
1866     inventory.
1867
1868     :::
1869
1870     >>> import consort
1871
1872     :::
1873
1874     >>> demultiplexed = {}
1875     >>> demultiplexed['foo'] = timespantools.TimespanInventory([
1876         ...     timespantools.Timespan(0, 10),
1877         ...     timespantools.Timespan(15, 30),
1878         ... ])
1879     >>> demultiplexed['bar'] = timespantools.TimespanInventory([
1880         ...     timespantools.Timespan(5, 15),
1881         ...     timespantools.Timespan(20, 35),
1882         ... ])
1883     >>> demultiplexed['baz'] = timespantools.TimespanInventory([
1884         ...     timespantools.Timespan(5, 40),
1885         ... ])
1886
1887     :::
1888
1889     >>> multiplexed = consort.SegmentMaker.multiplex_timespans(
1890         ...     demultiplexed)
1891     >>> print(format(multiplexed))
1892     timespantools.TimespanInventory(
1893         [
1894             timespantools.Timespan(
1895                 start_offset=durationtools.Offset(0, 1),
1896                 stop_offset=durationtools.Offset(10, 1),
1897                 ),
1898             timespantools.Timespan(
1899                 start_offset=durationtools.Offset(5, 1),
2000                 stop_offset=durationtools.Offset(15, 1),

```

```

1901             ),
1902             timespantools.Timespan(
1903                 start_offset=durationtools.Offset(5, 1),
1904                 stop_offset=durationtools.Offset(40, 1),
1905             ),
1906             timespantools.Timespan(
1907                 start_offset=durationtools.Offset(15, 1),
1908                 stop_offset=durationtools.Offset(30, 1),
1909             ),
1910             timespantools.Timespan(
1911                 start_offset=durationtools.Offset(20, 1),
1912                 stop_offset=durationtools.Offset(35, 1),
1913             ),
1914         ],
1915     )
1916
1917     Returns timespan inventory.
1918     """
1919     multiplexed_timepans = timespantools.TimespanInventory()
1920     for timepans in demultiplexed_maquette.values():
1921         multiplexed_timepans.extend(timepans)
1922     multiplexed_timepans.sort()
1923     return multiplexed_timepans
1924
1925     def populate_dependent_timepans(
1926         self,
1927         meter_offsets,
1928         multiplexed_timepans,
1929         score,
1930         score_template,
1931         settings,
1932         desired_duration,
1933         verbose=True,
1934     ):
1935         with systemtools.Timer(
1936             '    populated timepans:',
1937             verbose=verbose,
1938         ):
1939             populated = self.populate_multiplexed_maquette(
1940                 dependent=True,
1941                 score=score,
1942                 score_template=score_template,
1943                 settings=settings,
1944                 desired_duration=desired_duration,
1945                 timepans_inventory=multiplexed_timepans,
1946             )
1947         with systemtools.Timer(
1948             '    demultiplexed timepans:',
1949             verbose=verbose,
1950         ):
1951             demultiplexed_maquette = self.resolve_maquette(
1952                 multiplexed_timepans)
1953             self.debug_timepans(demultiplexed_maquette)
1954         with systemtools.Timer(

```

```

1955     '      split timespans:',
1956     verbose=verbose,
1957     ):
1958     self.split_demultiplexed_timespans(
1959         meter_offsets,
1960         demultiplexed_maquette,
1961     )
1962     with systemtools.Timer(
1963         '      pruned short timespans:',
1964         verbose=verbose,
1965     ):
1966         for voice_name, timespans in demultiplexed_maquette.items():
1967             self.prune_short_timespans(timespans)
1968     with systemtools.Timer(
1969         '      pruned malformed timespans:',
1970         verbose=verbose,
1971     ):
1972         for voice_name, timespans in demultiplexed_maquette.items():
1973             self.prune_malformed_timespans(timespans)
1974     with systemtools.Timer(
1975         '      consolidated timespans:',
1976         verbose=verbose,
1977     ):
1978         self.consolidate_demultiplexed_timespans(
1979             demultiplexed_maquette,
1980         )
1981     with systemtools.Timer(
1982         '      inscribed timespans:',
1983         verbose=verbose,
1984     ):
1985         self.inscribe_demultiplexed_timespans(
1986             demultiplexed_maquette,
1987             score,
1988         )
1989     return demultiplexed_maquette
1990
1991 def populate_independent_timespans(
1992     self,
1993     discard_final_silence,
1994     multiplexed_timespans,
1995     permitted_time_signatures,
1996     score,
1997     score_template,
1998     settings,
1999     desired_duration,
2000     timespan_quantization,
2001     verbose=True,
2002     ):
2003     with systemtools.Timer(
2004         '      populated timespans:',
2005         verbose=verbose,
2006     ):
2007         SegmentMaker.populate_multiplexed_maquette(
2008             dependent=False,

```

```

2009     score=score,
2010     score_template=score_template,
2011     settings=settings,
2012     desired_duration=desired_duration,
2013     timespan_inventory=multiplexed_timespans,
2014     timespan_quantization=timespan_quantization,
2015     )
2016 with systemtools.Timer(
2017     '    found meters:',
2018     verbose=verbose,
2019     ):
2020     meters = self.find_meters(
2021         permitted_time_signatures=permitted_time_signatures,
2022         desired_duration=desired_duration,
2023         timespan_inventory=multiplexed_timespans,
2024         )
2025     meter_offsets = SegmentMaker.meters_to_offsets(meters)
2026     with systemtools.Timer(
2027         '    demultiplexed timespans:',
2028         verbose=verbose,
2029         ):
2030         demultiplexed_maquette = SegmentMaker.resolve_maquette(
2031             multiplexed_timespans)
2032         with systemtools.Timer(
2033             '    split timespans:',
2034             verbose=verbose,
2035             ):
2036             SegmentMaker.split_demultiplexed_timespans(
2037                 meter_offsets,
2038                 demultiplexed_maquette,
2039                 )
2040 # TODO: Determine best place for malformed timespan pruning.
2041 with systemtools.Timer(
2042     '    pruned short timespans:',
2043     verbose=verbose,
2044     ):
2045     SegmentMaker.prune_short_timespans(multiplexed_timespans)
2046     with systemtools.Timer(
2047         '    pruned malformed timespans:',
2048         verbose=verbose,
2049         ):
2050         for voice_name, timespans in demultiplexed_maquette.items():
2051             SegmentMaker.prune_malformed_timespans(timespans)
2052         with systemtools.Timer(
2053             '    consolidated timespans:',
2054             verbose=verbose,
2055             ):
2056             SegmentMaker.consolidate_demultiplexed_timespans(
2057                 demultiplexed_maquette,
2058                 )
2059         with systemtools.Timer(
2060             '    inscribed timespans:',
2061             verbose=verbose,
2062             ):

```

```

2063     SegmentMaker.inscribe_demultiplexed_timespans(
2064         demultiplexed_maquette,
2065         score,
2066         )
2067     with systemtools.Timer(
2068         '    multiplexed timespans:',
2069         verbose=verbose,
2070         ):
2071         multiplexed_timespans = SegmentMaker.multiplex_timespans(
2072             demultiplexed_maquette)
2073     # TODO: Why prune after consolidation?
2074     with systemtools.Timer(
2075         '    pruned meters:',
2076         verbose=verbose,
2077         ):
2078         meters = SegmentMaker.prune_meters(
2079             discard_final_silence,
2080             meters,
2081             multiplexed_timespans.stop_offset,
2082             )
2083         meter_offsets = SegmentMaker.meters_to_offsets(meters)
2084     return meters, meter_offsets, multiplexed_timespans
2085
2086     @staticmethod
2087     def populate_multiplexed_maquette(
2088         dependent=False,
2089         score=None,
2090         score_template=None,
2091         settings=None,
2092         desired_duration=None,
2093         timespan_inventory=None,
2094         timespan_quantization=None,
2095         ):
2096         segment_timespan = timespantools.Timespan(0, desired_duration)
2097         if timespan_quantization is None:
2098             timespan_quantization = durationtools.Duration(1, 16)
2099         if timespan_inventory is None:
2100             timespan_inventory = timespantools.TimespanInventory()
2101         independent_settings = [setting for setting in settings
2102             if not setting.timespan_maker.is_dependent
2103             ]
2104         dependent_settings = [setting for setting in settings
2105             if setting.timespan_maker.is_dependent
2106             ]
2107         if dependent:
2108             settings = dependent_settings
2109             start_index = len(independent_settings)
2110         else:
2111             settings = independent_settings
2112             start_index = 0
2113         if not settings:
2114             return False
2115         for layer, music_setting in enumerate(settings, start_index):
2116             music_setting(

```

```

2117     layer=layer,
2118     score=score,
2119     score_template=score_template,
2120     segment_timespan=segment_timespan,
2121     timespan_inventory=timespan_inventory,
2122     timespan_quantization=timespan_quantization,
2123     )
2124 SegmentMaker.debug_timespans(timespan_inventory)
2125     return True
2126
2127 @staticmethod
2128 def populate_score(
2129     demultiplexed_maquette,
2130     score,
2131     ):
2132     for voice_name, timespans in demultiplexed_maquette.items():
2133         voice = score[voice_name]
2134         for timespan in timespans:
2135             assert timespan.duration == \
2136                 inspect_(timespan.music).get_duration()
2137             voice.append(timespan.music)
2138     return score
2139
2140 @staticmethod
2141 def populate_silent_timespans(
2142     demultiplexed_maquette,
2143     meter_offsets,
2144     voice_names=None,
2145     ):
2146     import consort
2147     silent_musicSpecifier = consort.MusicSpecifier()
2148     rhythm_maker = SegmentMaker.get_rhythm_maker(None)
2149     if voice_names is None:
2150         voice_names = demultiplexed_maquette.keys()
2151     else:
2152         voice_names = set(voice_names)
2153         voice_names.update(demultiplexed_maquette.keys())
2154     for voice_name in voice_names:
2155         if voice_name not in demultiplexed_maquette:
2156             demultiplexed_maquette[voice_name] = \
2157                 timespantools.TimespanInventory()
2158             timespans = demultiplexed_maquette[voice_name]
2159             silences = timespantools.TimespanInventory([
2160                 consort.SilentTimespan(
2161                     start_offset=0,
2162                     stop_offset=meter_offsets[-1],
2163                     voice_name=voice_name,
2164                     )
2165                 ])
2166             silences = SegmentMaker.subtract_timespan_inventories(
2167                 silences, timespans)
2168             silences = SegmentMaker.split_timespans(meter_offsets, silences)
2169             for group in silences.partition(include_tangent_timespans=True):
2170                 start_offset = group.start_offset,

```

```

2171     stop_offset = group.stop_offset,
2172     durations = [_.duration for _ in group]
2173     silence = SegmentMaker.make_music(
2174         rhythm_maker,
2175         durations,
2176         )
2177     attach(silent_musicSpecifier, silence, scope=scoretools.Voice)
2178     silent_timespan = consort.PerformedTimespan(
2179         music=silence,
2180         start_offset=start_offset,
2181         stop_offset=stop_offset,
2182         voice_name=voice_name,
2183         )
2184     timespans.append(silent_timespan)
2185     timespans.sort()
2186     return demultiplexed_maquette
2187
2188 @staticmethod
2189 def prune_meters():
2190     discard_final_silence,
2191     meters,
2192     stop_offset,
2193     ):
2194     discard_final_silence = bool(discard_final_silence)
2195     if discard_final_silence and stop_offset:
2196         meters = list(meters)
2197         total_meter_durations = sum(_.duration for _ in meters[:-1])
2198         while stop_offset <= total_meter_durations:
2199             meters.pop()
2200             total_meter_durations = sum(_.duration for _ in meters[:-1])
2201     return tuple(meters)
2202
2203 @staticmethod
2204 def prune_short_timespans(timespans):
2205     for timespan in timespans[:]:
2206         if timespan.minimum_duration and \
2207             timespan.duration < timespan.minimum_duration and \
2208             timespan.music is None:
2209             timespans.remove(timespan)
2210
2211 @staticmethod
2212 def prune_malformed_timespans(timespans):
2213     for timespan in timespans[:]:
2214         if not timespan.is_well_formed:
2215             assert timespan.music is None
2216             timespans.remove(timespan)
2217
2218 @staticmethod
2219 def report(timespan_inventory):
2220     print('REPORTING')
2221     for timespan in timespan_inventory:
2222         print(
2223             '\t',
2224             '{}:'.format(timespan.voice_name),

```

```

2225     '[{}]' .format(timespan.layer),
2226     type(timespan).__name__,
2227     float(timespan.start_offset),
2228     float(timespan.stop_offset),
2229     )
2230   print()
2231
2232 @staticmethod
2233 def resolve_timespan_inventories(
2234     timespan_inventories=None,
2235     ):
2236   import consort
2237   timespan_inventories = [x[1] for x in
2238     sorted(timespan_inventories.items(),
2239       key=lambda item: item[0],
2240       )
2241     ]
2242   for timespan_inventory in timespan_inventories:
2243     assert timespan_inventory.all_are_nonoverlapping
2244   resolved_inventory = consort.TimespanCollection()
2245   for timespan in timespan_inventories[0]:
2246     if isinstance(timespan, consort.SilentTimespan):
2247       continue
2248     resolved_inventory.insert(timespan)
2249   for timespan_inventory in timespan_inventories[1:]:
2250     resolved_inventory = SegmentMaker.subtract_timespan_inventories(
2251       resolved_inventory,
2252       timespan_inventory,
2253       )
2254   for timespan in resolved_inventory[:]:
2255     if timespan.minimum_duration and \
2256       timespan.duration < timespan.minimum_duration:
2257       resolved_inventory.remove(timespan)
2258   for timespan in timespan_inventory:
2259     if isinstance(timespan, consort.SilentTimespan):
2260       continue
2261     resolved_inventory.append(timespan)
2262   resolved_inventory.sort()
2263   resolved_inventory = timespantools.TimespanInventory(
2264     resolved_inventory[:],
2265     )
2266   return resolved_inventory
2267
2268 @staticmethod
2269 def rewrite_container_meter(
2270     container,
2271     meter_timespans,
2272     forbid_staff_lines_spinner=None,
2273     ):
2274   assert meter_timespans
2275   assert meter_timespans[0].start_offset <=
2276     inspect_(container).get_timespan().start_offset
2277   last_leaf = container.select_leaves()[-1]
2278   is_tied = SegmentMaker.leaf_is_tied(last_leaf)

```

```

2279     container_timespan = inspect_(container).get_timespan()
2280     if isinstance(container, scoretools.Tuplet):
2281         contents_duration = container._contents_duration
2282         meter = metertools.Meter(contents_duration)
2283         boundary_depth = 1
2284         if meter.numerator in (3, 4):
2285             boundary_depth = None
2286         mutate(container[:]).rewrite_meter(
2287             meter,
2288             boundary_depth=boundary_depth,
2289             maximum_dot_count=2,
2290             )
2291     elif len(meter_timespans) == 1:
2292         container_timespan = inspect_(container).get_timespan()
2293         container_start_offset = container_timespan.start_offset
2294         container_stop_offset = container_timespan.stop_offset
2295         meter_timespan = meter_timespans[0]
2296         relative_meter_start_offset = meter_timespan.start_offset
2297         assert relative_meter_start_offset <= container_start_offset
2298         absolute_meter_stop_offset = (
2299             relative_meter_start_offset +
2300             container_start_offset +
2301             meter_timespan.duration
2302             )
2303         assert container_stop_offset <= absolute_meter_stop_offset
2304         if meter_timespan.is_congruent_to_timespan(container_timespan) \
2305             and SegmentMaker.division_is_silent(container):
2306             multimeasure_rest = scoretools.MultimeasureRest(1)
2307             duration = inspect_(container).get_duration()
2308             multiplier = durationtools.Multiplier(duration)
2309             attach(multiplier, multimeasure_rest)
2310             container[:] = [multimeasure_rest]
2311             if not forbid_staff_lines_spinner:
2312                 previous_leaf = multimeasure_rest._get_leaf(-1)
2313                 if isinstance(previous_leaf, scoretools.MultimeasureRest):
2314                     staff_lines_spinner = \
2315                         inspect_(previous_leaf).get_spinner(
2316                             spannertools.StaffLinesSpinner)
2317                     components = staff_lines_spinner.components
2318                     components = components + [multimeasure_rest]
2319                     detach(staff_lines_spinner)
2320                 else:
2321                     staff_lines_spinner = spannertools.StaffLinesSpinner([0])
2322                     components = [multimeasure_rest]
2323                     attach(
2324                         staff_lines_spinner,
2325                         components,
2326                         name='staff_lines_spinner',
2327                         )
2328             else:
2329                 meter = meter_timespan.annotation
2330                 meter_offset = meter_timespan.start_offset
2331                 initial_offset = container_start_offset - meter_offset
2332                 boundary_depth = 1

```

```

2333     if meter.numerator in (3, 4):
2334         boundary_depth = None
2335         mutate(container[:]).rewrite_meter(
2336             meter,
2337             boundary_depth=boundary_depth,
2338             initial_offset=initial_offset,
2339             maximum_dot_count=2,
2340             )
2341     else:
2342         # TODO: handle bar-line-crossing containers
2343         raise AssertionError('Bar-line-crossing containers not permitted.')
2344     if is_tied:
2345         last_leaf = container.select_leaves()[-1]
2346         next_leaf = inspect_(last_leaf).get_leaf(1)
2347         selection = selectiontools.ContiguousSelection((
2348             last_leaf, next_leaf))
2349         selection._attach_tie_spacer_to_leaf_pair()
2350
2351     @staticmethod
2352     def rewrite_meters(
2353         demultiplexed_maquette,
2354         meters,
2355         score,
2356         verbose=True,
2357         ):
2358         import consort
2359         meter_timepoints = SegmentMaker.meters_to_timepoints(meters)
2360         cache = {}
2361         for context_name in sorted(demultiplexed_maquette):
2362             inscribed_timepoints = demultiplexed_maquette[context_name]
2363             consort.debug('CONTEXT: {}'.format(context_name))
2364             context = score[context_name]
2365             forbid_staff_lines_spacer = context.context_name == 'Dynamics'
2366             progress_indicator = systemtools.ProgressIndicator(
2367                 message='      rewriting {}'.format(context_name),
2368                 verbose=verbose,
2369                 )
2370             with progress_indicator:
2371                 for inscribed_timepoint in inscribed_timepoints:
2372                     consort.debug('\t{} {} {}'.format(
2373                         inscribed_timepoint.start_offset,
2374                         inscribed_timepoint.stop_offset,
2375                         inscribed_timepoint.music,
2376                         ))
2377                     if not SegmentMaker.can_rewrite_meter(inscribed_timepoint):
2378                         continue
2379                     with systemtools.ForbidUpdate(
2380                         inscribed_timepoint.music,
2381                         update_on_exit=True,
2382                         ):
2383                         for i, container in enumerate(inscribed_timepoint.music):
2384                             container_timepoint = inspect_(container).get_timepoint()
2385                             container_timepoint = container_timepoint.translate(
2386                                 inscribed_timepoint.start_offset)

```

```

2387     if i == 0:
2388         assert container_timespan.start_offset == \
2389             inscribed_timespan.start_offset
2390     if i == (len(inscribed_timespan.music) - 1):
2391         assert container_timespan.stop_offset == \
2392             inscribed_timespan.stop_offset
2393     if container_timespan in cache:
2394         intersecting_meters = cache[container_timespan]
2395     else:
2396         intersecting_meters = \
2397             meter_timepans.find_timepans_intersecting_timespan(
2398                 container_timespan)
2399         cache[container_timespan] = intersecting_meters
2400     shifted_intersecting_meters = [
2401         _.translate(-1 * inscribed_timespan.start_offset)
2402         for _ in intersecting_meters
2403     ]
2404     consort.debug('\t\t{!r} {!r}'.format(
2405         container,
2406         container_timespan,
2407     ))
2408     for intersecting_meter in intersecting_meters:
2409         consort.debug('\t\t\t' + repr(intersecting_meter))
2410     SegmentMaker.rewrite_container_meter(
2411         container,
2412         shifted_intersecting_meters,
2413         forbid_staff_lines_spacer,
2414     )
2415     SegmentMaker.cleanup_logical_ties(container)
2416     progress_indicator.advance()
2417
2418     @staticmethod
2419     def sort_voice_names(score, voice_names):
2420         result = []
2421         for voice in iterate(score).by_class(scoretools.Voice):
2422             if voice.name in voice_names:
2423                 result.append(voice.name)
2424         return tuple(result)
2425
2426     @staticmethod
2427     def split_demultiplexed_timepans(
2428         meter_offsets=None,
2429         demultiplexed_maquette=None,
2430     ):
2431         for voice_name in demultiplexed_maquette:
2432             timespan_inventory = demultiplexed_maquette[voice_name]
2433             split_inventory = SegmentMaker.split_timepans(
2434                 meter_offsets,
2435                 timespan_inventory,
2436             )
2437             demultiplexed_maquette[voice_name] = split_inventory
2438
2439     @staticmethod
2440     def split_timepans(offsets, timespan_inventory):

```

```

2441     offsets = list(offsets)
2442     timespan_inventory.sort()
2443     split_inventory = timespantools.TimespanInventory()
2444     for timespan in timespan_inventory:
2445         current_offsets = []
2446         while offsets and offsets[0] <= timespan.start_offset:
2447             offsets.pop(0)
2448         while offsets and offsets[0] < timespan.stop_offset:
2449             current_offsets.append(offsets.pop(0))
2450         if hasattr(timespan, 'music') and timespan.music:
2451             # We don't need to split already-inscribed timespans
2452             split_inventory.append(timespan)
2453             continue
2454         elif timespan.forbid_splitting:
2455             continue
2456         if current_offsets:
2457             shards = timespan.split_at_offsets(current_offsets)
2458             for shard in shards:
2459                 if shard.minimum_duration:
2460                     if shard.minimum_duration <= shard.duration:
2461                         split_inventory.append(shard)
2462                     else:
2463                         split_inventory.append(shard)
2464                 else:
2465                     if timespan.minimum_duration:
2466                         if timespan.minimum_duration <= timespan.duration:
2467                             split_inventory.append(timespan)
2468                     else:
2469                         split_inventory.append(timespan)
2470             split_inventory.sort()
2471     return split_inventory
2472
2473 @staticmethod
2474 def subtract_timespan_inventories(inventory_one, inventory_two):
2475     r'''Subtracts 'inventory_two' from 'inventory_one'.
2476
2477     ::

2478
2479     >>> inventory_one = timespantools.TimespanInventory([
2480         ...     timespantools.Timespan(0, 10),
2481         ...     timespantools.Timespan(10, 20),
2482         ...     timespantools.Timespan(40, 80),
2483         ... ])
2484
2485     ::

2486
2487     >>> inventory_two = timespantools.TimespanInventory([
2488         ...     timespantools.Timespan(5, 15),
2489         ...     timespantools.Timespan(25, 35),
2490         ...     timespantools.Timespan(35, 45),
2491         ...     timespantools.Timespan(55, 65),
2492         ...     timespantools.Timespan(85, 95),
2493         ... ])
2494

```

```

2495     :::
2496
2497     >>> import consort
2498     >>> manager = consort.SegmentMaker
2499     >>> result = manager.subtract_timespan_inventories(
2500         ...     inventory_one,
2501         ...     inventory_two,
2502         ...     )
2503     >>> print(format(result))
2504     timespantools.TimespanInventory(
2505         [
2506             timespantools.Timespan(
2507                 start_offset=durationtools.Offset(0, 1),
2508                 stop_offset=durationtools.Offset(5, 1),
2509                 ),
2510             timespantools.Timespan(
2511                 start_offset=durationtools.Offset(15, 1),
2512                 stop_offset=durationtools.Offset(20, 1),
2513                 ),
2514             timespantools.Timespan(
2515                 start_offset=durationtools.Offset(45, 1),
2516                 stop_offset=durationtools.Offset(55, 1),
2517                 ),
2518             timespantools.Timespan(
2519                 start_offset=durationtools.Offset(65, 1),
2520                 stop_offset=durationtools.Offset(80, 1),
2521                 ),
2522         ]
2523     )
2524
2525     :::
2526
2527     >>> result = manager.subtract_timespan_inventories(
2528         ...     inventory_two,
2529         ...     inventory_one,
2530         ...     )
2531     >>> print(format(result))
2532     timespantools.TimespanInventory(
2533         [
2534             timespantools.Timespan(
2535                 start_offset=durationtools.Offset(25, 1),
2536                 stop_offset=durationtools.Offset(35, 1),
2537                 ),
2538             timespantools.Timespan(
2539                 start_offset=durationtools.Offset(35, 1),
2540                 stop_offset=durationtools.Offset(40, 1),
2541                 ),
2542             timespantools.Timespan(
2543                 start_offset=durationtools.Offset(85, 1),
2544                 stop_offset=durationtools.Offset(95, 1),
2545                 ),
2546         ]
2547     )
2548

```

```

2549 """
2550 import consort
2551 resulting_timepans = consort.TimespanCollection()
2552 if not inventory_two:
2553     return timespantools.TimespanInventory(inventory_one)
2554 elif not inventory_one:
2555     return timespantools.TimespanInventory()
2556 subtractee_index = 0
2557 subtractor_index = 0
2558 subtractee = None
2559 subtractor = None
2560 subtractee_is_modified = False
2561 while subtractee_index < len(inventory_one) and \
2562     subtractor_index < len(inventory_two):
2563     if subtractee is None:
2564         subtractee = inventory_one[subtractee_index]
2565         subtractee_is_modified = False
2566     if subtractor is None:
2567         subtractor = inventory_two[subtractor_index]
2568     if subtractee.intersects_timespan(subtractor):
2569         subtraction = subtractee - subtractor
2570         if len(subtraction) == 1:
2571             subtractee = subtraction[0]
2572             subtractee_is_modified = True
2573         elif len(subtraction) == 2:
2574             resulting_timepans.insert(subtraction[0])
2575             subtractee = subtraction[1]
2576             subtractee_is_modified = True
2577         else:
2578             subtractee = None
2579             subtractee_index += 1
2580     else:
2581         if subtractee.stops_before_or_at_offset(
2582             subtractor.start_offset):
2583             resulting_timepans.insert(subtractee)
2584             subtractee = None
2585             subtractee_index += 1
2586         else:
2587             subtractor = None
2588             subtractor_index += 1
2589     if subtractee_is_modified:
2590         if subtractee:
2591             resulting_timepans.insert(subtractee)
2592             resulting_timepans.insert(inventory_one[subtractee_index + 1:])
2593     else:
2594         resulting_timepans.insert(inventory_one[subtractee_index:])
2595 resulting_timepans = timespantools.TimespanInventory(
2596     resulting_timepans[:])
2597 return resulting_timepans
2598
2599 @staticmethod
2600 def validate_timepans(demultiplexed_maquette):
2601     durations = set()
2602     for voice_name, timepans in demultiplexed_maquette.items():

```

```

2603     timespans.sort()
2604     assert timespans.start_offset == 0
2605     assert timespans.all_are_contiguous
2606     assert timespans.all_are_well_formed
2607     assert timespans.all_are_nonoverlapping
2608     durations.add(timespans.stop_offset)
2609     assert len(tuple(durations)) == 1
2610
2611     def update_segment_metadata(self):
2612         self._segment_metadata.update(
2613             end_instruments_by_staff=self.get_end_instruments(),
2614             end_tempo=self.get_end_tempo_indication(),
2615             end_time_signature=self.get_end_time_signature(),
2616             is_repeated=self.repeat,
2617             measure_count=len(self.meters),
2618         )
2619
2620     ### PUBLIC PROPERTIES ###
2621
2622     @property
2623     def attack_point_map(self):
2624         return self._attack_point_map
2625
2626     @property
2627     def meters(self):
2628         return self._meters
2629
2630     @property
2631     def score(self):
2632         return self._score
2633
2634     @property
2635     def voicewise_timespans(self):
2636         return self._voicewise_timespans
2637
2638     @property
2639     def desired_duration(self):
2640         tempo = self.tempo
2641         if tempo is None:
2642             tempo = indicatortools.Tempo((1, 4), 60)
2643             tempo_desired_duration_in_seconds = durationtools.Duration(
2644                 tempo.duration_to_milliseconds(tempo.duration),
2645                 1000,
2646             )
2647             desired_duration = durationtools.Duration(
2648                 self.desired_duration_in_seconds /
2649                 tempo_desired_duration_in_seconds
2650             ).limit_denominator(8))
2651             desired_duration *= tempo.duration
2652             count = desired_duration // durationtools.Duration(1, 8)
2653             desired_duration = durationtools.Duration(count, 8)
2654             assert 0 < desired_duration
2655             return desired_duration
2656

```

```

2657     @property
2658     def desired_duration_in_seconds(self):
2659         return self._desired_duration_in_seconds
2660
2661     @desired_duration_in_seconds.setter
2662     def desired_duration_in_seconds(self, desired_duration_in_seconds):
2663         if desired_duration_in_seconds is not None:
2664             desired_duration_in_seconds = durationtools.Duration(
2665                 desired_duration_in_seconds,
2666             )
2667             self._desired_duration_in_seconds = desired_duration_in_seconds
2668
2669     @property
2670     def discard_final_silence(self):
2671         return self._discard_final_silence
2672
2673     @discard_final_silence.setter
2674     def discard_final_silence(self, discard_final_silence):
2675         if discard_final_silence is not None:
2676             discard_final_silence = bool(discard_final_silence)
2677             self._discard_final_silence = discard_final_silence
2678
2679     @property
2680     def final_markup(self):
2681         return None
2682
2683     @property
2684     def annotate_colors(self):
2685         return self._annotate_colors
2686
2687     @annotate_colors.setter
2688     def annotate_colors(self, expr):
2689         if expr is not None:
2690             expr = bool(expr)
2691             self._annotate_colors = expr
2692
2693     @property
2694     def annotate_phrasing(self):
2695         return self._annotate_phrasing
2696
2697     @annotate_phrasing.setter
2698     def annotate_phrasing(self, expr):
2699         if expr is not None:
2700             expr = bool(expr)
2701             self._annotate_phrasing = expr
2702
2703     @property
2704     def annotate_timepans(self):
2705         return self._annotate_timepans
2706
2707     @annotate_timepans.setter
2708     def annotate_timepans(self, expr):
2709         if expr is not None:
2710             expr = bool(expr)

```

```

2711     self._annotate_time_spans = expr
2712
2713     @property
2714     def lilypond_file(self):
2715         return self._lilypond_file
2716
2717     @property
2718     def maximum_meter_run_length(self):
2719         return self._maximum_meter_run_length
2720
2721     @maximum_meter_run_length.setter
2722     def maximum_meter_run_length(self, expr):
2723         if expr is not None:
2724             expr = int(expr)
2725             assert 0 < expr
2726             self._maximum_meter_run_length = expr
2727
2728     @property
2729     def measure_offsets(self):
2730         measure_durations = [x.duration for x in self.time_signatures]
2731         measure_offsets = mathtools.cumulative_sums(measure_durations)
2732         return measure_offsets
2733
2734     @property
2735     def name(self):
2736         return self._name
2737
2738     @name.setter
2739     def name(self, expr):
2740         if expr is not None:
2741             expr = str(expr)
2742             self._name = expr
2743
2744     @property
2745     def omit_stylesheets(self):
2746         return self._omit_stylesheets
2747
2748     @omit_stylesheets.setter
2749     def omit_stylesheets(self, omit_stylesheets):
2750         if omit_stylesheets is not None:
2751             omit_stylesheets = bool(omit_stylesheets)
2752             self._omit_stylesheets = omit_stylesheets
2753
2754     @property
2755     def permitted_time_signatures(self):
2756         r'''Gets and sets segment maker's permitted time signatures.
2757
2758         :::
2759
2760         >>> import consort
2761         >>> segment_maker = consort.SegmentMaker()
2762         >>> time_signatures = [(3, 4), (2, 4), (5, 8)]
2763         >>> segment_maker.permitted_time_signatures = time_signatures
2764         >>> print(format(segment_maker))

```

```

2765     consort.tools.SegmentMaker(
2766         permitted_time_signatures=indicatortools.TimeSignatureInventory(
2767             [
2768                 indicatortools.TimeSignature((3, 4)),
2769                 indicatortools.TimeSignature((2, 4)),
2770                 indicatortools.TimeSignature((5, 8)),
2771             ]
2772         ),
2773     )
2774
2775     '',
2776     return self._permitted_time_signatures
2777
2778     @permitted_time_signatures.setter
2779     def permitted_time_signatures(self, permitted_time_signatures):
2780         if permitted_time_signatures is not None:
2781             permitted_time_signatures = indicatortools.TimeSignatureInventory(
2782                 items=permitted_time_signatures,
2783             )
2784             self._permitted_time_signatures = permitted_time_signatures
2785
2786     @property
2787     def score_package_metadata(self):
2788         module_name = '{}.__metadata__'.format(self.score_package_name)
2789         try:
2790             module = importlib.import_module(module_name)
2791             metadata = getattr(module, 'metadata')
2792         except ImportError:
2793             metadata = {}
2794         return metadata
2795
2796     @property
2797     def score_package_module(self):
2798         module = importlib.import_module(self.score_package_name)
2799         return module
2800
2801     @property
2802     def score_package_name(self):
2803         return 'consort'
2804
2805     @property
2806     def score_package_path(self):
2807         return self.score_package_module.__path__[0]
2808
2809     @property
2810     def score_template(self):
2811         r'''Gets and sets segment maker's score template.
2812
2813         :::
2814
2815         >>> import consort
2816         >>> segment_maker = consort.SegmentMaker()
2817         >>> score_template = templatetools.StringOrchestraScoreTemplate(
2818             ...
2819             violin_count=2,
```

```

2819     ...      viola_count=1,
2820     ...      cello_count=1,
2821     ...      contrabass_count=0,
2822     ...      )
2823     >>> segment_maker.score_template = score_template
2824     >>> print(format(segment_maker))
2825     consort.tools.SegmentMaker(
2826         score_template=templatetools.StringOrchestraScoreTemplate(
2827             violin_count=2,
2828             viola_count=1,
2829             cello_count=1,
2830             contrabass_count=0,
2831             split_hands=True,
2832             use_percussion_clefs=False,
2833             ),
2834         )
2835     '',
2836
2837     return self._score_template
2838
2839 @score_template.setter
2840 def score_template(self, score_template):
2841     self._score_template = score_template
2842
2843 @property
2844 def segment_duration(self):
2845     return sum(x.duration for x in self.time_signatures)
2846
2847 @property
2848 def settings(self):
2849     return tuple(self._settings)
2850
2851 @settings.setter
2852 def settings(self, settings):
2853     import consort
2854     if settings is not None:
2855         if not isinstance(settings, collections.Sequence):
2856             settings = (settings,)
2857         assert all(isinstance(_, consort.MusicSetting) for _ in settings)
2858         settings = list(settings)
2859     self._settings = settings or []
2860
2861 @property
2862 def tempo(self):
2863     r'''Gets and sets segment maker tempo.
2864
2865     :::
2866
2867     >>> import consort
2868     >>> segment_maker = consort.SegmentMaker()
2869     >>> tempo = indicatortools.Tempo((1, 4), 52)
2870     >>> segment_maker.tempo = tempo
2871     >>> print(format(segment_maker))
2872     consort.tools.SegmentMaker(

```

```

2873         tempo=indicatortools.Tempo(
2874             duration=durationtools.Duration(1, 4),
2875             units_per_minute=52,
2876             ),
2877         )
2878     """
2879     tempo = self._tempo
2880     if tempo is not None:
2881         return tempo
2882     elif self._previous_segment_metadata is not None:
2883         tempo = self._previous_segment_metadata.get(
2884             'end_tempo')
2885         if tempo:
2886             tempo = indicatortools.Tempo(*tempo)
2887     return tempo
2888
2889 @tempo.setter
2890 def tempo(self, tempo):
2891     if tempo is not None:
2892         if not isinstance(tempo, indicatortools.Tempo):
2893             tempo = indicatortools.Tempo(tempo)
2894     self._tempo = tempo
2895
2896
2897 @property
2898 def time_signatures(self):
2899     return tuple(
2900         meter.implied_time_signature
2901         for meter in self.meters
2902     )
2903
2904
2905 @property
2906 def timespan_quantization(self):
2907     """Gets and sets segment maker timespan quantization.
2908     :::
2909
2910     >>> import consort
2911     >>> segment_maker = consort.SegmentMaker()
2912     >>> timespan_quantization = (1, 8)
2913     >>> segment_maker.timespan_quantization = timespan_quantization
2914     >>> print(format(segment_maker))
2915     consort.tools.SegmentMaker(
2916         timespan_quantization=durationtools.Duration(1, 8),
2917     )
2918
2919     """
2920     return self._timespan_quantization
2921
2922 @timespan_quantization.setter
2923 def timespan_quantization(self, timespan_quantization):
2924     if timespan_quantization is not None:
2925         timespan_quantization = \
2926             durationtools.Duration(timespan_quantization)

```

```

2927     self._timespan_quantization = timespan_quantization
2928
2929     @property
2930     def voice_names(self):
2931         return self._voice_names
2932
2933     @property
2934     def repeat(self):
2935         return self._repeat
2936
2937     @repeat.setter
2938     def repeat(self, repeat):
2939         if repeat is not None:
2940             repeat = bool(repeat)
2941         self._repeat = repeat

```

## A.47 CONSORTTOOLS.SILENTTIMESPAN

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import markuptools
3 from abjad.tools import mathtools
4 from abjad.tools import timespantools
5
6
7 class SilentTimespan(timespantools.Timespan):
8     r'''A silent timespan.
9     '''
10
11     ### CLASS VARIABLES ###
12
13     __slots__ = (
14         '_layer',
15         '_voice_name',
16     )
17
18     ### INITIALIZER ###
19
20     def __init__(
21         self,
22         start_offset=mathtools.NegativeInfinity(),
23         stop_offset=mathtools.Infinity(),
24         layer=None,
25         voice_name=None,
26     ):
27         timespantools.Timespan.__init__(
28             self,
29             start_offset=start_offset,
30             stop_offset=stop_offset,
31         )
32         if layer is not None:
33             layer = int(layer)
34         self._layer = layer
35         self._voice_name = voice_name
36

```

```

37     ### PRIVATE METHODS #####
38
39     def _as_postscript(
40         self,
41         postscript_x_offset,
42         postscript_y_offset,
43         postscript_scale,
44     ):
45         start = (float(self.start_offset) * postscript_scale)
46         start -= postscript_x_offset
47         stop = (float(self.stop_offset) * postscript_scale)
48         stop -= postscript_x_offset
49         ps = markuptools.Postscript()
50         ps = ps.moveto(start, postscript_y_offset)
51         ps = ps.setdash([0.5])
52         ps = ps.lineto(stop, postscript_y_offset)
53         ps = ps.stroke()
54         ps = ps.moveto(start, postscript_y_offset + 0.75)
55         ps = ps.setdash()
56         ps = ps.lineto(start, postscript_y_offset - 0.75)
57         ps = ps.stroke()
58         ps = ps.moveto(stop, postscript_y_offset + 0.75)
59         ps = ps.lineto(stop, postscript_y_offset - 0.75)
60         ps = ps.stroke()
61         if self.layer is not None:
62             ps = ps.moveto(start, postscript_y_offset)
63             ps = ps.rmoveto(0.25, 0.5)
64             #ps = ps.scale(0.8, 0.8)
65             ps = ps.show(str(self.layer))
66             #ps = ps.scale(1.25, 1.25)
67         return ps
68
69     ### PUBLIC PROPERTIES #####
70
71     @property
72     def forbid_fusing(self):
73         return False
74
75     @property
76     def forbid_splitting(self):
77         return False
78
79     @property
80     def is_left_broken(self):
81         return False
82
83     @property
84     def is_right_broken(self):
85         return False
86
87     @property
88     def layer(self):
89         return self._layer
90

```

```

91     @property
92     def minimum_duration(self):
93         return 0
94
95     @property
96     def voice_name(self):
97         return self._voice_name

```

## A.48 CONSORTTOOLS.SIMPLEDYNAMICEXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import attach
3 from abjad import inspect_
4 from abjad import iterate
5 from abjad import override
6 from abjad.tools import abctools
7 from abjad.tools import durationtools
8 from abjad.tools import indicatortools
9 from abjad.tools import instrumenttools
10 from abjad.tools import lilypondparsertools
11 from abjad.tools import selectiontools
12 from abjad.tools import spannertools
13
14
15 class SimpleDynamicExpression(abctools.AbjadValueObject):
16     r'''A dynamic expression.
17
18     .. container:: example
19
20         :::
21
22         >>> import consort
23         >>> dynamic_expression = consort.SimpleDynamicExpression(
24             ...     hairpin_start_token='sfp',
25             ...     hairpin_stop_token='o',
26             ... )
27
28         :::
29
30         >>> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
31         >>> dynamic_expression(staff[2:-2])
32         >>> print(format(staff))
33         \new Staff {
34             c'8
35             d'8
36             \override Hairpin #'circled-tip = ##t
37             e'8 \> \sfp
38             f'8
39             g'8
40             a'8 \!
41             \revert Hairpin #'circled-tip
42             b'8
43             c''8
44         }

```

```

45
46 .. container:: example
47
48 :::
49
50     >>> dynamic_expression = consort.SimpleDynamicExpression(
51         ...      'f', 'p',
52         ...      )
53     >>> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
54     >>> dynamic_expression(staff[2:-2])
55     >>> print(format(staff))
56     \new Staff {
57         c'8
58         d'8
59         e'8 \> \f
60         f'8
61         g'8
62         a'8 \p
63         b'8
64         c''8
65     }
66
67     ''
68
69     ### CLASS VARIABLES ###
70
71     __slots__ = (
72         '_hairpin_start_token',
73         '_hairpin_stop_token',
74         '_minimum_duration',
75     )
76
77     ### INITIALIZER ###
78
79     def __init__(
80         self,
81         hairpin_start_token='p',
82         hairpin_stop_token=None,
83         minimum_duration=durationtools.Duration(1, 4),
84     ):
85         lilypond_parser = lilypondparsers.LilyPondParser
86         known_dynamics = list(lilypond_parser.list_known_dynamics())
87         known_dynamics.append('o')
88         assert hairpin_start_token in known_dynamics, \
89             (known_dynamics, hairpin_start_token)
90         if hairpin_stop_token is not None:
91             assert hairpin_stop_token in known_dynamics
92             assert hairpin_start_token != 'o' or hairpin_stop_token != 'o'
93         if hairpin_start_token == 'o':
94             assert not hairpin_stop_token is None
95         self._hairpin_start_token = hairpin_start_token
96         self._hairpin_stop_token = hairpin_stop_token
97         if minimum_duration is not None:
98             minimum_duration = durationtools.Duration(minimum_duration)

```

```

99         self._minimum_duration = minimum_duration
100
101     ### SPECIAL METHODS ###
102
103     def __call__(self, music, name=None):
104         if not isinstance(music, selectiontools.SliceSelection):
105             music = selectiontools.SliceSelection(music)
106         is_short_group = False
107         if len(music) < 2:
108             is_short_group = True
109         elif self.minimum_duration is not None:
110             if music.get_duration() < self.minimum_duration:
111                 is_short_group = True
112         instrument = inspect_(music[0]).get_effective(
113             instrumenttools.Instrument,
114             )
115         logical_ties = tuple(iterate(music).by_logical_tie(pitched=True))
116         if len(logical_ties) < 3:
117             if instrument == instrumenttools.Piano() or \
118                 instrument == instrumenttools.Percussion():
119                 is_short_group = True
120         grace_notes = None
121         previous_leaf = inspect_(music[0]).get_leaf(-1)
122         if previous_leaf is not None:
123             graces = inspect_(previous_leaf).get_grace_containers('after')
124             if graces:
125                 assert len(graces) == 1
126                 grace_notes = list(graces[0].select_leaves())
127                 music = selectiontools.ContiguousSelect(
128                     tuple(grace_notes) + tuple(music),
129                     )
130             start_token = self.hairpin_start_token
131             stop_token = self.hairpin_stop_token
132             if is_short_group or stop_token is None:
133                 if start_token == 'o':
134                     start_token = stop_token
135                 if start_token.startswith('fp'):
136                     start_token = start_token[1:]
137                 command = indicatortools.LilyPondCommand(start_token, 'right')
138                 attach(command, music[0], name=name)
139                 return
140             start_ordinal = NegativeInfinity
141             if start_token != 'o':
142                 start_ordinal = indicatortools.Dynamic.dynamic_name_to_dynamic_ordinal(
143                     start_token)
144             stop_ordinal = NegativeInfinity
145             if stop_token != 'o':
146                 stop_ordinal = indicatortools.Dynamic.dynamic_name_to_dynamic_ordinal(stop_token)
147             items = []
148             is_circled = False
149             if start_ordinal < stop_ordinal:
150                 if start_token != 'o':
151                     items.append(start_token)
152                 else:

```

```

153         is_circled = True
154         items.append('<')
155         items.append(stop_token)
156     elif stop_ordinal < start_ordinal:
157         items.append(start_token)
158         items.append('>')
159     if stop_token != 'o':
160         items.append(stop_token)
161     else:
162         items.append('!')
163         is_circled = True
164     hairpin_descriptor = ' '.join(items)
165     hairpin = spanertools.Hairpin(
166         descriptor=hairpin_descriptor,
167         include_rests=False,
168     )
169     if is_circled:
170         override(hairpin).hairpin.circled_tip = True
171     attach(hairpin, music, name=name)
172
173     ### PUBLIC PROPERTIES ###
174
175     @property
176     def hairpin_start_token(self):
177         return self._hairpin_start_token
178
179     @property
180     def hairpin_stop_token(self):
181         return self._hairpin_stop_token
182
183     @property
184     def minimum_duration(self):
185         return self._minimum_duration

```

## A.49 CONSORTTOOLS.STOPTRILLSpan

```

1 # -*- encoding: utf-8 -*-
2 from abjad import inspect_
3 from abjad.tools import abctools
4 from abjad.tools import scoretools
5 from abjad.tools import systemtools
6
7
8 class StopTrillSpan(abctools.AbjadValueObject):
9
10     __slots__ = ()
11
12     def _get_lilypond_format_bundle(self, component):
13         import consort
14         parentage = inspect_(component).get_parentage()
15         prototype = scoretools.GraceContainer
16         grace_container = None
17         for parent in parentage:
18             if isinstance(parent, prototype):

```

```

19         grace_container = parent
20         break
21     if grace_container is None:
22         return
23     prototype = consort.ConsortTrillSpanner
24     carrier = grace_container._carrier
25     spanners = inspect_(carrier).get_spanners(prototype)
26     if not spanners:
27         return
28     bundle = systemtools.LilyPondFormatBundle()
29     bundle.right.spanner_stops.append(r'\stopTrillSpan')
30     return bundle

```

## A.50 CONSORTTOOLS.STRINGCONTACTSPANNER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 from abjad import inspect_
4 from abjad.tools import indicatortools
5 from abjad.tools import markuptools
6 from abjad.tools import spannertools
7 from abjad.tools import scoretools
8 from abjad.tools import lilypondnametools
9
10
11 class StringContactSpanner(spannertools.Spanner):
12     r'''String contact spanner.
13
14     :::
15
16     >>> import consort
17     >>> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
18     >>> attach(indicatortools.StringContactPoint('sul tasto'),
19     ...     staff[2], scope=Staff)
20     >>> attach(indicatortools.StringContactPoint('sul tasto'),
21     ...     staff[3], scope=Staff)
22     >>> attach(indicatortools.StringContactPoint('ordinario'),
23     ...     staff[4], scope=Staff)
24     >>> attach(indicatortools.StringContactPoint('pizzicato'),
25     ...     staff[5], scope=Staff)
26     >>> attach(indicatortools.StringContactPoint('ordinario'),
27     ...     staff[6], scope=Staff)
28     >>> attach(indicatortools.StringContactPoint('sul ponticello'),
29     ...     staff[7], scope=Staff)
30     >>> attach(consort.StringContactSpanner(), staff[:])
31
32 .. doctest::
33
34     >>> print(format(staff))
35     \new Staff {
36         c'8 ^ \markup {
37             \vcenter
38                 \italic
39                 \caps

```

```

40          S.T.
41      }
42 d'8
43 e'8
44 \once \override TextSpanner.arrow-width = 0.25
45 \once \override TextSpanner.bound-details.left-broken.text = ##f
46 \once \override TextSpanner.bound-details.left.stencil-align-dir-y = #center
47 \once \override TextSpanner.bound-details.left.text = \markup {
48     \halign
49         #0
50     \halign
51         #0
52     \concat
53     {
54         \hspace
55             #1.5
56         \punctsize
57             \caps
58             S.T.
59         \hspace
60             #1.5
61     }
62 }
63 \once \override TextSpanner.bound-details.right-broken.padding = 0
64 \once \override TextSpanner.bound-details.right.arrow = ##t
65 \once \override TextSpanner.bound-details.right.padding = 0
66 \once \override TextSpanner.bound-details.right.stencil-align-dir-y = #center
67 \once \override TextSpanner.bound-details.right.text = \markup {
68     \halign
69         #0
70     \halign
71         #0
72     \concat
73     {
74         \hspace
75             #1.5
76         \caps
77             Ord.
78         \hspace
79             #1.5
80     }
81 }
82 \once \override TextSpanner.dash-fraction = 0.25
83 \once \override TextSpanner.dash-period = 1
84 f'8 \startTextSpan
85 g'8 \stopTextSpan ^ \markup {
86     \vcenter
87         \italic
88         \caps
89             Ord.
90     }
91 a'8 ^ \markup {
92     \vcenter
93         \italic

```

```

94             \caps
95                 Pizz.
96             }
97             \once \override TextSpanner.arrow-width = 0.25
98             \once \override TextSpanner.bound-details.left-broken.text = ##f
99             \once \override TextSpanner.bound-details.left.stencil-align-dir-y = #center
100            \once \override TextSpanner.bound-details.left.text = \markup {
101                \halign
102                    #0
103                \halign
104                    #0
105                \concat
106                {
107                    \hspace
108                        #1.5
109                    \caps
110                        Ord.
111                    \hspace
112                        #1.5
113                }
114            }
115            \once \override TextSpanner.bound-details.right-broken.padding = 0
116            \once \override TextSpanner.bound-details.right.arrow = ##
117            \once \override TextSpanner.bound-details.right.padding = 0
118            \once \override TextSpanner.bound-details.right.stencil-align-dir-y = #center
119            \once \override TextSpanner.bound-details.right.text = \markup {
120                \halign
121                    #0
122                \halign
123                    #0
124                \concat
125                {
126                    \hspace
127                        #1.5
128                    \caps
129                        S.P.
130                    \hspace
131                        #1.5
132                }
133            }
134            \once \override TextSpanner.dash-fraction = 0.25
135            \once \override TextSpanner.dash-period = 1
136            b'8 \stopTextSpan \startTextSpan
137            c''8 \stopTextSpan
138        }
139    :::
140
141
142     >>> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
143     >>> attach(indicatorTools.StringContactPoint('ordinario'),
144     ...     staff[0], scope=Staff)
145     >>> attach(indicatorTools.StringContactPoint('sul tasto'),
146     ...     staff[2], scope=Staff)
147     >>> attach(indicatorTools.StringContactPoint('ordinario'),

```

```

148     ...      staff[4], scope=Staff)
149     >>> attach(indicatortools.StringContactPoint('sul tasto'),
150     ...      staff[6], scope=Staff)
151     >>> attach(consort.StringContactSpanner(), staff[:])
152
153 .. doctest::
154
155     >>> print(format(staff))
156
157 \new Staff {
158     \once \override TextSpanner.arrow-width = 0.25
159     \once \override TextSpanner.bound-details.left-broken.text = ##f
160     \once \override TextSpanner.bound-details.left.stencil-align-dir-y = #center
161     \once \override TextSpanner.bound-details.left.text = \markup {
162         \halign
163             #0
164         \halign
165             #0
166         \concat
167             {
168                 \hspace
169                     #1.5
170                 \caps
171                     Ord.
172                 \hspace
173                     #1.5
174             }
175         \once \override TextSpanner.bound-details.right-broken.padding = 0
176         \once \override TextSpanner.bound-details.right.arrow = ##t
177         \once \override TextSpanner.bound-details.right.padding = 5
178         \once \override TextSpanner.bound-details.right.stencil-align-dir-y = #center
179         \once \override TextSpanner.dash-fraction = 0.25
180         \once \override TextSpanner.dash-period = 1
181         c'8 \startTextSpan
182         d'8
183         \once \override TextSpanner.arrow-width = 0.25
184         \once \override TextSpanner.bound-details.left-broken.text = ##f
185         \once \override TextSpanner.bound-details.left.stencil-align-dir-y = #center
186         \once \override TextSpanner.bound-details.left.text = \markup {
187             \halign
188                 #0
189             \halign
190                 #0
191             \concat
192                 {
193                     \hspace
194                         #1.5
195                     \caps
196                         S.T.
197                     \hspace
198                         #1.5
199                 }
200             }
201         \once \override TextSpanner.bound-details.right-broken.padding = 0

```

```

202     \once \override TextSpanner.bound-details.right.arrow = ##
203     \once \override TextSpanner.bound-details.right.padding = 5
204     \once \override TextSpanner.bound-details.right.stencil-align-dir-y = #center
205     \once \override TextSpanner.dash-fraction = 0.25
206     \once \override TextSpanner.dash-period = 1
207     e'8 \stopTextSpan \startTextSpan
208     f'8
209     \once \override TextSpanner.arrow-width = 0.25
210     \once \override TextSpanner.bound-details.left-broken.text = ##f
211     \once \override TextSpanner.bound-details.left.stencil-align-dir-y = #center
212     \once \override TextSpanner.bound-details.left.text = \markup {
213         \halign
214             #0
215         \halign
216             #0
217         \concat
218             {
219                 \hspace
220                     #1.5
221                 \caps
222                     Ord.
223                 \hspace
224                     #1.5
225             }
226         }
227     \once \override TextSpanner.bound-details.right-broken.padding = 0
228     \once \override TextSpanner.bound-details.right.arrow = ##
229     \once \override TextSpanner.bound-details.right.padding = 0
230     \once \override TextSpanner.bound-details.right.stencil-align-dir-y = #center
231     \once \override TextSpanner.bound-details.right.text = \markup {
232         \halign
233             #0
234         \halign
235             #0
236         \concat
237             {
238                 \hspace
239                     #1.5
240                 \caps
241                     S.T.
242                 \hspace
243                     #1.5
244             }
245         }
246     \once \override TextSpanner.dash-fraction = 0.25
247     \once \override TextSpanner.dash-period = 1
248     g'8 \stopTextSpan \startTextSpan
249     a'8
250     b'8 \stopTextSpan
251     c''8
252     }
253     ...
254

```

```

256     """ CLASS VARIABLES """
257
258     __slots__ = (
259         )
260
261     """ INITIALIZER """
262
263     def __init__(
264         self,
265         overrides=None,
266         ):
267         spannertools.Spanner.__init__(
268             self,
269             overrides=overrides,
270             )
271
272     """ PRIVATE METHODS """
273
274     def _get_annotations(self, leaf):
275         import consort
276
277         leaves = self._get_leaves()
278         index = leaves.index(leaf)
279         prototype = indicatortools.StringContactPoint
280         agent = inspect_(leaf)
281         pizzicato = indicatortools.StringContactPoint('pizzicato')
282
283         next_attached = None
284         for i in range(index + 1, len(leaves)):
285             next_leaf = leaves[i]
286             indicators = next_leaf._get_indicators(
287                 indicatortools.StringContactPoint,
288                 )
289             if indicators:
290                 next_attached = indicators[0]
291                 break
292
293         actually_attached = current_attached = None
294         indicators = inspect_(leaf).get_indicators(prototype)
295         if indicators:
296             actually_attached = current_attached = indicators[0]
297         if self._is_my_first_leaf(leaf) and current_attached is None:
298             current_attached = next_attached
299
300         next_different = None
301         next_after_different = None
302         next_next_different = None
303         for i in range(index + 1, len(leaves)):
304             next_leaf = leaves[i]
305             indicators = next_leaf._get_indicators(
306                 indicatortools.StringContactPoint,
307                 )
308             if indicators:
309                 indicator = indicators[0]

```

```

310     if next_different is not None:
311         if next_after_different is None:
312             next_after_different = indicator
313         if indicator != next_different:
314             next_next_different = indicator
315             break
316     if indicator != current_attached and next_different is None:
317         next_different = indicator
318
319     n = -1
320     if actually_attached is None:
321         n = 0
322     previous_effective = agent.get_effective(prototype, n=n)
323     previous_attached = None
324     for i in reversed(range(index)):
325         previous_leaf = leaves[i]
326         indicators = previous_leaf._get_indicators(
327             indicatortools.StringContactPoint,
328             )
329         if indicators:
330             previous_attached = indicators[0]
331             break
332     if current_attached is not None and \
333         not self._is_my_first_leaf(leaf) and \
334         previous_attached is None:
335         previous_attached = current_attached
336
337     previous_different = None
338     for i in reversed(range(index)):
339         previous_leaf = leaves[i]
340         indicators = previous_leaf._get_indicators(
341             indicatortools.StringContactPoint,
342             )
343         if indicators:
344             indicator = indicators[0]
345             if indicator != current_attached:
346                 previous_different = indicator
347
348     has_start_markup = False
349     if current_attached is not None and \
350         next_attached is not None and \
351         current_attached != pizzicato and \
352         next_different != pizzicato and \
353         current_attached != next_attached:
354         has_start_markup = True
355
356     has_stop_markup = False
357     if current_attached is not None and \
358         current_attached != pizzicato and \
359         (next_next_different == pizzicato or next_next_different is None):
360         has_stop_markup = True
361     elif current_attached is not None and \
362         next_different == next_after_different:
363         has_stop_markup = True

```

```

364
365     stops_text_spanner = False
366     if current_attached is not None and \
367         previous_different is not None and \
368             current_attached != pizzicato and \
369                 previous_different != pizzicato and \
370                     current_attached != previous_attached:
371                         stops_text_spanner = True
372
373     if self._is_my_first_leaf(leaf) and \
374         current_attached is None and \
375             previous_effective is not None:
376                 current_attached = previous_effective
377
378     is_cautionary = False
379     if current_attached and current_attached == previous_attached:
380         is_cautionary = True
381     elif current_attached and current_attached == previous_effective:
382         is_cautionary = True
383
384     current_markup = None
385     if current_attached is not None:
386         current_markup = current_attached.markup
387     if current_attached == previous_attached == next_attached and \
388         current_attached != pizzicato:
389         current_markup = None
390     if current_attached == previous_effective and \
391         next_attached is None and \
392             current_attached != pizzicato and \
393                 not self._is_my_first_leaf(leaf):
394                     current_markup = None
395
396     if current_markup is not None:
397         if is_cautionary:
398             current_markup = current_markup.parenthesize()
399
400     results = (
401         current_attached,
402         current_markup,
403         has_start_markup,
404         has_stop_markup,
405         is_cautionary,
406         next_attached,
407         next_different,
408         previous_attached,
409         previous_effective,
410         stops_text_spanner,
411     )
412
413     consort.debug(leaf)
414     consort.debug('\t', 'actually_attached', actually_attached)
415     consort.debug('\t', 'current_attached', current_attached)
416     consort.debug('\t', 'current_markup', current_markup)
417     consort.debug('\t', 'has_start_markup', has_start_markup)

```

```

418     consort.debug('\t', 'has_stop_markup', has_stop_markup)
419     consort.debug('\t', 'is_cautionary', is_cautionary)
420     consort.debug('\t', 'next_attached', next_attached)
421     consort.debug('\t', 'next_different', next_different)
422     consort.debug('\t', 'next_after_different', next_after_different)
423     consort.debug('\t', 'next_next_different', next_next_different)
424     consort.debug('\t', 'previous_attached', previous_attached)
425     consort.debug('\t', 'previous_effective', previous_effective)
426     consort.debug('\t', 'stops_text_spinner', stops_text_spinner)
427
428     return results
429
430 def _get_lilypond_format_bundle(self, leaf):
431     import consort
432
433     lilypond_format_bundle = self._get_basic_lilypond_format_bundle(leaf)
434     if not isinstance(leaf, scoretools.Leaf):
435         return lilypond_format_bundle
436     (
437         current_attached,
438         current_markup,
439         has_start_markup,
440         has_stop_markup,
441         is_cautionary,
442         next_attached,
443         next_different,
444         previous_attached,
445         previous_effective,
446         stops_text_spinner,
447     ) = self._get_annotations(leaf)
448
449     if current_markup is None:
450         consort.debug('\tRETURNING++++++++++++')
451         consort.debug()
452     return lilypond_format_bundle
453
454     if has_start_markup and has_stop_markup:
455         self._add_segment_start_contributions(
456             lilypond_format_bundle,
457             start_markup=current_markup,
458             stop_markup=next_different.markup,
459         )
460     elif has_start_markup:
461         self._add_segment_start_contributions(
462             lilypond_format_bundle,
463             start_markup=current_markup,
464         )
465
466     if stops_text_spinner:
467         self._add_segment_stop_contributions(lilypond_format_bundle)
468
469     should_attach_markup = False
470     if current_markup and \
471         not has_start_markup and \

```

```

472         not has_stop_markup:
473             should_attach_markup = True
474     if current_markup and \
475         previous_attached is None and \
476         not has_start_markup:
477         should_attach_markup = True
478     if current_markup and \
479         current_attached == indicatortools.StringContactPoint('pizzicato'):
480         should_attach_markup = True
481     if current_attached is not None and \
482         current_attached == next_attached and \
483         previous_attached != current_attached and \
484         previous_attached is not None:
485         should_attach_markup = False
486
487     consort.debug('Attaching??', should_attach_markup)
488     if should_attach_markup:
489         current_markup = markuptools.Markup(current_markup, Up)
490         current_markup = current_markup.italic()
491         current_markup = current_markup.vcenter()
492         lilypond_format_bundle.right.markup.append(current_markup)
493
494     consort.debug(format(lilypond_format_bundle))
495     consort.debug()
496
497     return lilypond_format_bundle
498
499     def _add_segment_start_contributions(
500         self,
501         lilypond_format_bundle,
502         start_markup=None,
503         stop_markup=None,
504         ):
505         right_padding = 5
506         if stop_markup is not None:
507             right_padding = 0
508         line_segment = indicatortools.Arrow(
509             dash_fraction=0.25,
510             dash_period=1,
511             right_padding=right_padding,
512             )
513
514         if start_markup is not None:
515             start_markup = markuptools.Markup.concat([
516                 markuptools.Markup.hspace(1.5),
517                 start_markup,
518                 markuptools.Markup.hspace(1.5),
519                 ])
520             start_markup = start_markup.halign(0)
521
522         if stop_markup is not None:
523             stop_markup = markuptools.Markup.concat([
524                 markuptools.Markup.hspace(1.5),
525                 stop_markup,

```

```

526         markuptools.Markup.hspace(1.5),
527     ])
528     stop_markup = stop_markup.halign(0)
529
530     string = r'\startTextSpan'
531     lilypond_format_bundle.right.spanner_starts.append(string)
532     overrides = line_segment._get_lilypond_grob_overrides()
533     for override_ in overrides:
534         override_string = '\n'.join(override_.override_format_pieces)
535         lilypond_format_bundle.grob_overrides.append(override_string)
536     if start_markup:
537         override_ = lilypondnametools.LilyPondGrobOverride(
538             grob_name='TextSpanner',
539             is_once=True,
540             property_path=(
541                 'bound-details',
542                 'left',
543                 'text',
544             ),
545             value=start_markup.halign(0),
546         )
547         override_string = '\n'.join(override_.override_format_pieces)
548         lilypond_format_bundle.grob_overrides.append(override_string)
549     if stop_markup:
550         override_ = lilypondnametools.LilyPondGrobOverride(
551             grob_name='TextSpanner',
552             is_once=True,
553             property_path=(
554                 'bound-details',
555                 'right',
556                 'text',
557             ),
558             value=stop_markup.halign(0),
559         )
560         override_string = '\n'.join(override_.override_format_pieces)
561         lilypond_format_bundle.grob_overrides.append(override_string)
562
563     def _add_segment_stop_contributions(
564         self,
565         lilypond_format_bundle,
566     ):
567         string = r'\stopTextSpan'
568         lilypond_format_bundle.right.spanner_stops.append(string)

```

## A.51 CONSORTTOOLS.STRINGQUARTETSCORETEMPLATE

```

1 # -*- encoding: utf-8 -*-
2 from abjad import detach
3 from abjad import iterate
4 from abjad.tools import indicatortools
5 from abjad.tools import instrumenttools
6 from abjad.tools import markuptools
7 from abjad.tools import scoretools
8 from consort.tools.ScoreTemplate import ScoreTemplate

```

```

9
10
11 class StringQuartetScoreTemplate(ScoreTemplate):
12     r'''A string quartet score template.
13
14     :::
15
16     >>> import consort
17     >>> template = consort.StringQuartetScoreTemplate()
18     >>> score = template()
19     >>> print(format(score))
20     \context Score = "String Quartet Score" <-
21         \tag #'time
22         \context TimeSignatureContext = "Time Signature Context" {
23             }
24         \tag #'violin-1
25         \context StringPerformerGroup = "Violin 1 Performer Group" \with {
26             instrumentName = \markup {
27                 \hcenter-in
28                     #10
29                     "Violin 1"
30             }
31             shortInstrumentName = \markup {
32                 \hcenter-in
33                     #10
34                     "Vln. 1"
35             }
36         } <-
37             \context BowingStaff = "Violin 1 Bowing Staff" {
38                 \clef "percussion"
39                 \context Voice = "Violin 1 Bowing Voice" {
40                     }
41             }
42             \context FingeringStaff = "Violin 1 Fingering Staff" {
43                 \clef "treble"
44                 \context Voice = "Violin 1 Fingering Voice" {
45                     }
46             }
47         >>
48         \tag #'violin-2
49         \context StringPerformerGroup = "Violin 2 Performer Group" \with {
50             instrumentName = \markup {
51                 \hcenter-in
52                     #10
53                     "Violin 2"
54             }
55             shortInstrumentName = \markup {
56                 \hcenter-in
57                     #10
58                     "Vln. 2"
59             }
60         } <-
61             \context BowingStaff = "Violin 2 Bowing Staff" {
62                 \clef "percussion"

```

```

63          \context Voice = "Violin 2 Bowing Voice" {
64          }
65      }
66      \context FingeringStaff = "Violin 2 Fingering Staff" {
67          \clef "treble"
68          \context Voice = "Violin 2 Fingering Voice" {
69          }
70      }
71  >>
72  \tag #'viola
73  \context StringPerformerGroup = "Viola Performer Group" \with {
74      instrumentName = \markup {
75          \hcenter-in
76              #10
77              Viola
78          }
79      shortInstrumentName = \markup {
80          \hcenter-in
81              #10
82              Va.
83          }
84  } <<
85      \context BowingStaff = "Viola Bowing Staff" {
86          \clef "percussion"
87          \context Voice = "Viola Bowing Voice" {
88          }
89      }
90      \context FingeringStaff = "Viola Fingering Staff" {
91          \clef "alto"
92          \context Voice = "Viola Fingering Voice" {
93          }
94      }
95  >>
96  \tag #'cello
97  \context StringPerformerGroup = "Cello Performer Group" \with {
98      instrumentName = \markup {
99          \hcenter-in
100             #10
101             Cello
102         }
103     shortInstrumentName = \markup {
104         \hcenter-in
105             #10
106             Vc.
107         }
108 } <<
109     \context BowingStaff = "Cello Bowing Staff" {
110         \clef "percussion"
111         \context Voice = "Cello Bowing Voice" {
112         }
113     }
114     \context FingeringStaff = "Cello Fingering Staff" {
115         \clef "bass"
116         \context Voice = "Cello Fingering Voice" {

```

```

117         }
118     }
119     >>
120     >>
121
122 :::
123
124     >>> for item in sorted(template.context_name_abbreviations.items()):
125         ...     item
126         ...
127         ('cello', 'Cello Performer Group')
128         ('cello_lh', 'Cello Fingering Voice')
129         ('cello_rh', 'Cello Bowing Voice')
130         ('viola', 'Viola Performer Group')
131         ('viola_lh', 'Viola Fingering Voice')
132         ('viola_rh', 'Viola Bowing Voice')
133         ('violin_1', 'Violin 1 Performer Group')
134         ('violin_1_lh', 'Violin 1 Fingering Voice')
135         ('violin_1_rh', 'Violin 1 Bowing Voice')
136         ('violin_2', 'Violin 2 Performer Group')
137         ('violin_2_lh', 'Violin 2 Fingering Voice')
138         ('violin_2_rh', 'Violin 2 Bowing Voice')
139
140 :::
141
142     >>> for item in sorted(template.composite_context_pairs.items()):
143         ...     item
144         ...
145         ('cello', ('cello_rh', 'cello_lh'))
146         ('viola', ('viola_rh', 'viola_lh'))
147         ('violin_1', ('violin_1_rh', 'violin_1_lh'))
148         ('violin_2', ('violin_2_rh', 'violin_2_lh'))
149
150 :::
151
152     >>> template = consort.StringQuartetScoreTemplate(split=False)
153     >>> score = template()
154     >>> print(format(score))
155 \context Score = "String Quartet Score" <<
156     \tag #'time
157     \context TimeSignatureContext = "Time Signature Context" {
158     }
159     \tag #'violin-1
160     \context StringPerformerGroup = "Violin 1 Performer Group" \with {
161         instrumentName = \markup {
162             \hcenter-in
163                 #10
164                 "Violin 1"
165             }
166         shortInstrumentName = \markup {
167             \hcenter-in
168                 #10
169                 "Vln. 1"
170         }

```

```

171 } <<
172     \context StringStaff = "Violin 1 Staff" {
173         \context Voice = "Violin 1 Voice" {
174             \clef "treble"
175         }
176     }
177     >>
178     \tag #'violin-2
179     \context StringPerformerGroup = "Violin 2 Performer Group" \with {
180         instrumentName = \markup {
181             \hcenter-in
182                 #10
183                 "Violin 2"
184             }
185         shortInstrumentName = \markup {
186             \hcenter-in
187                 #10
188                 "Vln. 2"
189             }
190     } <<
191         \context StringStaff = "Violin 2 Staff" {
192             \context Voice = "Violin 2 Voice" {
193                 \clef "treble"
194             }
195         }
196         >>
197         \tag #'viola
198         \context StringPerformerGroup = "Viola Performer Group" \with {
199             instrumentName = \markup {
200                 \hcenter-in
201                     #10
202                     Viola
203                 }
204             shortInstrumentName = \markup {
205                 \hcenter-in
206                     #10
207                     Va.
208                 }
209     } <<
210         \context StringStaff = "Viola Staff" {
211             \context Voice = "Viola Voice" {
212                 \clef "alto"
213             }
214         }
215         >>
216         \tag #'cello
217         \context StringPerformerGroup = "Cello Performer Group" \with {
218             instrumentName = \markup {
219                 \hcenter-in
220                     #10
221                     Cello
222                 }
223             shortInstrumentName = \markup {
224                 \hcenter-in

```

```

225             #10
226             Vc.
227         }
228     } <<
229         \context StringStaff = "Cello Staff" {
230             \context Voice = "Cello Voice" {
231                 \clef "bass"
232             }
233         }
234     >>
235     >>
236
237     :::
238
239     >>> for item in sorted(template.context_name_abbreviations.items()):
240         ...     item
241         ...
242         ('cello', 'Cello Voice')
243         ('viola', 'Viola Voice')
244         ('violin_1', 'Violin 1 Voice')
245         ('violin_2', 'Violin 2 Voice')
246
247     :::
248
249     >>> for item in sorted(template.composite_context_pairs.items()):
250         ...     item
251         ...
252
253     """
254
255
256     ### CLASS VARIABLES ###
257
258     __slots__ = (
259         '_split',
260         '_without_instruments',
261     )
262
263     ### INITIALIZER ###
264
265     def __init__(self, split=True, without_instruments=None):
266         if split is not None:
267             split = bool(split)
268         self._split = split
269         self._without_instruments = without_instruments
270         ScoreTemplate.__init__(self)
271
272     ### SPECIAL METHODS ###
273
274     def __call__(self):
275         import consort
276
277         manager = consort.ScoreTemplateManager
278

```

```

279     time_signature_context = manager.make_time_signature_context()
280
281     violin_one = manager.make_single_string_performer(
282         clef=indicatortools.Clef('treble'),
283         instrument=instrumenttools.Violin(
284             instrument_name='violin 1',
285             instrument_name_markup=markuptools.Markup(
286                 'Violin 1').hcenter_in(10),
287             short_instrument_name='vln. 1',
288             short_instrument_name_markup=markuptools.Markup(
289                 'Vln. 1').hcenter_in(10)
290             ),
291             split=self.split,
292             score_template=self,
293         )
294
295     violin_two = manager.make_single_string_performer(
296         clef=indicatortools.Clef('treble'),
297         instrument=instrumenttools.Violin(
298             instrument_name='violin 2',
299             instrument_name_markup=markuptools.Markup(
300                 'Violin 2').hcenter_in(10),
301             short_instrument_name='vln. 2',
302             short_instrument_name_markup=markuptools.Markup(
303                 'Vln. 2').hcenter_in(10)
304             ),
305             split=self.split,
306             score_template=self,
307         )
308
309     viola = manager.make_single_string_performer(
310         clef=indicatortools.Clef('alto'),
311         instrument=instrumenttools.Viola(
312             instrument_name='viola',
313             instrument_name_markup=markuptools.Markup(
314                 'Viola').hcenter_in(10),
315             short_instrument_name='va.',
316             short_instrument_name_markup=markuptools.Markup(
317                 'Va.').hcenter_in(10)
318             ),
319             split=self.split,
320             score_template=self,
321         )
322
323     cello = manager.make_single_string_performer(
324         clef=indicatortools.Clef('bass'),
325         instrument=instrumenttools.Cello(
326             instrument_name='cello',
327             instrument_name_markup=markuptools.Markup(
328                 'Cello').hcenter_in(10),
329             short_instrument_name='vc.',
330             short_instrument_name_markup=markuptools.Markup(
331                 'Vc.').hcenter_in(10)
332             ),

```

```

333         split=self.split,
334         score_template=self,
335     )
336
337     score = scoretools.Score(
338         [
339             time_signature_context,
340             violin_one,
341             violin_two,
342             viola,
343             cello,
344         ],
345         name='String Quartet Score',
346     )
347
348     if self.without_instruments:
349         for staff in iterate(score).by_class(scoretools.Context):
350             detach(instrumenttools.Instrument, staff)
351
352     return score
353
354     ### PUBLIC PROPERTIES ###
355
356     @property
357     def split(self):
358         return self._split
359
360     @property
361     def without_instruments(self):
362         return self._without_instruments

```

## A.52 CONSORTTOOLS.TALEATIMESPANMAKER

```

1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import collections
4 from abjad.tools import datastructuretools
5 from abjad.tools import durationtools
6 from abjad.tools import rhythmmakertools
7 from abjad.tools import timespanools
8 from consort.tools.TimespanMaker import TimespanMaker
9
10
11 class TaleaTimespanMaker(TimespanMaker):
12     r'''A talea timespan maker.
13
14     :::
15
16     >>> import consort
17     >>> timespan_maker = consort.TaleaTimespanMaker(
18     ...     initial_silence_talea=rhythmmakertools.Talea(
19     ...         counts=(0, 4),
20     ...         denominator=16,
21     ...     )

```

```

22     ...
23     )
24
25     >>> print(format(timespan_maker))
26     consort.tools.TaleaTimespanMaker(
27         initial_silence_talea=rhythmmakertools.Talea(
28             counts=(0, 4),
29             denominator=16,
30             ),
31             playing_talea=rhythmmakertools.Talea(
32                 counts=(4,),
33                 denominator=16,
34                 ),
35                 playing_groupings=(1,),
36                 repeat=True,
37                 silence_talea=rhythmmakertools.Talea(
38                     counts=(4,),
39                     denominator=16,
40                     ),
41                     step_anchor=Right,
42                     synchronize_groupings=False,
43                     synchronize_step=False,
44                     )
45
46 :::
47
48     >>> import collections
49
50     >>> music_specifiers = collections.OrderedDict([
51         ('Violin', None),
52         ('Viola', None),
53         ])
54
55     >>> target_timespan = timespantools.Timespan(0, 1)
56     >>> timespan_inventory = timespan_maker(
57         ...     music_specifiers=music_specifiers,
58         ...     target_timespan=target_timespan,
59         ...     )
60
61     >>> print(format(timespan_inventory))
62     timespantools.TimespanInventory(
63         [
64             consort.tools.PerformedTimespan(
65                 start_offset=durationtools.Offset(0, 1),
66                 stop_offset=durationtools.Offset(1, 4),
67                 voice_name='Violin',
68                 ),
69             consort.tools.PerformedTimespan(
70                 start_offset=durationtools.Offset(1, 4),
71                 stop_offset=durationtools.Offset(1, 2),
72                 voice_name='Viola',
73                 ),
74             consort.tools.PerformedTimespan(
75                 start_offset=durationtools.Offset(1, 2),
76                 stop_offset=durationtools.Offset(3, 4),
77                 voice_name='Violin',
78                 ),
79             consort.tools.PerformedTimespan(
80                 start_offset=durationtools.Offset(3, 4),
81

```

```

76             stop_offset=durationtools.Offset(1, 1),
77             voice_name='Viola',
78             ),
79         ],
80     )
81
82 :::
83
84     >>> timespan_maker = new(timespan_maker,
85     ...     initial_silence_talea=None,
86     ...     synchronize_step=True,
87     ...     )
88     >>> timespan_inventory = timespan_maker(
89     ...     music_specifiers=music_specifiers,
90     ...     target_timespan=target_timespan,
91     ...     )
92     >>> print(format(timespan_inventory))
93 timespanmaker.TimespanInventory(
94     [
95         consort.tools.PerformedTimespan(
96             start_offset=durationtools.Offset(0, 1),
97             stop_offset=durationtools.Offset(1, 4),
98             voice_name='Viola',
99             ),
100        consort.tools.PerformedTimespan(
101            start_offset=durationtools.Offset(0, 1),
102            stop_offset=durationtools.Offset(1, 4),
103            voice_name='Violin',
104            ),
105        consort.tools.PerformedTimespan(
106            start_offset=durationtools.Offset(1, 2),
107            stop_offset=durationtools.Offset(3, 4),
108            voice_name='Viola',
109            ),
110        consort.tools.PerformedTimespan(
111            start_offset=durationtools.Offset(1, 2),
112            stop_offset=durationtools.Offset(3, 4),
113            voice_name='Violin',
114            ),
115    ],
116 )
117
118 :::
119
120     >>> timespan_maker = new(timespan_maker,
121     ...     initial_silence_talea=rhythmmakertools.Talea(
122     ...         counts=(0, 2),
123     ...         denominator=16,
124     ...         ),
125     ...     )
126     >>> timespan_inventory = timespan_maker(
127     ...     music_specifiers=music_specifiers,
128     ...     target_timespan=target_timespan,
129     ...     )

```

```

130     >>> print(format(timespan_inventory))
131     timespanInventory(
132         [
133             consort.tools.PerformedTimespan(
134                 start_offset=durationtools.Offset(0, 1),
135                 stop_offset=durationtools.Offset(1, 4),
136                 voice_name='Violin',
137             ),
138             consort.tools.PerformedTimespan(
139                 start_offset=durationtools.Offset(1, 8),
140                 stop_offset=durationtools.Offset(3, 8),
141                 voice_name='Viola',
142             ),
143             consort.tools.PerformedTimespan(
144                 start_offset=durationtools.Offset(5, 8),
145                 stop_offset=durationtools.Offset(7, 8),
146                 voice_name='Violin',
147             ),
148             consort.tools.PerformedTimespan(
149                 start_offset=durationtools.Offset(3, 4),
150                 stop_offset=durationtools.Offset(1, 1),
151                 voice_name='Viola',
152             ),
153         ]
154     )
155
156     """
157
158     ### CLASS VARIABLES ###
159
160     __slots__ = (
161         '_fuse_groups',
162         '_initial_silence_talea',
163         '_padding',
164         '_playing_talea',
165         '_playing_groupings',
166         '_reflect',
167         '_repeat',
168         '_silence_talea',
169         '_step_anchor',
170         '_synchronize_groupings',
171         '_synchronize_step',
172     )
173
174     ### INITIALIZER ###
175
176     def __init__(
177         self,
178         fuse_groups=None,
179         initial_silence_talea=None,
180         output_masks=None,
181         padding=None,
182         playing_talea=rhythmmakertools.Talea(
183             counts=[4],

```

```

184         denominator=16,
185         ),
186     playing_groupings=(1,),
187     reflect=None,
188     repeat=True,
189     seed=None,
190     silence_talea=rhythmmakertools.Talea(
191         counts=[4],
192         denominator=16,
193         ),
194     step_anchor=Right,
195     synchronize_groupings=False,
196     synchronize_step=False,
197     timespanSpecifier=None,
198     ):
199     TimespanMaker.__init__(
200         self,
201         output_masks=output_masks,
202         padding=padding,
203         seed=seed,
204         timespanSpecifier=timespanSpecifier,
205         )
206
207     if fuse_groups is not None:
208         fuse_groups = bool(fuse_groups)
209     self._fuse_groups = fuse_groups
210
211     if initial_silence_talea is not None:
212         assert isinstance(initial_silence_talea, rhythmmakertools.Talea)
213         assert initial_silence_talea.counts
214         assert all(0 <= x for x in initial_silence_talea.counts)
215     self._initial_silence_talea = initial_silence_talea
216
217     assert isinstance(playing_talea, rhythmmakertools.Talea)
218     assert playing_talea.counts
219     assert all(0 < x for x in playing_talea.counts)
220     self._playing_talea = playing_talea
221
222     if not isinstance(playing_groupings, collections.Sequence):
223         playing_groupings = (playing_groupings,)
224     playing_groupings = tuple(int(x) for x in playing_groupings)
225     assert len(playing_groupings)
226     assert all(0 < x for x in playing_groupings)
227     self._playing_groupings = playing_groupings
228
229     if reflect is not None:
230         reflect = bool(reflect)
231     self._reflect = reflect
232
233     self._repeat = bool(repeat)
234
235     if silence_talea is not None:
236         assert isinstance(silence_talea, rhythmmakertools.Talea)
237         assert silence_talea.counts

```

```

238         assert all(0 <= x for x in silence_talea.counts)
239         self._silence_talea = silence_talea
240
241     assert step_anchor in (Left, Right)
242     self._step_anchor = step_anchor
243     self._synchronize_groupings = bool(synchronize_groupings)
244     self._synchronize_step = bool(synchronize_step)
245
246     ### PRIVATE METHODS ###
247
248     def _make_infinite_iterator(self, sequence):
249         index = 0
250         sequence = datastructuretools.CyclicTuple(sequence)
251         while True:
252             yield sequence[index]
253             index += 1
254
255     def _make_timespans(
256         self,
257         layer=None,
258         music_specifiers=None,
259         target_timespan=None,
260         timespan_inventory=None,
261     ):
262         import consort
263         initial_silence_talea = self.initial_silence_talea
264         if not initial_silence_talea:
265             initial_silence_talea = rhythmmakertools.Talea((0,), 1)
266             initial_silence_talea = consort.Cursor(initial_silence_talea)
267             playing_talea = consort.Cursor(self.playing_talea)
268             playing_groupings = consort.Cursor(self.playing_groupings)
269             silence_talea = self.silence_talea
270             if silence_talea is None:
271                 silence_talea = rhythmmakertools.Talea((0,), 1)
272             silence_talea = consort.Cursor(silence_talea)
273
274         if self.seed is not None and 0 < self.seed:
275             for _ in range(self.seed):
276                 next(initial_silence_talea)
277                 next(playing_talea)
278                 next(playing_groupings)
279                 next(silence_talea)
280
281         if self.synchronize_step:
282             procedure = self._make_with_synchronized_step
283         else:
284             procedure = self._make_without_synchronized_step
285         new_timespan_inventory, final_offset = procedure(
286             initial_silence_talea=initial_silence_talea,
287             layer=layer,
288             playing_talea=playing_talea,
289             playing_groupings=playing_groupings,
290             music_specifiers=music_specifiers,
291             silence_talea=silence_talea,

```

```

292         target_timespan=target_timespan,
293     )
294     assert all(0 < _.duration for _ in new_timespan_inventory), \
295     (format(self), target_timespan)
296
297     if self.reflect:
298         new_timespan_inventory = new_timespan_inventory.reflect(
299             axis=target_timespan.axis,
300         )
301
302     return new_timespan_inventory
303
304 def _make_with_synchronized_step(
305     self,
306     initial_silence_talea=None,
307     layer=None,
308     playing_talea=None,
309     playing_groupings=None,
310     music_specifiers=None,
311     silence_talea=None,
312     target_timespan=None,
313 ):
314     import consort
315     counter = collections.Counter()
316     timespan_inventory = timespantools.TimespanInventory()
317     start_offset = target_timespan.start_offset
318     stop_offset = target_timespan.stop_offset
319     can_continue = True
320     while start_offset < stop_offset and can_continue:
321         silence_duration = next(silence_talea)
322         durations = []
323         if self.synchronize_groupings:
324             grouping = next(playing_groupings)
325             durations = [next(playing_talea) for _ in range(grouping)]
326         for context_name, music_specifier in music_specifiers.items():
327             if context_name not in counter:
328                 counter[context_name] = 0
329             seed = counter[context_name]
330             initial_silence_duration = next(initial_silence_talea)
331             if not self.synchronize_groupings:
332                 grouping = next(playing_groupings)
333                 durations = [next(playing_talea) for _ in range(grouping)]
334             maximum_offset = (
335                 start_offset +
336                 sum(durations) +
337                 silence_duration +
338                 initial_silence_duration
339             )
340             #if self.padding:
341             #    maximum_offset += (self.padding * 2)
342             maximum_offset = min(maximum_offset, stop_offset)
343             if self.step_anchor is Left:
344                 maximum_offset = min(
345                     maximum_offset,

```

```

346             (
347                 initial_silence_duration +
348                 start_offset +
349                 silence_duration
350             ),
351         )
352     current_offset = start_offset + initial_silence_duration
353     #if self.padding:
354     #    current_offset += self.padding
355     #    maximum_offset -= self.padding
356     group_offset = current_offset
357
358     valid_durations = []
359     for duration in durations:
360         if maximum_offset < (current_offset + duration):
361             can_continue = False
362             break
363         valid_durations.append(duration)
364     if self.fuse_groups:
365         valid_durations = [sum(valid_durations)]
366
367     new_timeespans = music_specifier(
368         durations=valid_durations,
369         layer=layer,
370         output_masks=self.output_masks,
371         padding=self.padding,
372         seed=seed,
373         start_offset=group_offset,
374         timespanSpecifier=self.timespanSpecifier,
375         voice_name=context_name,
376     )
377
378     if all(isinstance(_, consort.SilentTimespan)
379             for _ in new_timeespans):
380         new_timeespans[:] = []
381     timespan_inventory.extend(new_timeespans)
382     counter[context_name] += 1
383     timespan_inventory.sort()
384     if self.step_anchor == Right and timespan_inventory:
385         start_offset = timespan_inventory.stop_offset
386         start_offset += silence_duration
387         if not self.repeat:
388             break
389     return timespan_inventory, start_offset
390
391 def _make_without_synchronized_step(
392     self,
393     initial_silence_talea=None,
394     layer=None,
395     playing_talea=None,
396     playing_groupings=None,
397     music_specifiers=None,
398     silence_talea=None,
399     target_timespan=None,

```

```

400     ):
401     import consort
402     counter = collections.Counter()
403     timespan_inventory = timespantools.TimespanInventory()
404     start_offset = target_timespan.start_offset
405     stop_offset = target_timespan.stop_offset
406     final_offset = durationtools.Offset(0)
407     for context_name, music_specifier in music_specifiers.items():
408
409         if context_name not in counter:
410             counter[context_name] = 0
411
412         start_offset = target_timespan.start_offset
413         start_offset += next(initial_silence_talea)
414         can_continue = True
415
416         while start_offset < stop_offset and can_continue:
417
418             seed = counter[context_name]
419
420             silence_duration = next(silence_talea)
421             grouping = next(playing_groupings)
422             durations = [next(playing_talea) for _ in range(grouping)]
423             #if self.padding:
424             #    start_offset += self.padding
425
426             maximum_offset = start_offset + sum(durations) + \
427                 silence_duration
428             maximum_offset = min(maximum_offset, stop_offset)
429             if self.step_anchor is Left:
430                 maximum_offset = min(maximum_offset,
431                     start_offset + silence_duration)
432             #if self.padding:
433             #    maximum_offset -= self.padding
434
435             group_offset = current_offset = start_offset
436
437             valid_durations = []
438             for duration in durations:
439                 if maximum_offset < (current_offset + duration):
440                     can_continue = False
441                     break
442                 valid_durations.append(duration)
443                 current_offset += duration
444             if len(durations) != len(valid_durations):
445                 for _ in range(len(durations) - len(valid_durations)):
446                     playing_talea.backtrack()
447             if valid_durations and self.fuse_groups:
448                 valid_durations = [sum(valid_durations)]
449
450             new_timespans = music_specifier(
451                 durations=valid_durations,
452                 layer=layer,
453                 output_masks=self.output_masks,

```

```

454         padding=self.padding,
455         seed=seed,
456         start_offset=group_offset,
457         timespanSpecifier=self.timespanSpecifier,
458         voice_name=context_name,
459     )
460
461     if all(isinstance(_, consort.SilentTimespan)
462           for _ in new_timespans):
463         new_timespans = []
464         timespanInventory.extend(new_timespans)
465
466     if self.step_anchor is Left:
467         start_offset += silence_duration
468     else:
469         start_offset = current_offset + silence_duration
470
471     if stop_offset <= start_offset:
472         can_continue = False
473
474     if not can_continue:
475         if not valid_durations:
476             silenceTalea.backtrack()
477             silenceTalea.backtrack()
478             playingGroupings.backtrack()
479
480         if not self.repeat:
481             break
482         counter[context_name] += 1
483     if final_offset < start_offset:
484         final_offset = start_offset
485     return timespanInventory, final_offset
486
487     ### PUBLIC PROPERTIES ###
488
489     @property
490     def fuseGroups(self):
491         return self._fuseGroups
492
493     @property
494     def initialSilenceTalea(self):
495         return self._initialSilenceTalea
496
497     @property
498     def playingGroupings(self):
499         return self._playingGroupings
500
501     @property
502     def playingTalea(self):
503         return self._playingTalea
504
505     @property
506     def reflect(self):
507         return self._reflect

```

```

508
509     @property
510     def repeat(self):
511         return self._repeat
512
513     @property
514     def silence_talea(self):
515         return self._silence_talea
516
517     @property
518     def step_anchor(self):
519         return self._step_anchor
520
521     @property
522     def synchronize_groupings(self):
523         return self._synchronize_groupings
524
525     @property
526     def synchronize_step(self):
527         return self._synchronize_step

```

## A.53 CONSORTTOOLS.TEXTSPANNEREXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import abctools
3 from abjad import attach
4 from abjad.tools import datastructuretools
5 from abjad.tools import durationtools
6 from abjad.tools import indicatortools
7 from abjad.tools import markuptools
8 from abjad.tools import spannertools
9
10
11 class TextSpannerExpression(abctools.AbjadValueObject):
12
13     ### CLASS VARIABLES ###
14
15     __slots__ = (
16         '_markup_tokens',
17         '_transitions',
18     )
19
20     ### INITIALIZER ###
21
22     def __init__(
23         self,
24         markup_tokens=None,
25         transitions=None,
26     ):
27         if markup_tokens is not None:
28             coerced_markup_tokens = []
29             for x in markup_tokens:
30                 if isinstance(x, (markuptools.Markup, type(None))):
31                     markup = x

```

```

32         elif hasattr(x, 'markup'):
33             markup = x.markup
34         elif hasattr(x, '_get_markup'):
35             markup = x._get_markup()
36         else:
37             markup = markuptools.Markup(x)
38             coerced_markup_tokens.append(markup)
39         markup_tokens = datastructuretools.CyclicTuple(
40             coerced_markup_tokens)
41     self._markup_tokens = markup_tokens
42     if transitions:
43         prototype = (indicatortools.LineSegment, type(None))
44         assert len(transitions)
45         assert all(isinstance(_, prototype) for _ in transitions)
46         transitions = datastructuretools.CyclicTuple(transitions)
47     self._transitions = transitions
48
49     ### SPECIAL METHODS ###
50
51     def __call__(self, music, name=None, seed=0):
52         selections = self._get_selections(music)
53         if 1 < len(selections):
54             for selection in selections[:-1]:
55                 markup, transition = self._get_attachments(seed)
56                 # do stuff
57                 seed += 1
58             markup, transition = self._get_attachments(seed)
59             selection = selections[-1]
60             if selection.get_duration() <= durationtools.Duration(1, 8):
61                 # do stuff
62                 pass
63             else:
64                 # do stuff
65                 seed += 1
66                 markup, transition = self._get_attachments(seed)
67                 # do stuff
68             text_spinner = spannertools.TextSpinner()
69             attach(text_spinner, music, name=name)
70
71     ### PRIVATE METHODS ###
72
73     def _get_selections(self, music):
74         selections = []
75         for division in music:
76             selection = division.select_leaves()
77             selections.append(selection)
78         return selections
79
80     ### PUBLIC PROPERTIES ###
81
82     @property
83     def markup_tokens(self):
84         return self._markup_tokens
85

```

```

86     @property
87     def transitions(self):
88         return self._transitions
```

## A.54 CONSORTTOOLS.TIMESPANCOLLECTION

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3 from abjad.tools import timespantools
4
5
6 class TimespanCollection(abctools.AbjadObject):
7     r'''A mutable always-sorted collection of timespans.
8
9     :::
10
11     >>> import consort
12     >>> timespans = (
13     ...     timespantools.Timespan(0, 3),
14     ...     timespantools.Timespan(1, 3),
15     ...     timespantools.Timespan(1, 2),
16     ...     timespantools.Timespan(2, 5),
17     ...     timespantools.Timespan(6, 9),
18     ... )
19     >>> timespan_collection = consort.TimespanCollection(timespans)
20
21     ''
22
23     ### CLASS VARIABLES ###
24
25     __slots__ = (
26         '_root_node',
27     )
28
29     ### INITIALIZER ###
30
31     def __init__(
32         self,
33         timespans=None,
34     ):
35         self._root_node = None
36         if timespans is not None and timespans:
37             self.insert(timespans)
38
39     ### SPECIAL METHODS ###
40
41     def __contains__(self, timespan):
42         r'''Is true if this timespan collection contains 'timespan'. Otherwise
43         false.
44
45         :::
46
47         >>> timespans = (
48             ...     timespantools.Timespan(0, 3),
```

```

49         ...      timespantools.Timespan(1, 3),
50         ...      timespantools.Timespan(1, 2),
51         ...      timespantools.Timespan(2, 5),
52         ...      timespantools.Timespan(6, 9),
53         ...
54     >>> timespan_collection = consort.TimespanCollection(timespans)
55
56     :::
57
58     >>> timespans[0] in timespan_collection
59     True
60
61     :::
62
63     >>> timespantools.Timespan(-1, 100) in timespan_collection
64     False
65
66     Returns boolean.
67     ''
68     assert TimespanCollection._is_timespan(timespan)
69     candidates = self.find_timespans_starting_at(timespan.start_offset)
70     result = timespan in candidates
71     return result
72
73 def __getitem__(self, i):
74     '''Gets timespan at index 'i'.
75
76     :::
77
78     >>> timespans = (
79         ...      timespantools.Timespan(0, 3),
80         ...      timespantools.Timespan(1, 3),
81         ...      timespantools.Timespan(1, 2),
82         ...      timespantools.Timespan(2, 5),
83         ...      timespantools.Timespan(6, 9),
84         ...
85     >>> timespan_collection = consort.TimespanCollection(timespans)
86
87     :::
88
89     >>> timespan_collection[-1]
90     Timespan(start_offset=Offset(6, 1), stop_offset=Offset(9, 1))
91
92     :::
93
94     >>> for timespan in timespan_collection[:3]:
95         ...      timespan
96         ...
97         Timespan(start_offset=Offset(0, 1), stop_offset=Offset(3, 1))
98         Timespan(start_offset=Offset(1, 1), stop_offset=Offset(2, 1))
99         Timespan(start_offset=Offset(1, 1), stop_offset=Offset(3, 1))
100
101    Returns timespan or timespans.
102    '''

```

```

103     def recurse_by_index(node, index):
104         if node.node_start_index <= index < node.node_stop_index:
105             return node.payload[index - node.node_start_index]
106         elif node.left_child and index < node.node_start_index:
107             return recurse_by_index(node.left_child, index)
108         elif node.right_child and node.node_stop_index <= index:
109             return recurse_by_index(node.right_child, index)
110
111     def recurse_by_slice(node, start, stop):
112         result = []
113         if node is None:
114             return result
115         if start < node.node_start_index and node.left_child:
116             result.extend(recurse_by_slice(node.left_child, start, stop))
117         if start < node.node_stop_index and node.node_start_index < stop:
118             node_start = start - node.node_start_index
119             if node_start < 0:
120                 node_start = 0
121             node_stop = stop - node.node_start_index
122             result.extend(node.payload[node_start:node_stop])
123         if node.node_stop_index <= stop and node.right_child:
124             result.extend(recurse_by_slice(node.right_child, start, stop))
125         return result
126
127     if isinstance(i, int):
128         if self._root_node is None:
129             raise IndexError
130         if i < 0:
131             i = self._root_node.subtree_stop_index + i
132         if i < 0 or self._root_node.subtree_stop_index <= i:
133             raise IndexError
134         return recurse_by_index(self._root_node, i)
135     elif isinstance(i, slice):
136         if self._root_node is None:
137             return []
138         indices = i.indices(self._root_node.subtree_stop_index)
139         start, stop = indices[0], indices[1]
140         return recurse_by_slice(self._root_node, start, stop)
141
142     raise TypeError('Indices must be integers or slices, got {}'.format(i))
143
144     def __iter__(self):
145         r'''Iterates timespans in this timespan collection.
146
147         ::

148             >>> timespans = (
149             ...     timespantools.Timespan(0, 3),
150             ...     timespantools.Timespan(1, 3),
151             ...     timespantools.Timespan(1, 2),
152             ...     timespantools.Timespan(2, 5),
153             ...     timespantools.Timespan(6, 9),
154             ...
155             )
156             >>> timespan_collection = consort.TimespanCollection(timespans)

```

```

157
158      ::

159
160      >>> for timespan in timespan_collection:
161          ...     timespan
162          ...
163          Timespan(start_offset=Offset(0, 1), stop_offset=Offset(3, 1))
164          Timespan(start_offset=Offset(1, 1), stop_offset=Offset(2, 1))
165          Timespan(start_offset=Offset(1, 1), stop_offset=Offset(3, 1))
166          Timespan(start_offset=Offset(2, 1), stop_offset=Offset(5, 1))
167          Timespan(start_offset=Offset(6, 1), stop_offset=Offset(9, 1))

168
169      Returns generator.
170      ''

171
172  def recurse(node):
173      if node is not None:
174          if node.left_child is not None:
175              for timespan in recurse(node.left_child):
176                  yield timespan
177          for timespan in node.payload:
178              yield timespan
179          if node.right_child is not None:
180              for timespan in recurse(node.right_child):
181                  yield timespan
182      return recurse(self._root_node)

183
184  def __len__():
185      '''Gets length of this timespan collection.
186
187      ::

188
189      >>> timespans = (
190          ...     timespantools.Timespan(0, 3),
191          ...     timespantools.Timespan(1, 3),
192          ...     timespantools.Timespan(1, 2),
193          ...     timespantools.Timespan(2, 5),
194          ...     timespantools.Timespan(6, 9),
195          ...
196      )
197      >>> timespan_collection = consort.TimespanCollection(timespans)
198
199      ::

200      >>> len(timespan_collection)
201      5

202
203      Returns integer.
204      ''
205
206      if self._root_node is None:
207          return 0
208      return self._root_node.subtree_stop_index

209  def __setitem__(self, i, new):
210      '''Sets timespans at index 'i' to 'new'.

```

```

211
212      ::

213
214      >>> timespans = (
215          ...     timespantools.Timespan(0, 3),
216          ...     timespantools.Timespan(1, 3),
217          ...     timespantools.Timespan(1, 2),
218          ...     timespantools.Timespan(2, 5),
219          ...     timespantools.Timespan(6, 9),
220          ...
221      )
222      >>> timespan_collection = consort.TimespanCollection(timespans)

223      ::

224
225      >>> timespan_collection[:3] = [timespantools.Timespan(100, 200)]
226

227      Returns none.
228      ''
229      if isinstance(i, (int, slice)):
230          old = self[i]
231          self.remove(old)
232          self.insert(new)
233      else:
234          message = 'Indices must be ints or slices, got {}'.format(i)
235          raise TypeError(message)
236
237      def __sub__(self, timespan):
238          r'''Delete material that intersects 'timespan'::

239
240          ::

241
242          >>> timespan_collection = consort.TimespanCollection([
243              ...     timespantools.Timespan(0, 16),
244              ...     timespantools.Timespan(5, 12),
245              ...     timespantools.Timespan(-2, 8),
246              ... ])
247
248          ::

249
250          >>> timespan = timespantools.Timespan(5, 10)
251          >>> result = timespan_collection - timespan
252
253          ::

254
255          >>> print(format(timespan_collection))
256          consort.tools.TimespanCollection(
257              [
258                  timespantools.Timespan(
259                      start_offset=durationtools.Offset(-2, 1),
260                      stop_offset=durationtools.Offset(5, 1),
261                      ),
262                  timespantools.Timespan(
263                      start_offset=durationtools.Offset(0, 1),
264                      stop_offset=durationtools.Offset(5, 1),

```

```

265             ),
266             timespantools.Timespan(
267                 start_offset=durationtools.Offset(10, 1),
268                 stop_offset=durationtools.Offset(12, 1),
269             ),
270             timespantools.Timespan(
271                 start_offset=durationtools.Offset(10, 1),
272                 stop_offset=durationtools.Offset(16, 1),
273             ),
274         ],
275     )
276
277     Operates in place and returns timespan collection.
278     """
279     intersecting_timespans = self.find_timespans_intersecting_timespan(
280         timespan)
281     self.remove(intersecting_timespans)
282     for intersecting_timespan in intersecting_timespans:
283         for x in (intersecting_timespan - timespan):
284             self.insert(x)
285     return self
286
287     ### PRIVATE METHODS ###
288
289     def _insert_node(self, node, start_offset):
290         import consort
291         if node is None:
292             return consort.TimespanCollectionNode(start_offset)
293         if start_offset < node.start_offset:
294             node.left_child = self._insert_node(node.left_child, start_offset)
295         elif node.start_offset < start_offset:
296             node.right_child = self._insert_node(node.right_child, start_offset)
297         return self._rebalance(node)
298
299     def _insert_timespan(self, timespan):
300         self._root_node = self._insert_node(
301             self._root_node,
302             timespan.start_offset,
303         )
304         node = self._search(self._root_node, timespan.start_offset)
305         node.payload.append(timespan)
306         node.payload.sort(key=lambda x: x.stop_offset)
307
308     @staticmethod
309     def _is_timespan(expr):
310         if hasattr(expr, 'start_offset') and hasattr(expr, 'stop_offset'):
311             return True
312         return False
313
314     def _rebalance(self, node):
315         if node is not None:
316             if 1 < node.balance:
317                 if 0 <= node.right_child.balance:
318                     node = self._rotate_right_right(node)

```

```

319         else:
320             node = self._rotate_right_left(node)
321     elif node.balance < -1:
322         if node.left_child.balance <= 0:
323             node = self._rotate_left_left(node)
324         else:
325             node = self._rotate_left_right(node)
326     assert -1 <= node.balance <= 1
327     return node
328
329 def _remove_node(self, node, start_offset):
330     if node is not None:
331         if node.start_offset == start_offset:
332             if node.left_child and node.right_child:
333                 next_node = node.right_child
334                 while next_node.left_child:
335                     next_node = next_node.left_child
336                 node._start_offset = next_node._start_offset
337                 node._payload = next_node._payload
338                 node.right_child = self._remove_node(
339                     node.right_child,
340                     next_node.start_offset,
341                     )
342             else:
343                 node = node.left_child or node.right_child
344             elif start_offset < node.start_offset:
345                 node.left_child = self._remove_node(
346                     node.left_child,
347                     start_offset,
348                     )
349             elif node.start_offset < start_offset:
350                 node.right_child = self._remove_node(
351                     node.right_child,
352                     start_offset,
353                     )
354     return self._rebalance(node)
355
356 def _remove_timespan(self, timespan, old_start_offset=None):
357     start_offset = timespan.start_offset
358     if old_start_offset is not None:
359         start_offset = old_start_offset
360     node = self._search(self._root_node, start_offset)
361     if node is None:
362         return
363     if timespan in node.payload:
364         node.payload.remove(timespan)
365     if not node.payload:
366         self._root_node = self._remove_node(
367             self._root_node,
368             start_offset,
369             )
370     if isinstance(timespan, TimespanCollection):
371         timespan._parents.remove(self)
372

```

```

373     def _rotate_left_left(self, node):
374         next_node = node.left_child
375         node.left_child = next_node.right_child
376         next_node.right_child = node
377         return next_node
378
379     def _rotate_left_right(self, node):
380         node.left_child = self._rotate_right_right(node.left_child)
381         next_node = self._rotate_left_left(node)
382         return next_node
383
384     def _rotate_right_left(self, node):
385         node.right_child = self._rotate_left_left(node.right_child)
386         next_node = self._rotate_right_right(node)
387         return next_node
388
389     def _rotate_right_right(self, node):
390         next_node = node.right_child
391         node.right_child = next_node.left_child
392         next_node.left_child = node
393         return next_node
394
395     def _search(self, node, start_offset):
396         if node is not None:
397             if node.start_offset == start_offset:
398                 return node
399             elif node.left_child and start_offset < node.start_offset:
400                 return self._search(node.left_child, start_offset)
401             elif node.right_child and node.start_offset < start_offset:
402                 return self._search(node.right_child, start_offset)
403         return None
404
405     def _update_indices(
406         self,
407         node,
408     ):
409         def recurse(
410             node,
411             parent_stop_index=None,
412         ):
413             if node is None:
414                 return
415             if node.left_child is not None:
416                 recurse(
417                     node.left_child,
418                     parent_stop_index=parent_stop_index,
419                 )
420                 node._node_start_index = node.left_child.subtree_stop_index
421                 node._subtree_start_index = node.left_child.subtree_start_index
422             elif parent_stop_index is None:
423                 node._node_start_index = 0
424                 node._subtree_start_index = 0
425             else:
426                 node._node_start_index = parent_stop_index

```

```

427     node._subtree_start_index = parent_stop_index
428     node._node_stop_index = node.node_start_index + len(node.payload)
429     node._subtree_stop_index = node.node_stop_index
430     if node.right_child is not None:
431         recurse(
432             node.right_child,
433             parent_stop_index=node.node_stop_index,
434             )
435         node._subtree_stop_index = node.right_child.subtree_stop_index
436     recurse(node)
437
438 def _update_offsets(
439     self,
440     node,
441     ):
442     if node is None:
443         return
444     stop_offset_low = min(x.stop_offset for x in node.payload)
445     stop_offset_high = max(x.stop_offset for x in node.payload)
446     if node.left_child:
447         left_child = self._update_offsets(
448             node.left_child,
449             )
450         if left_child.stop_offset_low < stop_offset_low:
451             stop_offset_low = left_child.stop_offset_low
452         if stop_offset_high < left_child.stop_offset_high:
453             stop_offset_high = left_child.stop_offset_high
454     if node.right_child:
455         right_child = self._update_offsets(
456             node.right_child,
457             )
458         if right_child.stop_offset_low < stop_offset_low:
459             stop_offset_low = right_child.stop_offset_low
460         if stop_offset_high < right_child.stop_offset_high:
461             stop_offset_high = right_child.stop_offset_high
462     node._stop_offset_low = stop_offset_low
463     node._stop_offset_high = stop_offset_high
464     return node
465
466 ### PRIVATE PROPERTIES ###
467
468 @property
469 def _storage_format_specification(self):
470     from abjad.tools import systemtools
471     positional_argument_values = ()
472     timespans = [x for x in self]
473     if timespans:
474         positional_argument_values = (timespans,)
475     keyword_argument_names = ()
476     return systemtools.StorageFormatSpecification(
477         self,
478         keyword_argument_names=keyword_argument_names,
479         positional_argument_values=positional_argument_values,
480         )

```

```

481
482     """ PUBLIC METHODS """
483
484     def find_timeperiods_starting_at(self, offset):
485         results = []
486         node = self._search(self._root_node, offset)
487         if node is not None:
488             results.extend(node.payload)
489         return tuple(results)
490
491     def find_timeperiods_stopping_at(self, offset):
492         def recurse(node, offset):
493             result = []
494             if node is not None:
495                 if node.stop_offset_low <= offset <= node.stop_offset_high:
496                     for timeperiod in node.payload:
497                         if timeperiod.stop_offset == offset:
498                             result.append(timeperiod)
499                         if node.left_child is not None:
500                             result.extend(recurse(node.left_child, offset))
501                         if node.right_child is not None:
502                             result.extend(recurse(node.right_child, offset))
503             return result
504         results = recurse(self._root_node, offset)
505         results.sort(key=lambda x: (x.start_offset, x.stop_offset))
506         return tuple(results)
507
508     def find_timeperiods_overlapping_offset(self, offset):
509         """Finds timeperiods overlapping 'offset'.
510
511         :::
512
513         >>> timeperiods = (
514             ...     timeperiods.Timeperiod(0, 3),
515             ...     timeperiods.Timeperiod(1, 3),
516             ...     timeperiods.Timeperiod(1, 2),
517             ...     timeperiods.Timeperiod(2, 5),
518             ...     timeperiods.Timeperiod(6, 9),
519             ...
520         )
521         >>> timeperiod_collection = consort.TimeperiodCollection(timeperiods)
522
523         :::
524
525         >>> for x in timeperiod_collection.find_timeperiods_overlapping_offset(1.5):
526             ...
527             x
528             ...
529             Timeperiod(start_offset=Offset(0, 1), stop_offset=Offset(3, 1))
530             Timeperiod(start_offset=Offset(1, 1), stop_offset=Offset(2, 1))
531             Timeperiod(start_offset=Offset(1, 1), stop_offset=Offset(3, 1))
532
533             Returns tuple of 0 or more timeperiods.
534             """
535
536         def recurse(node, offset, indent=0):
537             result = []

```

```

535     if node is not None:
536         if node.start_offset < offset < node.stop_offset_high:
537             result.extend(recurse(node.left_child, offset, indent + 1))
538             for timespan in node.payload:
539                 if offset < timespan.stop_offset:
540                     result.append(timespan)
541             result.extend(recurse(node.right_child, offset, indent + 1))
542         elif offset <= node.start_offset:
543             result.extend(recurse(node.left_child, offset, indent + 1))
544     return result
545 results = recurse(self._root_node, offset)
546 results.sort(key=lambda x: (x.start_offset, x.stop_offset))
547 return tuple(results)
548
549 def find_timeperiods_intersecting_timeperiod(self, timeperiod):
550     r'''Finds timeperiods overlapping 'timeperiod'.
551
552     :::
553
554     >>> timeperiods = (
555         ...     timeperiodtools.Timeperiod(0, 3),
556         ...     timeperiodtools.Timeperiod(1, 3),
557         ...     timeperiodtools.Timeperiod(1, 2),
558         ...     timeperiodtools.Timeperiod(2, 5),
559         ...     timeperiodtools.Timeperiod(6, 9),
560         ...
561     )
562     >>> timeperiod_collection = consort.TimeperiodCollection(timeperiods)
563
564     :::
565
566     >>> timeperiod = timeperiodtools.Timeperiod(2, 4)
567     >>> for x in timeperiod_collection.find_timeperiods_intersecting_timeperiod(timeperiod):
568         ...
569         ...
570         Timeperiod(start_offset=Offset(0, 1), stop_offset=Offset(3, 1))
571         Timeperiod(start_offset=Offset(1, 1), stop_offset=Offset(3, 1))
572         Timeperiod(start_offset=Offset(2, 1), stop_offset=Offset(5, 1))
573
574     Returns tuple of 0 or more timeperiods.
575     '''
576
577     def recurse(node, timeperiod):
578         result = []
579         if node is not None:
580             if timeperiod.intersects_timeperiod(node):
581                 result.extend(recurse(node.left_child, timeperiod))
582                 for candidate_timeperiod in node.payload:
583                     if candidate_timeperiod.intersects_timeperiod(timeperiod):
584                         result.append(candidate_timeperiod)
585                 result.extend(recurse(node.right_child, timeperiod))
586             elif (timeperiod.start_offset <= node.start_offset) or \
587                  (timeperiod.stop_offset <= node.start_offset):
588                 result.extend(recurse(node.left_child, timeperiod))
589         return result
590
591 results = recurse(self._root_node, timeperiod)

```

```

589     results.sort(key=lambda x: (x.start_offset, x.stop_offset))
590     return tuple(results)
591
592     def get_simultaneity_at(self, offset):
593         r'''Gets simultaneity at 'offset'.
594
595         :::
596
597         >>> timespans = (
598             ...     timespantools.Timespan(0, 3),
599             ...     timespantools.Timespan(1, 3),
600             ...     timespantools.Timespan(1, 2),
601             ...     timespantools.Timespan(2, 5),
602             ...     timespantools.Timespan(6, 9),
603             ...
604         )
605         >>> timespan_collection = consort.TimespanCollection(timespans)
606
607         :::
608
609         >>> timespan_collection.get_simultaneity_at(1)
610         <TimespanSimultaneity(1 <<3>>)>
611
612         :::
613
614         >>> timespan_collection.get_simultaneity_at(6.5)
615         <TimespanSimultaneity(6.5 <<1>>)>
616
617         '''
618
619         import consort
620         start_timespans = self.find_timespans_starting_at(offset)
621         stop_timespans = self.find_timespans_stopping_at(offset)
622         overlap_timespans = self.find_timespans_overlapping_offset(offset)
623         simultaneity = consort.TimespanSimultaneity(
624             timespan_collection=self,
625             overlap_timespans=overlap_timespans,
626             start_timespans=start_timespans,
627             start_offset=offset,
628             stop_timespans=stop_timespans,
629             )
630
631         return simultaneity
632
633     def get_start_offset_after(self, offset):
634         r'''Gets start offset in this timespan collection after 'offset'.
635
636         :::
637
638         >>> timespans = (
639             ...     timespantools.Timespan(0, 3),
640             ...     timespantools.Timespan(1, 3),
641             ...     timespantools.Timespan(1, 2),
642             ...     timespantools.Timespan(2, 5),
643             ...     timespantools.Timespan(6, 9),
644             ...
645         )
646         >>> timespan_collection = consort.TimespanCollection(timespans)

```

```

643
644      ::

645
646      >>> timespan_collection.get_start_offset_after(-1)
647      Offset(0, 1)

648
649      ::

650
651      >>> timespan_collection.get_start_offset_after(0)
652      Offset(1, 1)

653
654      ::

655
656      >>> timespan_collection.get_start_offset_after(1)
657      Offset(2, 1)

658
659      ::

660
661      >>> timespan_collection.get_start_offset_after(2)
662      Offset(6, 1)

663
664      ::

665
666      >>> timespan_collection.get_start_offset_after(6) is None
667      True

668
669      ''
670
671      def recurse(node, offset):
672          if node is None:
673              return None
674          result = None
675          if node.start_offset <= offset and node.right_child:
676              result = recurse(node.right_child, offset)
677          elif offset < node.start_offset:
678              result = recurse(node.left_child, offset) or node
679          return result
680
681      result = recurse(self._root_node, offset)
682      if result is None:
683          return None
684      return result.start_offset

685
686
687      def get_start_offset_before(self, offset):
688          r'''Gets start offset in this timespan collection before 'offset'.
689
690          ::

691          >>> timespans = (
692              ...     timespantools.Timespan(0, 3),
693              ...     timespantools.Timespan(1, 3),
694              ...     timespantools.Timespan(1, 2),
695              ...     timespantools.Timespan(2, 5),
696              ...     timespantools.Timespan(6, 9),
697              ...
698          )
699          >>> timespan_collection = consort.TimespanCollection(timespans)

```

```

697
698      ::

699      >>> timespan_collection.get_start_offset_before(7)
700      Offset(6, 1)

702
703      ::

704      >>> timespan_collection.get_start_offset_before(6)
705      Offset(2, 1)

707
708      ::

709      >>> timespan_collection.get_start_offset_before(2)
710      Offset(1, 1)

712
713      ::

714      >>> timespan_collection.get_start_offset_before(1)
715      Offset(0, 1)

717
718      ::

719      >>> timespan_collection.get_start_offset_before(0) is None
720      True

722
723      ...
724      def recurse(node, offset):
725          if node is None:
726              return None
727          result = None
728          if node.start_offset < offset:
729              result = recurse(node.right_child, offset) or node
730          elif offset <= node.start_offset and node.left_child:
731              result = recurse(node.left_child, offset)
732          return result
733          result = recurse(self._root_node, offset)
734          if result is None:
735              return None
736          return result.start_offset

737
738      def index(self, timespan):
739          assert self._is_timespan(timespan)
740          node = self._search(self._root_node, timespan.start_offset)
741          if node is None or timespan not in node.payload:
742              raise ValueError('{} not in timespan collection.'.format(timespan))
743          index = node.payload.index(timespan) + node.node_start_index
744          return index

745
746      def insert(self, timespans):
747          r'''Inserts 'timespans' into this timespan collection.

748
749          ::

750

```

```

751     >>> timespan_collection = consort.TimespanCollection()
752     >>> timespan_collection.insert(timespantools.Timespan(1, 3))
753     >>> timespan_collection.insert((
754         ...     timespantools.Timespan(0, 4),
755         ...     timespantools.Timespan(2, 6),
756         ... ))
757
758     :::
759
760     >>> for x in timespan_collection:
761         ...
762         ...
763         Timespan(start_offset=Offset(0, 1), stop_offset=Offset(4, 1))
764         Timespan(start_offset=Offset(1, 1), stop_offset=Offset(3, 1))
765         Timespan(start_offset=Offset(2, 1), stop_offset=Offset(6, 1))
766
767     'timespans' may be a single timespan or an iterable of timespans.
768
769     Returns None.
770     ''
771     if self._is_timespan(timespans):
772         timespans = [timespans]
773     for timespan in timespans:
774         if not self._is_timespan(timespan):
775             continue
776         self._insert_timespan(timespan)
777     self._update_indices(self._root_node)
778     self._update_offsets(self._root_node)
779
780     def iterate_simultaneities(
781         self,
782         reverse=False,
783     ):
784         r"""Iterates simultaneities in this timespan collection.
785
786         :::
787
788         >>> timespans = (
789             ...     timespantools.Timespan(0, 3),
790             ...     timespantools.Timespan(1, 3),
791             ...     timespantools.Timespan(1, 2),
792             ...     timespantools.Timespan(2, 5),
793             ...     timespantools.Timespan(6, 9),
794             ...
795         )
796         >>> timespan_collection = consort.TimespanCollection(timespans)
797
798         :::
799
800         >>> for x in timespan_collection.iterate_simultaneities():
801             ...
802             ...
803             <TimespanSimultaneity(0 <<1>>)|
804             <TimespanSimultaneity(1 <<3>>)|
805             <TimespanSimultaneity(2 <<3>>)|

```

```

805             <TimespanSimultaneity(6 <<1>>)(>
806
807     :::
808
809     >>> for x in timespan_collection.iterate_simultaneities(
810         ...      reverse=True):
811         ...
812         ...
813         <TimespanSimultaneity(6 <<1>>)(>
814         <TimespanSimultaneity(2 <<3>>)(>
815         <TimespanSimultaneity(1 <<3>>)(>
816         <TimespanSimultaneity(0 <<1>>)(>
817
818     Returns generator.
819     """
820
821
822     if reverse:
823         start_offset = self.latest_start_offset
824         simultaneity = self.get_simultaneity_at(start_offset)
825         yield simultaneity
826         simultaneity = simultaneity.previous_simultaneity
827         while simultaneity is not None:
828             yield simultaneity
829             simultaneity = simultaneity.previous_simultaneity
830     else:
831         start_offset = self.earliest_start_offset
832         simultaneity = self.get_simultaneity_at(start_offset)
833         yield simultaneity
834         simultaneity = simultaneity.next_simultaneity
835         while simultaneity is not None:
836             yield simultaneity
837             simultaneity = simultaneity.next_simultaneity
838
839     def iterate_simultaneities_nwise(
840         self,
841         n=3,
842         reverse=False,
843     ):
844         r'''Iterates simultaneities in this timespan collection in groups of
845         'n'.
846
847         :::
848
849         >>> timespans = (
850             ...      timespantools.Timespan(0, 3),
851             ...      timespantools.Timespan(1, 3),
852             ...      timespantools.Timespan(1, 2),
853             ...      timespantools.Timespan(2, 5),
854             ...      timespantools.Timespan(6, 9),
855             ...
856         )
857         >>> timespan_collection = consort.TimespanCollection(timespans)
858
859     :::

```

```

859     >>> for x in timespan_collection.iterate_simultaneities_nwise(n=2):
860         ...
861         ...
862         (<TimespanSimultaneity(0 <<1>>), <TimespanSimultaneity(1 <<3>>)>
863         (<TimespanSimultaneity(1 <<3>>), <TimespanSimultaneity(2 <<3>>)>
864         (<TimespanSimultaneity(2 <<3>>), <TimespanSimultaneity(6 <<1>>)>
865
866     :::
867
868     >>> for x in timespan_collection.iterate_simultaneities_nwise(
869         ...      n=2, reverse=True):
870         ...
871         ...
872         (<TimespanSimultaneity(2 <<3>>), <TimespanSimultaneity(6 <<1>>)>
873         (<TimespanSimultaneity(1 <<3>>), <TimespanSimultaneity(2 <<3>>)>
874         (<TimespanSimultaneity(0 <<1>>), <TimespanSimultaneity(1 <<3>>)>
875
876     Returns generator.
877     ...
878     n = int(n)
879     assert 0 < n
880     if reverse:
881         for simultaneity in self.iterate_simultaneities(reverse=True):
882             simultaneities = [simultaneity]
883             while len(simultaneities) < n:
884                 next_simultaneity = simultaneities[-1].next_simultaneity
885                 if next_simultaneity is None:
886                     break
887                 simultaneities.append(next_simultaneity)
888             if len(simultaneities) == n:
889                 yield tuple(simultaneities)
890     else:
891         for simultaneity in self.iterate_simultaneities():
892             simultaneities = [simultaneity]
893             while len(simultaneities) < n:
894                 previous_simultaneity = simultaneities[-1].previous_simultaneity
895                 if previous_simultaneity is None:
896                     break
897                 simultaneities.append(previous_simultaneity)
898             if len(simultaneities) == n:
899                 yield tuple(reversed(simultaneities))
900
901     def remove(self, timespans):
902         r'''Removes timespans from this timespan collection.
903
904     :::
905
906     >>> timespans = (
907         ...      timespantools.Timespan(0, 3),
908         ...      timespantools.Timespan(1, 3),
909         ...      timespantools.Timespan(1, 2),
910         ...      timespantools.Timespan(2, 5),
911         ...      timespantools.Timespan(6, 9),
912         ...
913

```

```

913         >>> timespan_collection = consort.TimespanCollection(timespans)
914
915         :::::
916
917         >>> timespan_collection.remove(timespans[1:-1])
918
919         :::::
920
921         >>> for timespan in timespan_collection:
922             ...     timespan
923             ...
924             Timespan(start_offset=Offset(0, 1), stop_offset=Offset(3, 1))
925             Timespan(start_offset=Offset(6, 1), stop_offset=Offset(9, 1))
926
927             ...
928         if self._is_timespan(timespans):
929             timespans = [timespans]
930         for timespan in timespans:
931             if not self._is_timespan(timespan):
932                 continue
933             self._remove_timespan(timespan)
934             self._update_indices(self._root_node)
935             self._update_offsets(self._root_node)
936
937     ### PUBLIC PROPERTIES ###
938
939     @property
940     def all_offsets(self):
941         offsets = set()
942         for timespan in self:
943             offsets.add(timespan.start_offset)
944             offsets.add(timespan.stop_offset)
945         return tuple(sorted(offsets))
946
947     @property
948     def all_start_offsets(self):
949         start_offsets = set()
950         for timespan in self:
951             start_offsets.add(timespan.start_offset)
952         return tuple(sorted(start_offsets))
953
954     @property
955     def all_stop_offsets(self):
956         stop_offsets = set()
957         for timespan in self:
958             stop_offsets.add(timespan.stop_offset)
959         return tuple(sorted(stop_offsets))
960
961     @property
962     def earliest_start_offset(self):
963         def recurse(node):
964             if node.left_child is not None:
965                 return recurse(node.left_child)
966             return node.start_offset

```

```

967     if self._root_node is not None:
968         return recurse(self._root_node)
969     return float('-inf')
970
971     @property
972     def earliest_stop_offset(self):
973         if self._root_node is not None:
974             return self._root_node.stop_offset_low
975         return float('inf')
976
977     @property
978     def latest_start_offset(self):
979         def recurse(node):
980             if node.right_child is not None:
981                 return recurse(node._right_child)
982             return node.start_offset
983         if self._root_node is not None:
984             return recurse(self._root_node)
985         return float('-inf')
986
987     @property
988     def latest_stop_offset(self):
989         if self._root_node is not None:
990             return self._root_node.stop_offset_high
991         return float('inf')
992
993     @property
994     def start_offset(self):
995         return self.earliest_start_offset
996
997     @property
998     def stop_offset(self):
999         return self.latest_stop_offset

```

## A.55 CONSORTTOOLS.TIMESPANCOLLECTIONNODE

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3 from abjad.tools import timespantools
4
5
6 class TimespanCollectionNode(abctools.AbjadObject):
7     r'''A node in a timespan collection.
8     '''
9
10    ### CLASS VARIABLES ###
11
12    __slots__ = (
13        '_balance',
14        '_height',
15        '_left_child',
16        '_node_start_index',
17        '_node_stop_index',
18        '_payload',

```

```

19     '_right_child',
20     '_start_offset',
21     '_stop_offset_high',
22     '_stop_offset_low',
23     '_subtree_start_index',
24     '_subtree_stop_index',
25 )
26
27     ### INITIALIZER ###
28
29 def __init__(self, start_offset=0):
30     self._balance = 0
31     self._height = 0
32     self._left_child = None
33     self._node_start_index = -1
34     self._node_stop_index = -1
35     self._payload = []
36     self._right_child = None
37     self._start_offset = start_offset
38     self._stop_offset_high = None
39     self._stop_offset_low = None
40     self._subtree_start_index = -1
41     self._subtree_stop_index = -1
42
43     ### SPECIAL METHODS ###
44
45 def __repr__(self):
46     r'''Gets the repr of this timespan collection node.
47     '''
48     return '<Node: Start:{} Indices:{}:{}:{}:{} Length:{}>'.format(
49         self.start_offset,
50         self.subtree_start_index,
51         self.node_start_index,
52         self.node_stop_index,
53         self.subtree_stop_index,
54         len(self.payload),
55     )
56
57     ### PRIVATE METHODS ###
58
59 def _debug(self):
60     return '\n'.join(self._get_debug_pieces())
61
62 def _get_debug_pieces(self):
63     result = []
64     result.append(repr(self))
65     if self.left_child:
66         subresult = self.left_child._get_debug_pieces()
67         result.append('    L: {}'.format(subresult[0]))
68         result.extend('        ' + x for x in subresult[1:])
69     if self.right_child:
70         subresult = self.right_child._get_debug_pieces()
71         result.append('    R: {}'.format(subresult[0]))
72         result.extend('        ' + x for x in subresult[1:])

```

```

73     return result
74
75     def _update(self):
76         left_height = -1
77         right_height = -1
78         if self.left_child is not None:
79             left_height = self.left_child.height
80         if self.right_child is not None:
81             right_height = self.right_child.height
82         self._height = max(left_height, right_height) + 1
83         self._balance = right_height - left_height
84         return self.height
85
86     ### PUBLIC PROPERTIES ###
87
88     @property
89     def balance(self):
90         r'''Gets the balance of this timespan collection node.
91         '''
92         return self._balance
93
94     @property
95     def height(self):
96         r'''Gets the height of this timespan collection node.
97         '''
98         return self._height
99
100    @property
101    def left_child(self):
102        r'''Gets and sets the left child of this timespan collection node.
103        '''
104        return self._left_child
105
106    @left_child.setter
107    def left_child(self, node):
108        self._left_child = node
109        self._update()
110
111    @property
112    def node_start_index(self):
113        r'''Gets the node start index of this timespan collection node.
114        '''
115        return self._node_start_index
116
117    @property
118    def node_stop_index(self):
119        r'''Gets the node stop index of this timespan collection node.
120        '''
121        return self._node_stop_index
122
123    @property
124    def payload(self):
125        r'''Gets the payload of this timespan collection node.
126        '''

```

```

127     return self._payload
128
129     @property
130     def right_child(self):
131         '''Gets and sets the right child of this timespan collection node.
132         '''
133         return self._right_child
134
135     @right_child.setter
136     def right_child(self, node):
137         self._right_child = node
138         self._update()
139
140     @property
141     def start_offset(self):
142         '''Gets the start offset of this timespan collection node.
143         '''
144         return self._start_offset
145
146     @property
147     def stop_offset_high(self):
148         '''Gets the highest stop offset of the subtree rooted on this timespan
149         collection node.
150         '''
151         return self._stop_offset_high
152
153     @property
154     def stop_offset_low(self):
155         '''Gets the lowest stop offset of the subtree rooted on this timespan
156         collection node.
157         '''
158         return self._stop_offset_low
159
160     @property
161     def subtree_start_index(self):
162         '''Gets the start index of the subtree rooted on this timespan
163         collection node.
164         '''
165         return self._subtree_start_index
166
167     @property
168     def subtree_stop_index(self):
169         '''Gets the stop index of the subtree rooted on this timespan
170         collection node.
171         '''
172         return self._subtree_stop_index
173
174     @property
175     def timespan(self):
176         return timespantools.Timespan(
177             start_offset=self.start_offset,
178             stop_offset=self.stop_offset_high,
179         )

```

## A.56 CONSORTTOOLS.TIMESPANINVENTORYMAPPING

```
1 # -*- encoding: utf-8 -*-
2 from abjad.tools import timespanools
3
4
5 class TimespanInventoryMapping(dict):
6
7     ### SPECIAL METHODS ###
8
9     def __illustrate__(self, range_=None, scale=None):
10         timespan_inventory = timespanools.TimespanInventory()
11         for key, value in self.items():
12             timespan_inventory.extend(value)
13         return timespan_inventory.__illustrate__(
14             key='voice_name',
15             range_=range_,
16             scale=scale,
17         )
```

## A.57 CONSORTTOOLS.TIMESPANMAKER

```
1 # -*- encoding: utf-8 -*-
2 from __future__ import print_function
3 import abc
4 import collections
5 from abjad import new
6 from abjad.tools import abctools
7 from abjad.tools import durationtools
8 from abjad.tools import rhythmmakertools
9 from abjad.tools import timespanools
10
11
12 class TimespanMaker(abctools.AbjadValueObject):
13     r'''Abstract base class for timespan makers.
14     '''
15
16     ### CLASS VARIABLES ###
17
18     __slots__ = (
19         '_output_masks',
20         '_padding',
21         '_seed',
22         '_timespanSpecifier',
23     )
24
25     ### INITIALIZER ###
26
27     @abc.abstractmethod
28     def __init__(
29         self,
30         output_masks=None,
31         padding=None,
32         seed=None,
```

```

33     timespanSpecifier=None,
34     ):
35     import consort
36     if output_masks is not None:
37         if isinstance(output_masks, rhythmmakertools.BooleanPattern):
38             output_masks = (output_masks,)
39             output_masks = rhythmmakertools.BooleanPatternInventory(
40                 items=output_masks,
41             )
42             self._output_masks = output_masks
43             if padding is not None:
44                 padding = durationtools.Duration(padding)
45             self._padding = padding
46             if seed is not None:
47                 seed = int(seed)
48             self._seed = seed
49             if timespanSpecifier is not None:
50                 assert isinstance(timespanSpecifier, consort.TimespanSpecifier)
51             self._timespanSpecifier = timespanSpecifier
52
53     ### SPECIAL METHODS ###
54
55     def __call__(
56         self,
57         layer=None,
58         music_specifiers=None,
59         rotation=None,
60         silenced_context_names=None,
61         target_timespan=None,
62         timespan_inventory=None,
63     ):
64         if not isinstance(timespan_inventory, timespantools.TimespanInventory):
65             timespan_inventory = timespantools.TimespanInventory(
66                 timespan_inventory,
67             )
68         if target_timespan is None:
69             if timespan_inventory:
70                 target_timespan = timespan_inventory.timespan
71             else:
72                 raise TypeError
73         assert isinstance(timespan_inventory, timespantools.TimespanInventory)
74         if not music_specifiers:
75             return timespan_inventory
76         music_specifiers = self._coerce_music_specifiers(music_specifiers)
77         new_timespans = self._make_timespans(
78             layer=layer,
79             music_specifiers=music_specifiers,
80             target_timespan=target_timespan,
81             timespan_inventory=timespan_inventory,
82         )
83         self._cleanup_silent_timespans(
84             layer=layer,
85             silenced_context_names=silenced_context_names,
86             timespans=new_timespans,

```

```

87         )
88     timespan_inventory.extend(new_timespans)
89     timespan_inventory.sort()
90     return timespan_inventory
91
92     ##### PRIVATE METHODS #####
93
94     @staticmethod
95     def _coerce_music_specifiers(music_specifiers):
96         import consort
97         result = collections.OrderedDict()
98         prototype = (
99             consort.MusicSpecifierSequence,
100            consort.CompositeMusicSpecifier,
101        )
102        for context_name, music_specifier in music_specifiers.items():
103            if music_specifier is None:
104                music_specifier = [None]
105            if not isinstance(music_specifier, prototype):
106                music_specifier = consort.MusicSpecifierSequence(
107                    music_specifiers=music_specifier,
108                )
109            result[context_name] = music_specifier
110        return result
111
112    def _cleanup_silent_timespans(
113        self,
114        layer,
115        silenced_context_names,
116        timespans,
117    ):
118        import consort
119        if not silenced_context_names or not timespans:
120            return
121
122        silent_timespans_by_context = {}
123        for context_name in silenced_context_names:
124            if context_name not in silent_timespans_by_context:
125                silent_timespans_by_context[context_name] = \
126                    timespantools.TimespanInventory()
127
128        sounding_timespans_by_context = {}
129        sounding_timespans = timespantools.TimespanInventory()
130
131        for timespan in timespans:
132            voice_name = timespan.voice_name
133            if isinstance(timespan, consort.PerformedTimespan):
134                if voice_name not in sounding_timespans_by_context:
135                    sounding_timespans_by_context[voice_name] = \
136                        timespantools.TimespanInventory()
137                    sounding_timespans_by_context[voice_name].append(timespan)
138                    sounding_timespans.append(timespan)
139            else:
140                if voice_name not in silent_timespans_by_context:

```

```

141         silent_timepans_by_context[voice_name] = \
142             timespantools.TimespanInventory()
143         silent_timepans_by_context[voice_name].append(timespan)
144
145     sounding_timepans.sort()
146     sounding_timepans.compute_logical_or()
147
148     # Create silences.
149     for shard in sounding_timepans.partition(True):
150         for context_name in silenced_context_names:
151             timespan = consort.SilentTimespan(
152                 layer=layer,
153                 voice_name=context_name,
154                 start_offset=shard.start_offset,
155                 stop_offset=shard.stop_offset,
156                 )
157             silent_timepans_by_context[context_name].append(timespan)
158
159     # Remove any overlap between performed and silent timepans.
160     # Then add the silent timepans into the original timespan inventory.
161     for context_name, silent_timepans in \
162         sorted(silent_timepans_by_context.items()):
163         silent_timepans.sort()
164         if context_name in sounding_timepans_by_context:
165             for timespan in sounding_timepans_by_context[context_name]:
166                 silent_timepans -= timespan
167             timespans.extend(silent_timepans)
168
169     ### PUBLIC METHODS ###
170
171     def rotate(self, rotation):
172         seed = self.seed or 0
173         seed = seed + rotation
174         return new(self, seed=seed)
175
176     ### PUBLIC PROPERTIES ###
177
178     @property
179     def is_dependent(self):
180         return False
181
182     @property
183     def output_masks(self):
184         return self._output_masks
185
186     @property
187     def padding(self):
188         return self._padding
189
190     @property
191     def seed(self):
192         return self._seed
193
194     @property

```

```
195     def timespanSpecifier(self):
196         return self._timespanSpecifier
```

## A.58 CONSORTTOOLS.TIMESPANSIMULTANEITY

```
1 # -*- encoding: utf-8 -*-
2 from abjad import abctools
3
4
5 class TimespanSimultaneity(abctools.AbjadObject):
6     r'''A simultaneity of timespans in a timespan collection.
7     '''
8
9     ### CLASS VARIABLES ###
10
11     __slots__ = (
12         '_timespan_collection',
13         '_overlap_timespans',
14         '_start_timespans',
15         '_start_offset',
16         '_stop_timespans',
17     )
18
19     ### INITIALIZER ###
20
21     def __init__(
22         self,
23         timespan_collection=None,
24         overlap_timespans=None,
25         start_timespans=None,
26         start_offset=None,
27         stop_timespans=None,
28     ):
29         import consort
30         if timespan_collection is not None:
31             prototype = consort.TimespanCollection
32             assert isinstance(timespan_collection, prototype)
33             self._timespan_collection = timespan_collection
34             self._start_offset = start_offset
35             assert isinstance(start_timespans, (tuple, type(None)))
36             assert isinstance(stop_timespans, (tuple, type(None)))
37             assert isinstance(overlap_timespans, (tuple, type(None)))
38             self._start_timespans = start_timespans
39             self._stop_timespans = stop_timespans
40             self._overlap_timespans = overlap_timespans
41
42     ### SPECIAL METHODS ###
43
44     def __repr__(self):
45         r'''Gets the repr of this simultaneity.
46         '''
47         return '<{}({} <<{})>>'.format(
48             type(self).__name__,
49             str(self.start_offset),
```

```

50         len(self.start_timepans) + len(self.overlap_timepans),
51     )
52
53     ### PUBLIC PROPERTIES ###
54
55     @property
56     def next_simultaneity(self):
57         r'''Gets the next simultaneity in this simultaneity's timespan
58         collection.
59         '''
60         tree = self._timespan_collection
61         if tree is None:
62             return None
63         start_offset = tree.get_start_offset_after(self.start_offset)
64         if start_offset is None:
65             return None
66         return tree.get_simultaneity_at(start_offset)
67
68     @property
69     def next_start_offset(self):
70         r'''Gets the next simultaneity start offset in this simultaneity's
71         timespan collection.
72         '''
73         tree = self._timespan_collection
74         if tree is None:
75             return None
76         start_offset = tree.get_start_offset_after(self.start_offset)
77         return start_offset
78
79     @property
80     def overlap_timepans(self):
81         r'''Gets the timepans in this simultaneity which overlap this
82         simultaneity's start offset.
83         '''
84         return self._overlap_timepans
85
86     @property
87     def previous_simultaneity(self):
88         r'''Gets the previous simultaneity in this simultaneity's timespan
89         collection.
90         '''
91         tree = self._timespan_collection
92         if tree is None:
93             return None
94         start_offset = tree.get_start_offset_before(self.start_offset)
95         if start_offset is None:
96             return None
97         return tree.get_simultaneity_at(start_offset)
98
99     @property
100    def previous_start_offset(self):
101        r'''Gets the previous simultaneity start offset in this simultaneity's
102        timespan collection.
103        '''

```

```

104     tree = self._timespan_collection
105     if tree is None:
106         return None
107     start_offset = tree.get_start_offset_before(self.start_offset)
108     return start_offset
109
110     @property
111     def start_offset(self):
112         '''Gets this simultaneity's start offset.
113         '''
114         return self._start_offset
115
116     @property
117     def start_timespans(self):
118         '''Gets the timespans in this simultaneity which start at this
119         simultaneity's start offset.
120         '''
121         return self._start_timespans
122
123     @property
124     def stop_timespans(self):
125         '''Gets the timespans in this simultaneity which stop at this
126         simultaneity's start offset.
127         '''
128         return self._stop_timespans
129
130     @property
131     def timespan_collection(self):
132         '''Gets this simultaneity's timespan collection.
133         '''
134         return self._timespan_collection

```

## A.59 CONSORTTOOLS.TIMESPANSPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3 from abjad.tools import durationtools
4
5
6 class TimespanSpecifier(abctools.AbjadValueObject):
7
8     """CLASS VARIABLES"""
9
10    __slots__ = (
11        '_forbid_fusing',
12        '_forbid_splitting',
13        '_minimum_duration',
14    )
15
16    """INITIALIZER"""
17
18    def __init__(
19        self,
20        forbid_fusing=None,

```

```

21     forbid_splitting=None,
22     minimum_duration=None,
23     ):
24     if forbid_fusing is not None:
25         forbid_fusing = bool(forbid_fusing)
26     self._forbid_fusing = forbid_fusing
27     if forbid_splitting is not None:
28         forbid_splitting = bool(forbid_splitting)
29     self._forbid_splitting = forbid_splitting
30     if minimum_duration is not None:
31         minimum_duration = durationtools.Duration(minimum_duration)
32     self._minimum_duration = minimum_duration
33
34     ### PUBLIC PROPERTIES ###
35
36     @property
37     def forbid_fusing(self):
38         return self._forbid_fusing
39
40     @property
41     def forbid_splitting(self):
42         return self._forbid_splitting
43
44     @property
45     def minimum_duration(self):
46         return self._minimum_duration

```

This page intentionally left blank.

# B

## *zaira* source code

### B.1 ZAIRA MAKERS SOURCE

#### B.1.1 ZAIRA.MAKERS.PERCUSION

```
1 # -*- encoding: utf-8 -*-
2 from abjad.tools import pitchtools
3
4
5 class Percussion(object):
6
7     BRAKE_DRUM = pitchtools.NamedPitch("g'")
8     HIGH_CYMBAL = pitchtools.NamedPitch("e'")
9     MIDDLE_CYMBAL = pitchtools.NamedPitch("c'")
10    LOW_CYMBAL = pitchtools.NamedPitch("a")
11    TAM_TAM = pitchtools.NamedPitch("f")
12
13    TAMBOURINE = pitchtools.NamedPitch("e'")
14    GUERO = pitchtools.NamedPitch("c")
15    BAMBOO_WINDCHIMES = pitchtools.NamedPitch("a")
16
17    HIGH_TOM = pitchtools.NamedPitch("f'")
18    LOW_TOM = pitchtools.NamedPitch("d'")
19    KICK_DRUM = pitchtools.NamedPitch("b")
20    BASS_DRUM = pitchtools.NamedPitch("g")
```

#### B.1.2 ZAIRA.MAKERS.ZAIRASCORETEMPLATE

```
1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad.tools import abctools
4 from abjad.tools import indicatortools
5 from abjad.tools import instrumenttools
6 from abjad.tools import scoretools
```

```

7 import consort
8
9
10 class ZairaScoreTemplate(abctools.AbjadValueObject):
11     r'''Zaira score template.
12
13     :::
14
15     >>> import zaira
16     >>> template = zaira.makers.ZairaScoreTemplate()
17     >>> score = template()
18     >>> print(format(score))
19     \context Score = "Zaira Score" <<
20         \tag #'time
21         \context TimeSignatureContext = "Time Signature Context" {
22             }
23         \context EnsembleGroup = "Wind Section Staff Group" <<
24             \tag #'flute
25             \context PerformerGroup = "Flute Performer Group" \with {
26                 instrumentName = \markup { Flute }
27                 shortInstrumentName = \markup { Fl. }
28             } <<
29                 \context FluteStaff = "Flute Staff" {
30                     \context Voice = "Flute Voice" {
31                         \clef "treble"
32                     }
33                 }
34             >>
35             \tag #'oboe
36             \context PerformerGroup = "Oboe Performer Group" \with {
37                 instrumentName = \markup { Oboe }
38                 shortInstrumentName = \markup { Ob. }
39             } <<
40                 \context OboeStaff = "Oboe Staff" {
41                     \context Voice = "Oboe Voice" {
42                         \clef "treble"
43                     }
44                 }
45             >>
46             \tag #'clarinet-in-b-flat
47             \context PerformerGroup = "Clarinet In B-Flat Performer Group" \with {
48                 instrumentName = \markup { Clarinet in B-flat }
49                 shortInstrumentName = \markup { Cl. in B-flat }
50             } <<
51                 \context ClarinetInBFlatStaff = "Clarinet In B-Flat Staff" {
52                     \context Voice = "Clarinet In B-Flat Voice" {
53                         \clef "treble"
54                     }
55                 }
56             >>
57             >>
58             \tag #'percussion
59             \context EnsembleGroup = "Percussion Section Staff Group" <<
60                 \context PerformerGroup = "Metals Performer Group" \with {

```

```

61     instrumentName = \markup { Metals }
62     shortInstrumentName = \markup { Metals }
63 } <<
64     \context MetalsStaff = "Metals Staff" {
65         \context Voice = "Metals Voice" {
66             \clef "percussion"
67         }
68     }
69 >>
70     \context PerformerGroup = "Woods Performer Group" \with {
71         instrumentName = \markup { Woods }
72         shortInstrumentName = \markup { Woods }
73 } <<
74     \context WoodsStaff = "Woods Staff" {
75         \context Voice = "Woods Voice" {
76             \clef "percussion"
77         }
78     }
79 >>
80     \context PerformerGroup = "Drums Performer Group" \with {
81         instrumentName = \markup { Drums }
82         shortInstrumentName = \markup { Drums }
83 } <<
84     \context DrumsStaff = "Drums Staff" {
85         \context Voice = "Drums Voice" {
86             \clef "percussion"
87         }
88     }
89 >>
90 >>
91     \tag #'piano
92     \context PianoStaff = "Piano Performer Group" \with {
93         instrumentName = \markup { Piano }
94         shortInstrumentName = \markup { Pf. }
95 } <<
96     \context PianoUpperStaff = "Piano Upper Staff" {
97         \context Voice = "Piano Upper Voice" {
98             \clef "treble"
99         }
100    }
101    \context Dynamics = "Piano Dynamics" {
102    }
103    \context PianoLowerStaff = "Piano Lower Staff" {
104        \context Voice = "Piano Lower Voice" {
105            \clef "bass"
106        }
107    }
108    \context Dynamics = "Piano Pedals" {
109    }
110 >>
111     \context EnsembleGroup = "String Section Staff Group" <<
112         \tag #'violin
113         \context StringPerformerGroup = "Violin Performer Group" \with {
114             instrumentName = \markup { Violin }

```

```

115         shortInstrumentName = \markup { \text{Vn.} }
116     } <<
117         \context StringStaff = "Violin Staff" {
118             \context Voice = "Violin Voice" {
119                 \clef "treble"
120             }
121         }
122     >>
123     \tag #'viola
124     \context StringPerformerGroup = "Viola Performer Group" \with {
125         instrumentName = \markup { \text{Viola} }
126         shortInstrumentName = \markup { \text{Va.} }
127     } <<
128         \context StringStaff = "Viola Staff" {
129             \context Voice = "Viola Voice" {
130                 \clef "alto"
131             }
132         }
133     >>
134     \tag #'cello
135     \context StringPerformerGroup = "Cello Performer Group" \with {
136         instrumentName = \markup { \text{Cello} }
137         shortInstrumentName = \markup { \text{Vc.} }
138     } <<
139         \context StringStaff = "Cello Staff" {
140             \context Voice = "Cello Voice" {
141                 \clef "bass"
142             }
143         }
144     >>
145     >>
146     >>
147
148     ...
149
150     ### CLASS VARIABLES ###
151
152     __slots__ = (
153         '_context_nameAbbreviations',
154     )
155
156     ### INITIALIZER ###
157
158     def __init__(self):
159         self._context_nameAbbreviations = collections.OrderedDict()
160
161     ### SPECIAL METHODS ###
162
163     def __call__(self):
164
165         manager = consort.ScoreTemplateManager
166
167     ### WINDS ###
168

```

```

169     flute = manager.make_single_wind_performer(
170         clef=indicatortools.Clef('treble'),
171         instrument=instrumenttools.Flute(),
172         score_template=self,
173     )
174
175     oboe = manager.make_single_wind_performer(
176         clef=indicatortools.Clef('treble'),
177         instrument=instrumenttools.Oboe(),
178         score_template=self,
179     )
180
181     clarinet = manager.make_single_wind_performer(
182         abbreviation='clarinet',
183         clef=indicatortools.Clef('treble'),
184         instrument=instrumenttools.ClarinetInBFlat(),
185         score_template=self,
186     )
187
188     winds = manager.make_ensemble_group(
189         name='Wind Section Staff Group',
190         performer_groups=[
191             flute,
192             oboe,
193             clarinet,
194         ],
195     )
196
197     ### PERCUSSION ###
198
199     metals = manager.make_single_basic_performer(
200         clef=indicatortools.Clef('percussion'),
201         instrument=instrumenttools.Percussion(
202             instrument_name='Metals',
203             short_instrument_name='Metals',
204         ),
205         score_template=self,
206     )
207
208     woods = manager.make_single_basic_performer(
209         clef=indicatortools.Clef('percussion'),
210         instrument=instrumenttools.Percussion(
211             instrument_name='Woods',
212             short_instrument_name='Woods',
213         ),
214         score_template=self,
215     )
216
217     drums = manager.make_single_basic_performer(
218         clef=indicatortools.Clef('percussion'),
219         instrument=instrumenttools.Percussion(
220             instrument_name='Drums',
221             short_instrument_name='Drums',
222         ),

```

```

223     score_template=self,
224 )
225
226 percussion = manager.make_ensemble_group(
227     label='percussion',
228     name='Percussion Section Staff Group',
229     performer_groups=[
230         metals,
231         woods,
232         drums,
233     ],
234 )
235
236     ### PIANO ###
237
238 piano = manager.make_single_piano_performer(
239     instrument=instrumenttools.Piano(),
240     score_template=self,
241 )
242
243     ### STRINGS ###
244
245 violin = manager.make_single_string_performer(
246     clef=indicatortools.Clef('treble'),
247     instrument=instrumenttools.Violin(),
248     split=False,
249     score_template=self,
250 )
251
252 viola = manager.make_single_string_performer(
253     clef=indicatortools.Clef('alto'),
254     instrument=instrumenttools.Viola(),
255     split=False,
256     score_template=self,
257 )
258
259 cello = manager.make_single_string_performer(
260     clef=indicatortools.Clef('bass'),
261     instrument=instrumenttools.Cello(),
262     split=False,
263     score_template=self,
264 )
265
266 strings = manager.make_ensemble_group(
267     name='String Section Staff Group',
268     performer_groups=[
269         violin,
270         viola,
271         cello,
272     ],
273 )
274
275     ### SCORE ###
276

```

```

277     time_signature_context = manager.make_time_signature_context()
278
279     score = scoretools.Score(
280         [
281             time_signature_context,
282             winds,
283             percussion,
284             piano,
285             strings,
286         ],
287         name='Zaira Score',
288     )
289
290     return score
291
292     ##### PUBLIC PROPERTIES #####
293
294     @property
295     def context_nameAbbreviations(self):
296         return self._context_nameAbbreviations

```

### B.1.3 ZAIRA.MAKERS.ZAIRASEGMENTMAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import markuptools
4
5
6 class ZairaSegmentMaker(consort.SegmentMaker):
7
8     ##### CLASS VARIABLES #####
9
10    __slots__ = ()
11
12    ##### INITIALIZER #####
13
14    def __init__(
15        self,
16        annotate_colors=None,
17        annotate_phrasing=None,
18        annotate_timestamps=None,
19        desired_duration_in_seconds=None,
20        discard_final_silence=None,
21        maximum_meter_run_length=None,
22        name=None,
23        omit_stylesheets=None,
24        permitted_time_signatures=None,
25        repeat=None,
26        score_template=None,
27        settings=None,
28        tempo=None,
29        timespan_quantization=None,
30    ):
31        import zaira

```

```

32     permitted_time_signatures = permitted_time_signatures or \
33         zaira.materials.time_signatures
34     score_template = score_template or zaira.makers.ZairaScoreTemplate()
35     consort.SegmentMaker.__init__(
36         self,
37         annotate_colors=annotate_colors,
38         annotate_phrasing=annotate_phrasing,
39         annotate_timespans=annotate_timespans,
40         desired_duration_in_seconds=desired_duration_in_seconds,
41         discard_final_silence=discard_final_silence,
42         maximum_meter_run_length=maximum_meter_run_length,
43         name=name,
44         omit_stylesheets=omit_stylesheets,
45         permitted_time_signatures=permitted_time_signatures,
46         repeat=repeat,
47         score_template=score_template,
48         settings=settings,
49         tempo=tempo,
50         timespan_quantization=timespan_quantization,
51     )
52
53     ### PUBLIC PROPERTIES ###
54
55     @property
56     def final_markup(self):
57         jamaica_plain = markuptools.Markup('Jamaica Plain, OR')
58         queens = markuptools.Markup('Fresh Meadows, NY')
59         date = markuptools.Markup('June 2014 - September 2014')
60         null = markuptools.Markup.null()
61         contents = [
62             null,
63             null,
64             null,
65             jamaica_plain,
66             queens,
67             date,
68         ]
69         markup = markuptools.Markup.right_column(contents)
70         markup = markup.italic()
71         return markup
72
73     @property
74     def score_package_name(self):
75         return 'zaira'

```

## B.2 ZAIRA

### MATERIALS

SOURCE

#### B.2.1 ZAIRA.MATERIALS.BACKGROUND\_DYNAMIC\_ATTACHMENT\_EXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5

```

```

6 background_dynamic_attachment_expression = consort.AttachmentExpression(
7     attachments=(
8         consort.SimpleDynamicExpression('ppp'),
9         consort.SimpleDynamicExpression('p'),
10        consort.SimpleDynamicExpression('pp'),
11    ),
12    selector=selectortools.select_pitched_runs(),
13 )

```

## B.2.2 ZAIRA.MATERIALS.BRAZIL\_NUT\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import spannertools
5 from abjad.tools import selectortools
6 import consort
7 import zaira
8
9
10 brazil_nut_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
13         clef_spacer=consort.ClefSpanner('percussion'),
14         staff_lines_spacer=spannertools.StaffLinesSpanner(
15             lines=(4, -4),
16             overrides={
17                 'note_head__no_ledgers': True,
18                 'note_head__style': 'cross',
19             }
20         ),
21         staccato=consort.AttachmentExpression(
22             attachments=indicatortools.Articulation('.'),
23             selector=selectortools.Selector(
24                 .by_logical_tie(pitched=True
25                 .by_duration('<', (1, 8)
26                 .by_length(1)
27             ),
28             stem_tremolo_spacer=consort.AttachmentExpression(
29                 attachments=spannertools.StemTremoloSpanner(),
30                 selector=selectortools.Selector(
31                     .by_logical_tie(pitched=True
32                     .by_duration('>', (1, 16))
33             ),
34             text_spacer=consort.AttachmentExpression(
35                 attachments=consort.ComplexTextSpanner(
36                     markup=Markup(r'\concat { \vstrut shaker }')
37                     .italic()
38                     .pad_around(0.5)
39                     .box(),
40             ),
41             selector=selectortools.Selector().by_leaves(),
42         ),
43     ),

```

```

44     pitch_handler=consort.AbsolutePitchHandler(
45         pitches_are_nonsemantic=True,
46     ),
47     rhythm_maker=zaira.materials.undergrowth_rhythm_maker,
48 )

```

### B.2.3 ZAIRA.MATERIALS.CELLO\_SOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 from abjad.tools.topleveltools import new
7 import consort
8 import zaira
9
10
11 cello_solo_musicSpecifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
14         trill_spanner=consort.AttachmentExpression(
15             attachments=(
16                 None,
17                 spannertools.ComplexTrillSpanner(interval='+m3'),
18                 None,
19                 spannertools.ComplexTrillSpanner(interval='+m3'),
20                 None,
21                 None,
22                 spannertools.ComplexTrillSpanner(interval='+M2'),
23             ),
24             selector=selectortools.Selector()
25             .by_leaves()
26             .by_logical_tie(pitched=True)
27         ),
28         tenuto=consort.AttachmentExpression(
29             attachments=indicatortools.Articulation('tenuto'),
30             selector=selectortools.Selector()
31             .by_leaves()
32             .by_logical_tie(pitched=True)[0],
33         ),
34         text_spanner=consort.AttachmentExpression(
35             attachments=(
36                 consortium.ComplexTextSpanner(
37                     markup=Markup(r'\concat { \vstrut "col legno" }')
38                     .italic()
39                     .pad_around(0.5)
40                     .box(),
41             ),
42             None,
43             spannertools.Glissando(),
44         ),
45         selector=selectortools.select_pitched_runs(),
46     ),

```

```

47     ),
48     pitch_handler=consort.AbsolutePitchHandler(
49         pitchSpecifier="d, f, d, fqs, ef, d, ef, f, fqs, d, g, d, d, as,",
50         pitchApplicationRate='division',
51         ),
52     rhythm_maker=new(
53         zaira.materials.reiterating_rhythm_maker,
54         denominators=(8, 4, 8, 1),
55         ),
56 )

```

#### B.2.4 ZAIRA.MATERIALS.DENSE\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 dense_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=rhythmmakertools.Talea(
8         counts=(0, 3, 4, 2, 5),
9         denominator=16,
10        ),
11     playing_talea=rhythmmakertools.Talea(
12         counts=(6, 8, 4, 5, 6, 6, 4),
13         denominator=16,
14        ),
15     playing_groupings=(2, 1, 2, 3, 1, 1, 2, 2),
16     repeat=True,
17     silence_talea=rhythmmakertools.Talea(
18         counts=(2, 4, 6, 3, 4, 10),
19         denominator=16,
20        ),
21     step_anchor=Right,
22     synchronize_groupings=False,
23     synchronize_step=False,
24     timespanSpecifier=consort.TimespanSpecifier(
25         minimumDuration=durationtools.Duration(1, 8),
26         ),
27 )

```

#### B.2.5 ZAIRA.MATERIALS.DRUM\_AGITATION\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import pitchtools
4 from abjad.tools import selectortools
5 from abjad.tools.topleveltools import new
6 import consort
7 import zaira
8
9
10 drum_agitation_music_specifier = consort.MusicSpecifier(
11     attachmentHandler=consort.AttachmentHandler(

```

```

12     dynamic_expression=zaira.materials.foreground_dynamic_attachment_expression,
13     accent=consort.AttachmentExpression(
14         attachments=indicatortools.Articulation('accent'),
15         selector=selectortools.select_first_logical_tie_in_pitched_runs()[0],
16         ),
17     staccato=consort.AttachmentExpression(
18         attachments=indicatortools.Articulation('staccato'),
19         selector=selectortools.select_all_but_first_logical_tie_in_pitched_runs()[0],
20         ),
21     ),
22     pitch_handler=consort.AbsolutePitchHandler(
23         pitch_specifier=pitchtools.PitchSegment(
24             items=(
25                 zaira.makers.Percussion.HIGH_TOM,
26                 zaira.makers.Percussion.LOW_TOM,
27                 zaira.makers.Percussion.BASS_DRUM,
28                 zaira.makers.Percussion.HIGH_TOM,
29                 zaira.makers.Percussion.BASS_DRUM,
30                 zaira.makers.Percussion.LOW_TOM,
31                 zaira.makers.Percussion.HIGH_TOM,
32                 zaira.makers.Percussion.BASS_DRUM,
33                 zaira.makers.Percussion.LOW_TOM,
34             ),
35         ),
36     ),
37     rhythm_maker=new(
38         zaira.materials.stuttering_rhythm_maker,
39         extra_counts_per_division=(2, 1, 2, 1, 0, 2, 1, 0),
40     ),
41 )

```

## B.2.6 ZAIRA.MATERIALS.DRUM\_BRUSHED\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import pitchtools
4 from abjad.tools import spannertools
5 from abjad.tools import selectortools
6 import consort
7 import zaira
8
9
10 drum_brushed_musicSpecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.midground_dynamic_attachment_expression,
13         text_spanner=consort.AttachmentExpression(
14             attachments=consort.ComplexTextSpanner(
15                 markup=Markup(r'\concat { \vstrut brush }')
16                 .italic()
17                 .pad_around(0.5)
18                 .box(),
19             ),
20             selector=selectortools.Selector().by_leaves(),
21         ),

```

```

22     stem_tremolo_spinner=consort.AttachmentExpression(
23         attachments=(
24             spanertools.StemTremoloSpanner(),
25             None,
26         ),
27         selector=selectortools.select_pitched_runs(),
28     ),
29 ),
30 pitch_handler=consort.AbsolutePitchHandler(
31     pitchSpecifier=pitchtools.PitchSegment(
32         items=(
33             zaira.makers.Percussion.HIGH_TOM,
34             zaira.makers.Percussion.LOW_TOM,
35             zaira.makers.Percussion.BASS_DRUM,
36             zaira.makers.Percussion.HIGH_TOM,
37             zaira.makers.Percussion.BASS_DRUM,
38             zaira.makers.Percussion.LOW_TOM,
39             zaira.makers.Percussion.HIGH_TOM,
40             zaira.makers.Percussion.BASS_DRUM,
41             zaira.makers.Percussion.LOW_TOM,
42         ),
43     ),
44 ),
45 rhythm_maker=zaira.materials.sustained_rhythm_maker,
46 )

```

## B.2.7 ZAIRA.MATERIALS.DRUM\_HEARTBEAT\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import pitchtools
4 import consort
5 import zaira
6
7
8 drum_heartbeat_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         dynamic_expression=zaira.materials.midground_dynamic_attachment_expression,
11     ),
12     pitch_handler=consort.AbsolutePitchHandler(
13         pitchSpecifier=pitchtools.PitchSegment(
14             items=[zaira.makers.Percussion.KICK_DRUM],
15         ),
16     ),
17     rhythm_maker=new(
18         zaira.materials.stuttering_rhythm_maker,
19         inciseSpecifier__talea_denominator=8,
20     ),
21 )

```

## B.2.8 ZAIRA.MATERIALS.DRUM\_STORM\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools

```

```

3 from abjad.tools import pitchtools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 from abjad import new
7 import consort
8 import zaira
9
10
11 drum_storm_musicSpecifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
14         stem_tremolo_spanner=consort.AttachmentExpression(
15             attachments=spannertools.StemTremoloSpanner(),
16             selector=selectortools.Selector(
17                 ).by_logical_tie(pitched=True
18                 ).by_duration('>', (1, 16))
19             ),
20         accent=consort.AttachmentExpression(
21             attachments=(
22                 indicatortools.Articulation('accent'),
23                 ),
24             selector=selectortools.Selector(
25                 ).by_logical_tie(pitched=True)[0],
26             ),
27         ),
28     pitch_handler=consort.AbsolutePitchHandler(
29         pitchSpecifier=pitchtools.PitchSegment(
30             items=(
31                 zaira.makers.Percussion.HIGH_TOM,
32                 zaira.makers.Percussion.LOW_TOM,
33                 zaira.makers.Percussion.BASS_DRUM,
34                 zaira.makers.Percussion.HIGH_TOM,
35                 zaira.makers.Percussion.BASS_DRUM,
36                 zaira.makers.Percussion.LOW_TOM,
37                 zaira.makers.Percussion.HIGH_TOM,
38                 zaira.makers.Percussion.BASS_DRUM,
39                 zaira.makers.Percussion.LOW_TOM,
40             ),
41         ),
42     ),
43     rhythm_maker=new(
44         zaira.materials.reiterating_rhythm_maker,
45         denominators=(16, 16, 4, 16, 4),
46         extra_counts_per_division=(0, 1, 0, 1, 2),
47     ),
48 )

```

## B.2.9 ZAIRA.MATERIALS.DRUM\_TRANQUILO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import pitchtools
4 import consort
5 import zaira

```

```

6
7
8 drum_tranquilo_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
11         laissez_vibrer=zaira.materials.laissez_vibrer_attachment_expression,
12     ),
13     pitch_handler=consort.AbsolutePitchHandler(
14         pitch_specifier=pitchtools.PitchSegment(
15             items=(
16                 zaira.makers.Percussion.HIGH_TOM,
17                 zaira.makers.Percussion.LOW_TOM,
18                 zaira.makers.Percussion.BASS_DRUM,
19                 zaira.makers.Percussion.HIGH_TOM,
20                 zaira.makers.Percussion.BASS_DRUM,
21                 zaira.makers.Percussion.LOW_TOM,
22                 zaira.makers.Percussion.HIGH_TOM,
23                 zaira.makers.Percussion.BASS_DRUM,
24                 zaira.makers.Percussion.LOW_TOM,
25             ),
26         ),
27     ),
28     rhythm_maker=new(
29         zaira.materials.stuttering_rhythm_maker,
30         extra_counts_per_division=None,
31         inciseSpecifier__prefix_talea=(1,),
32         inciseSpecifier__prefix_counts=(1,),
33         inciseSpecifier__talea_denominator=16,
34     ),
35 )

```

## B.2.10 ZAIRA.MATERIALS.ERRATIC\_DYNAMIC\_ATTACHMENT\_EXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 erratic_dynamic_attachment_expression = consort.AttachmentExpression(
7     attachments=(
8         consort.SimpleDynamicExpression(
9             hairpin_start_token='p',
10            hairpin_stop_token='f',
11            minimum_duration=durationtools.Duration(1, 8),
12        ),
13        consort.SimpleDynamicExpression(
14            hairpin_start_token='f',
15            hairpin_stop_token='p',
16            minimum_duration=durationtools.Duration(1, 8),
17        ),
18        consort.SimpleDynamicExpression(
19            hairpin_start_token='mf',
20            hairpin_stop_token='o',
21            minimum_duration=durationtools.Duration(1, 8),

```

```

22         ),
23     consort.SimpleDynamicExpression(
24         hairpin_start_token='o',
25         hairpin_stop_token='ff',
26         minimum_duration=durationtools.Duration(1, 8),
27         ),
28     ),
29     selector=selectortools.select_pitched_runs(),
30 )

```

## B.2.11 ZAIRA.MATERIALS.FLOURISH\_RHYTHM MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 flourish_rhythm_maker = rhythmmakertools.TaleaRhythmMaker(
6     talea=rhythmmakertools.Talea(
7         counts=(1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2,),
8         denominator=32,
9         ),
10    split_divisions_by_counts=None,
11    extra_counts_per_division=(0, 0, 0, 1, 0, 0, 1, 1),
12    beamSpecifier=rhythmmakertools.BeamSpecifier(
13        beam_each_division=False,
14        beam_divisions_together=False,
15        ),
16    burnishSpecifier=rhythmmakertools.BurnishSpecifier(
17        left_classes=(-1, 1, -1, -1, 1),
18        right_classes=(1, -1, -1, 1, -1, 1, -1),
19        left_counts=(1,),
20        right_counts=(1,),
21        outer_divisions_only=True,
22        ),
23    duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
24        decrease_durations_monotonically=True,
25        ),
26    tieSpecifier=rhythmmakertools.TieSpecifier(
27        tie_across_divisions=True,
28        ),
29    tie_split_notes=False,
30    tuplet_spellingSpecifier=rhythmmakertools.TupletSpellingSpecifier(
31        avoid_dots=False,
32        is_diminution=False,
33        simplify_tuples=False,
34        ),
35    )

```

## B.2.12 ZAIRA.MATERIALS.FOREGROUND\_DYNAMIC\_ATTACHMENT\_EXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4

```

```

5
6 foreground_dynamic_attachment_expression = consort.AttachmentExpression(
7     attachments=(
8         consort.SimpleDynamicExpression('fff'),
9         consort.SimpleDynamicExpression('f'),
10        consort.SimpleDynamicExpression('ff'),
11        consort.SimpleDynamicExpression('mf'),
12    ),
13    selector=selectortools.select_pitched_runs()[0],
14 )

```

### B.2.13 ZAIRA.MATERIALS.GLISSANDO\_RHYTHM MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 glissando_rhythm_maker = rhythmmakertools.IncisedRhythmMaker(
6     inciseSpecifier=rhythmmakertools.InciseSpecifier(
7         prefix_talea=(1,),
8         prefix_counts=(0,),
9         suffix_talea=(1,),
10        suffix_counts=(1,),
11        talea_denominator=16,
12        body_ratio=mathtools.Ratio([1]),
13        outer_divisions_only=True,
14    ),
15    beamSpecifier=rhythmmakertools.BeamSpecifier(
16        beam_each_division=False,
17        beam_divisions_together=False,
18    ),
19    duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
20        decrease_durations_monotonically=True,
21        forbidden_written_duration=durationtools.Duration(1, 2),
22    ),
23    tuplet_spellingSpecifier=rhythmmakertools.TupletSpellingSpecifier(
24        avoid_dots=True,
25        is_diminution=True,
26        simplify_tuples=True,
27    ),
28 )

```

### B.2.14 ZAIRA.MATERIALS.GRANULAR\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 granular_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=rhythmmakertools.Talea(
8         counts=(1, 4, 3),
9         denominator=16,
10    ),

```

```

11     playing_talea=rhythmmakertools.Talea(
12         counts=(1, 2, 1, 2, 2, 1, 2),
13         denominator=16,
14         ),
15     playing_groupings=(1, 1, 2, 1),
16     repeat=True,
17     silence_talea=rhythmmakertools.Talea(
18         counts=(4, 8, 7, 9, 2, 13),
19         denominator=8,
20         ),
21     step_anchor=Right,
22     synchronize_groupings=False,
23     synchronize_step=False,
24     timespanSpecifier=consort.TimespanSpecifier(
25         minimum_duration=None,
26         ),
27     )

```

## B.2.15 ZAIRA.MATERIALS.LAISSEZ\_VIBRER\_ATTACHMENT\_EXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 laissez_vibrer_attachment_expression = consort.AttachmentExpression(
7     attachments=(
8         (
9             indicatortools.LaissezVibrer(),
10            markuptools.Markup('L.V.', Up)
11                .caps()
12                .smaller()
13                .parenthesize()
14                .override(('padding', 0.5))
15            ),
16        ),
17     selector=selectortools.select_pitched_runs()[-1],
18 )

```

## B.2.16 ZAIRA.MATERIALS.LEGATO\_RHYTHM MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 legato_rhythm_maker = rhythmmakertools.TaleaRhythmMaker(
6     talea=rhythmmakertools.Talea(
7         counts=(3, 2, 1, 1, 1, 4, 3, 1, 1, 1, 1, 5, 2, 4),
8         denominator=8,
9         ),
10    # split_divisions_by_counts=(4, 3, 6, 5),
11    extra_counts_per_division=(0, 0, 1, 2, 0, 1),
12    beamSpecifier=rhythmmakertools.BeamSpecifier(
13        beam_each_division=False,

```

```

14     beam_divisions_together=False,
15     ),
16 #   burnishSpecifier=rhythmmakertools.BurnishSpecifier(
17 #     outer_divisions_only=True,
18 #     lefts=(-1, 0),
19 #     left_lengths=(1,),
20 #     right_lengths=(0,),
21 #     ),
22     tie_split_notes=False,
23     tuplet_spellingSpecifier=rhythmmakertools.TupletSpellingSpecifier(
24       avoid_dots=True,
25       is_diminution=True,
26       simplify_tuplets=True,
27     ),
28   )

```

## B.2.17 ZAIRA.MATERIALS.LEGATO\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 legato_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=rhythmmakertools.Talea(
8         counts=(0, 2, 1),
9         denominator=8,
10        ),
11     playing_talea=rhythmmakertools.Talea(
12         counts=(4, 5, 4, 3, 7, 6),
13         denominator=8,
14        ),
15     playing_groupings=(3, 4, 2, 2, 3),
16     repeat=True,
17     silence_talea=rhythmmakertools.Talea(
18         counts=(2, 1, 2, 1, 3, 7, 1, 2),
19         denominator=8,
20        ),
21     step_anchor=Right,
22     synchronize_groupings=False,
23     synchronize_step=False,
24     timespanSpecifier=consort.TimespanSpecifier(
25         minimum_duration=durationtools.Duration(1, 8),
26        ),
27    )

```

## B.2.18 ZAIRA.MATERIALS.METAL\_AGGITATION\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import pitchtools
4 from abjad.tools import selectortools
5 from abjad.tools.topleveltools import new
6 import consort

```

```

7 import zaira
8
9
10 metal_agitation_musicSpecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.foreground_dynamic_attachment_expression,
13         accent=consort.AttachmentExpression(
14             attachments=indicatortools.Articulation('accent'),
15             selector=selectortools.select_first_logical_tie_in_pitched_runs()[0],
16         ),
17         staccato=consort.AttachmentExpression(
18             attachments=indicatortools.Articulation('staccato'),
19             selector=selectortools.select_all_but_first_logical_tie_in_pitched_runs()[0],
20         ),
21     ),
22     pitch_handler=consort.AbsolutePitchHandler(
23         pitch_specifier=pitchtools.PitchSegment(
24             items=(
25                 zaira.makers.Percussion.HIGH_CYMBAL,
26                 zaira.makers.Percussion.LOW_CYMBAL,
27                 zaira.makers.Percussion.MIDDLE_CYMBAL,
28                 zaira.makers.Percussion.HIGH_CYMBAL,
29                 zaira.makers.Percussion.MIDDLE_CYMBAL,
30                 zaira.makers.Percussion.LOW_CYMBAL,
31                 zaira.makers.Percussion.HIGH_CYMBAL,
32                 zaira.makers.Percussion.TAM_TAM,
33                 zaira.makers.Percussion.MIDDLE_CYMBAL,
34                 zaira.makers.Percussion.LOW_CYMBAL,
35                 zaira.makers.Percussion.HIGH_CYMBAL,
36                 zaira.makers.Percussion.TAM_TAM,
37                 zaira.makers.Percussion.LOW_CYMBAL,
38                 zaira.makers.Percussion.MIDDLE_CYMBAL,
39             ),
40         ),
41     ),
42     rhythm_maker=new(
43         zaira.materials.stuttering_rhythm_maker,
44         extra_counts_per_division=(2, 1, 2, 1, 0, 2, 1, 0),
45     ),
46 )

```

## B.2.19 ZAIRA.MATERIALS.METAL\_BRUSHED\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import pitchtools
4 from abjad.tools import spannertools
5 from abjad.tools import selectortools
6 import consort
7 import zaira
8
9
10 metal_brushed_musicSpecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(

```

```

12     dynamic_expression=zaira.materials.midground_dynamic_attachment_expression,
13     text_spinner=consort.AttachmentExpression(
14         attachments=consort.ComplexTextSpanner(
15             markup=Markup(r'\concat {\vstrut brush}')
16             .italic()
17             .pad_around(0.5)
18             .box(),
19         ),
20         selector=selectortools.Selector().by_leaves(),
21     ),
22     stem_tremolo_spinner=consort.AttachmentExpression(
23         attachments=(
24             spannertools.StemTremoloSpanner(),
25             None,
26         ),
27         selector=selectortools.select_pitched_runs(),
28     ),
29     ),
30     pitch_handler=consort.AbsolutePitchHandler(
31         pitchSpecifier=pitchtools.PitchSegment(
32             items=(
33                 zaira.makers.Percussion.HIGH_CYMBAL,
34                 zaira.makers.Percussion.LOW_CYMBAL,
35                 zaira.makers.Percussion.MIDDLE_CYMBAL,
36                 zaira.makers.Percussion.HIGH_CYMBAL,
37                 zaira.makers.Percussion.MIDDLE_CYMBAL,
38                 zaira.makers.Percussion.LOW_CYMBAL,
39                 zaira.makers.Percussion.HIGH_CYMBAL,
40                 zaira.makers.Percussion.TAM_TAM,
41                 zaira.makers.Percussion.MIDDLE_CYMBAL,
42                 zaira.makers.Percussion.LOW_CYMBAL,
43                 zaira.makers.Percussion.HIGH_CYMBAL,
44                 zaira.makers.Percussion.TAM_TAM,
45                 zaira.makers.Percussion.LOW_CYMBAL,
46                 zaira.makers.Percussion.MIDDLE_CYMBAL,
47             ),
48         ),
49     ),
50     rhythm_maker=zaira.materials.sustained_rhythm_maker,
51 )

```

## B.2.20 ZAIRA.MATERIALS.METAL\_TRANQUILO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import pitchtools
4 import consort
5 import zaira
6
7
8 metal_tranquilo_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10        dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
11        laissez_vibrer=zaira.materials.laissez_vibrer_attachment_expression,

```

```

12     ),
13     pitch_handler=consort.AbsolutePitchHandler(
14         pitchSpecifier=pitchtools.PitchSegment(
15             items=(
16                 zaira.makers.Percussion.HIGH_CYMBAL,
17                 zaira.makers.Percussion.LOW_CYMBAL,
18                 zaira.makers.Percussion.MIDDLE_CYMBAL,
19                 zaira.makers.Percussion.HIGH_CYMBAL,
20                 zaira.makers.Percussion.MIDDLE_CYMBAL,
21                 zaira.makers.Percussion.LOW_CYMBAL,
22                 zaira.makers.Percussion.HIGH_CYMBAL,
23                 zaira.makers.Percussion.TAM_TAM,
24                 zaira.makers.Percussion.MIDDLE_CYMBAL,
25                 zaira.makers.Percussion.LOW_CYMBAL,
26                 zaira.makers.Percussion.HIGH_CYMBAL,
27                 zaira.makers.Percussion.TAM_TAM,
28                 zaira.makers.Percussion.LOW_CYMBAL,
29                 zaira.makers.Percussion.MIDDLE_CYMBAL,
30             ),
31         ),
32     ),
33     rhythm_maker=new(
34         zaira.materials.stuttering_rhythm_maker,
35         extra_counts_per_division=None,
36         inciseSpecifier__prefix_talea=(1,),
37         inciseSpecifier__prefix_counts=(1,),
38         inciseSpecifier__talea_denominator=16,
39     ),
40 )

```

## B.2.21 ZAIRA.MATERIALS.MIDGROUND\_DYNAMIC\_ATTACHMENT\_EXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 midground_dynamic_attachment_expression = consort.AttachmentExpression(
7     attachments=(
8         consort.SimpleDynamicExpression('mf'),
9         consort.SimpleDynamicExpression('mp'),
10    ),
11    selector=selectortools.select_pitched_runs()[0],
12 )

```

## B.2.22 ZAIRA.MATERIALS.OBOE\_SOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import spannertools
4 from abjad.tools import selectortools
5 from abjad.tools.topleveltools import new
6 import consort
7 import zaira

```

```

8
9
10 oboe_solo_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
13         trill_spinner=consort.AttachmentExpression(
14             attachments=(
15                 spannertools.ComplexTrillSpanner(interval='+m3'),
16                 spannertools.ComplexTrillSpanner(interval='+m3'),
17                 spannertools.ComplexTrillSpanner(interval='+M2'),
18             ),
19             selector=selectortools.Selector(
20                 .by_logical_tie(pitched=True
21                 .by_duration('>', (1, 32))
22             ),
23             staccato=consort.AttachmentExpression(
24                 attachments=indicatortools.Articulation('staccato'),
25                 selector=selectortools.Selector(
26                     .by_logical_tie(pitched=True
27                     .by_duration('<', (1, 16)
28                     .by_length(1)
29             ),
30             accent=consort.AttachmentExpression(
31                 attachments(
32                     indicatortools.Articulation('accent'),
33                     ),
34                     selector=selectortools.Selector(
35                         .by_logical_tie(pitched=True)[0],
36                     ),
37             ),
38             pitch_handler=consort.AbsolutePitchHandler(
39                 pitchSpecifier="d'' d'' ef'' d'' ef''' f''' d''' g''' d''' d''' as'''',
40                 pitch_application_rate='division',
41             ),
42             rhythm_maker=new(
43                 zaira.materials.reiterating_rhythm_maker,
44                 denominators=(32, 1, 32, 32, 1),
45             )
46         )

```

## B.2.23 ZAIRA.MATERIALS.OVERPRESSURE\_ATTACHMENT\_EXPRESSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import selectortools
3 import consort
4
5
6 overpressure_attachment_expression = consort.AttachmentExpression(
7     attachments=consort.ComplexTextSpanner(
8         markup=r"\filled-box #'(-0.5 . 1.5) #'(-1 . 1) #0.25",
9         ),
10    selector=selectortools.Selector(),
11    )

```

### B.2.24 ZAIRA.MATERIALS.PEDALS\_TIMESPAN MAKER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 pedals_timespan_maker = consort.DependentTimespanMaker(
7     labels=(
8         'pedaled',
9         ),
10    voice_names=(
11        'Piano Upper Voice',
12        'Piano Lower Voice',
13        ),
14    )
```

### B.2.25 ZAIRA.MATERIALS.PERCUSION\_BRUSHED\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import selectortools
4 import consort
5 import zaira
6
7
8 percussion_brushed_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
11         text_spanner=consort.AttachmentExpression(
12             attachments=consort.ComplexTextSpanner(
13                 markup=Markup(r'\concat { \vstrut brush }')
14                 .italic()
15                 .pad_around(0.5)
16                 .box(),
17             ),
18             selector=selectortools.Selector().by_leaves(),
19             ),
20         ),
21         pitch_handler=consort.AbsolutePitchHandler(
22             ),
23         rhythm_maker=zaira.materials.sustained_rhythm_maker,
24     )
```

### B.2.26 ZAIRA.MATERIALS.PERCUSION\_BRUSHED\_TREMOLO\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import spannertools
4 from abjad.tools import selectortools
5 import consort
6 import zaira
7
8
```

```

9 percussion_brushed_tremolo_music_specifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
12         text_spanner=consort.AttachmentExpression(
13             attachments=consort.ComplexTextSpanner(
14                 markup=Markup(r'\concat { \vstrut brush }')
15                 .italic()
16                 .pad_around(0.5)
17                 .box(),
18             ),
19             selector=selectortools.Selector().by_leaves(),
20         ),
21         stem_tremolo_spanner=consort.AttachmentExpression(
22             attachments=spannertools.StemTremoloSpanner(),
23             selector=selectortools.Selector().by_leaves(),
24         ),
25     ),
26     pitch_handler=consort.AbsolutePitchHandler(
27     ),
28     rhythm_maker=zaira.materials.sustained_rhythm_maker,
29 )

```

### B.2.27 ZAIRA.MATERIALS.PERCUSION\_FANFARE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import selectortools
4 import consort
5 import zaira
6
7
8 percussion_fanfare_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         dynamic_expression=consort.AttachmentExpression(
11             attachments=indicatortools.Dynamic('fff'),
12             selector=selectortools.select_pitched_runs()[0],
13         ),
14         accent=consort.AttachmentExpression(
15             attachments=indicatortools.Articulation('accent'),
16             selector=selectortools.Selector(
17                 ).by_leaves(
18                 ).by_logical_tie(pitched=True,
19             )[0],
20         ),
21     ),
22     pitch_handler=consort.AbsolutePitchHandler(),
23     rhythm_maker=zaira.materials.reiterating_rhythm_maker,
24 )

```

### B.2.28 ZAIRA.MATERIALS.PERCUSION\_REITERATION\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import selectortools

```

```

4 import consort
5 import zaira
6
7
8 percussion_reiteration_musicSpecifier = consort.MusicSpecifier(
9     attachmentHandler=consort.AttachmentHandler(
10        dynamicExpression=consort.AttachmentExpression(
11            attachments=indicatortools.Dynamic('ppp'),
12            selector=selectortools.select_pitched_runs()[0],
13            ),
14        staccato=consort.AttachmentExpression(
15            attachments=indicatortools.Articulation('.'),
16            selector=selectortools.Selector(
17                ).by_leaves(
18                ).by_logical_tie(pitched=True
19                )[0],
20            ),
21        ),
22    pitchHandler=consort.AbsolutePitchHandler(),
23    rhythmMaker=zaira.materials.reiterating_rhythmMaker,
24 )

```

### B.2.29 ZAIRA.MATERIALS.PERCUSION\_SUPERBALL\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import selectortools
4 import consort
5 import zaira
6
7
8 percussion_superball_musicSpecifier = consort.MusicSpecifier(
9     attachmentHandler=consort.AttachmentHandler(
10        dynamicExpression=zaira.materials.background_dynamic_attachment_expression,
11        textSpanner=consort.AttachmentExpression(
12            attachments=consort.ComplexTextSpanner(
13                markup=Markup(r'\concat { \vstrut superball }')
14                .italic()
15                .pad_around(0.5)
16                .box(),
17                ),
18            selector=selectortools.Selector().by_leaves(),
19            ),
20        ),
21    pitchHandler=consort.AbsolutePitchHandler(
22        ),
23    rhythmMaker=zaira.materials.sustained_rhythmMaker,
24 )

```

### B.2.30 ZAIRA.MATERIALS.PIANO\_CLUSTERS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import pitchtools
3 import consort

```

```

4 import zaira
5
6
7 piano_clusters_music_specifier = consort.MusicSpecifier(
8     attachment_handler=consort.AttachmentHandler(
9         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
10    ),
11    labels='pedaled',
12    pitch_handler=consort.PitchClassPitchHandler(
13        logical_tie_expressions=(
14            consort.KeyClusterExpression(
15                staff_space_width=7,
16            ),
17            consort.KeyClusterExpression(
18                staff_space_width=7,
19            ),
20            consort.KeyClusterExpression(
21                staff_space_width=9,
22            ),
23            consort.KeyClusterExpression(
24                include_black_keys=False,
25                staff_space_width=7,
26            ),
27            consort.KeyClusterExpression(
28                staff_space_width=7,
29            ),
30            consort.KeyClusterExpression(
31                include_white_keys=False,
32                staff_space_width=9,
33            ),
34            consort.KeyClusterExpression(
35                staff_space_width=9,
36            ),
37        ),
38        pitchSpecifier="c g e a",
39        pitchRange=pitchtools.PitchRange('[A1, C7)'),
40        registerSpecifier=zaira.materials.registerSpecifierInventory[0],
41    ),
42    rhythm_maker=zaira.materials.stuttering_rhythm_maker,
43 )

```

### B.2.31 ZAIRA.MATERIALS.PIANO\_DRONE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import spannertools
3 from abjad.tools import selectortools
4 import consort
5 import zaira
6
7
8 piano_drone_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10        dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
11        stem_tremolo_spanner=consort.AttachmentExpression(

```

```

12     attachments=spannertools.StemTremoloSpanner(),
13     selector=selectortools.select_pitched_runs(),
14     ),
15   ),
16   labels='pedaled',
17   pitch_handler=consort.AbsolutePitchHandler(
18     logical_tie_expressions=(
19       consort.ChordExpression(
20         chord_expr=(-7, -3, 0, 5, 6, 12),
21         ),
22       consort.ChordExpression(
23         chord_expr=(-7, -3, 0, 1, 5, 12),
24         ),
25     ),
26     pitchSpecifier="d",
27   ),
28 )

```

### B.2.32 ZAIRA.MATERIALS.PIANO\_FANFARE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import selectortools
4 from abjad.tools import pitchtools
5 import consort
6 import zaira
7
8
9 piano_fanfare_music_specifier = consort.MusicSpecifier(
10   attachment_handler=consort.AttachmentHandler(
11     dynamic_expression=consort.AttachmentExpression(
12       attachments=indicatortools.Dynamic('fff'),
13       selector=selectortools.select_pitched_runs()[0],
14     ),
15     accent=consort.AttachmentExpression(
16       attachments=indicatortools.Articulation('accent'),
17       selector=selectortools.Selector(
18         ).by_leaves(
19           ).by_logical_tie(pitched=True,
20             )[0],
21         ),
22       ),
23   labels='pedaled',
24   pitch_handler=consort.PitchClassPitchHandler(
25     logical_tie_expressions=(
26       consort.KeyClusterExpression(
27         include_black_keys=False,
28         staff_space_width=7,
29       ),
30       consort.KeyClusterExpression(
31         staff_space_width=9,
32       ),
33       consort.KeyClusterExpression(
34         staff_space_width=7,

```

```

35         ),
36         consort.KeyClusterExpression(
37             include_black_keys=False,
38             staff_space_width=7,
39         ),
40         consort.KeyClusterExpression(
41             include_black_keys=False,
42             staff_space_width=9,
43         ),
44         consort.KeyClusterExpression(
45             staff_space_width=7,
46         ),
47         consort.KeyClusterExpression(
48             include_white_keys=False,
49             staff_space_width=9,
50         ),
51         consort.KeyClusterExpression(
52             include_black_keys=False,
53             staff_space_width=7,
54         ),
55         consort.KeyClusterExpression(
56             staff_space_width=7,
57         ),
58         consort.KeyClusterExpression(
59             include_white_keys=False,
60             staff_space_width=9,
61         ),
62     ),
63     pitchSpecifier="c a f d e b",
64     pitchRange=pitchtools.PitchRange('A1, C7'),
65     registerSpecifier=consort.RegisterSpecifier(),
66 ),
67 rhythm_maker=zaira.materials.reiterating_rhythm_maker,
68 )

```

### B.2.33 ZAIRA.MATERIALS.PIANO\_FLOURISH\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import pitchtools
3 from abjad.tools import spannertools
4 from abjad.tools import selectortools
5 import consort
6 import zaira
7
8
9 piano_flourish_music_specifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
12         slur=consort.AttachmentExpression(
13             attachments=spannertools.Slur(),
14             selector=selectortools.select_pitched_runs(),
15         ),
16     ),
17     labels='pedaled',

```

```

18     pitch_handler=consort.PitchClassPitchHandler(
19         pitchSpecifier='d f e g cs as',
20         pitchRange=pitchtools.PitchRange('A1, C7'),
21         registerSpecifier=zaira.materials.registerSpecifierInventory[2],
22         ),
23     rhythm_maker=zaira.materials.flourish_rhythm_maker,
24 )

```

### B.2.34 ZAIRA.MATERIALS.PIANO\_GUERO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import selectortools
4 from abjad.tools import spannertools
5 import consort
6 import zaira
7
8
9 slow_markup = Markup(r'\concat { \vstrut slow }', Up)
10 slow_markup = slow_markup.italic()
11 slow_markup = slow_markup.pad_around(0.5)
12 slow_markup = slow_markup.box()
13
14 fast_markup = Markup(r'\concat { \vstrut fast }', Up)
15 fast_markup = fast_markup.italic()
16 fast_markup = fast_markup.pad_around(0.5)
17 fast_markup = fast_markup.box()
18
19 piano_guero_musicSpecifier = consort.MusicSpecifier(
20     attachmentHandler=consort.AttachmentHandler(
21         clefSpanner=consort.ClefSpanner('percussion'),
22         dynamicExpression=zaira.materials.midground_dynamic_attachment_expression,
23         staffLinesSpanner=spannertools.StaffLinesSpanner(
24             lines=(4, -4),
25             overrides={
26                 'note_head__no_ledgers': True,
27                 'note_head__style': 'cross',
28             }
29         ),
30         textSpanner=consort.AttachmentExpression(
31             attachments=consort.ComplexTextSpanner(
32                 markup=Markup(r'\concat { \vstrut guero }')
33                 .italic()
34                 .pad_around(0.5)
35                 .box(),
36             ),
37             selector=selectortools.Selector().by_leaves(),
38         ),
39         directionMarkup=consort.AttachmentExpression(
40             attachments=(
41                 slow_markup,
42                 slow_markup,
43                 fast_markup,
44                 slow_markup,

```

```

45         fast_markup,
46         ),
47         selector=selectortools.select_pitched_runs()[0],
48         ),
49     ),
50     pitch_handler=consort.AbsolutePitchHandler(),
51     rhythm_maker=zaira.materials.sustained_rhythm_maker,
52 )

```

### B.2.35 ZAIRA.MATERIALS.PIANO\_PEDALS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import rhythmmakertools
3 from abjad.tools import selectortools
4 import consort
5
6
7 piano_pedals_music_specifier = consort.MusicSpecifier(
8     attachment_handler=consort.AttachmentHandler(
9         piano_pedal_spinner=consort.AttachmentExpression(
10            attachments=(
11                consort.ComplexPianoPedalSpinner(),
12                ),
13                selector=selectortools.Selector(),
14                ),
15            ),
16            rhythm_maker=rhythmmakertools.SkipRhythmMaker(
17                duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
18                    forbid_meter_rewriting=True,
19                    ),
20                ),
21            )
22 )

```

### B.2.36 ZAIRA.MATERIALS.PIANO\_PREPARED\_BASS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import pitchtools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 import consort
8 import zaira
9
10
11 staccato_selector = selectortools.Selector()
12 staccato_selector = staccato_selector.by_leaves()
13 staccato_selector = staccato_selector.by_logical_tie(
14     pitched=True,
15     )
16 staccato_selector = staccato_selector.by_duration('<=', (1, 16))
17
18 sustain_selector = selectortools.Selector()
19 sustain_selector = sustain_selector.by_leaves()

```

```

20 sustain_selector = sustain_selector.by_logical_tie(
21     pitched=True,
22 )
23 sustain_selector = sustain_selector.by_duration('>', (1, 16))
24
25
26 piano_prepared_bass_music_specifier = consort.MusicSpecifier(
27     attachment_handler=consort.AttachmentHandler(
28         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
29         clef_spacer=consort.AttachmentExpression(
30             attachments=consort.ClefSpanner(
31                 clef='bass_15',
32             ),
33             selector=selectortools.Selector(),
34         ),
35         text_spacer=consort.AttachmentExpression(
36             attachments=consort.ComplexTextSpanner(
37                 markup=Markup(r'\concat { \vstrut prepared }')
38                 .italic()
39                 .pad_around(0.5)
40                 .box(),
41             overrides={
42                 'note_head__style': 'cross',
43             },
44         ),
45         selector=selectortools.Selector(),
46     ),
47     staccato=consort.AttachmentExpression(
48         attachments=indicatortools.Articulation('.'),
49         selector=staccato_selector,
50     ),
51     trill_spacer=consort.AttachmentExpression(
52         attachments=spannertools.ComplexTrillSpanner(
53             interval='m2',
54         ),
55         selector=sustain_selector,
56     ),
57     ),
58     labels='pedaled',
59     pitch_handler=consort.AbsolutePitchHandler(
60         pitch_specifier=pitchtools.PitchSegment(
61             'A0 C1 B0 D1 C#1 E1 D#1 F1 G1 A#0 F#1',
62         ),
63     ),
64     rhythm_maker=zaira.materials.undergrowth_rhythm_maker,
65 )

```

### B.2.37 ZAIRA.MATERIALS.PIANO\_PREPARED\_TREBLE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import pitchtools
5 from abjad.tools import selectortools

```

```

6 from abjad.tools import spannertools
7 import consort
8 import zaira
9
10
11 staccato_selector = selectortools.Selector()
12 staccato_selector = staccato_selector.by_leaves()
13 staccato_selector = staccato_selector.by_logical_tie(
14     pitched=True,
15 )
16 staccato_selector = staccato_selector.by_duration('<=', (1, 16))
17
18
19 sustain_selector = selectortools.Selector()
20 sustain_selector = sustain_selector.by_leaves()
21 sustain_selector = sustain_selector.by_logical_tie(
22     pitched=True,
23 )
24 sustain_selector = sustain_selector.by_duration('>', (1, 16))
25
26
27 piano_prepared_treble_music_specifier = consort.MusicSpecifier(
28     attachment_handler=consort.AttachmentHandler(
29         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
30         clef_spanner=consort.AttachmentExpression(
31             attachments=consort.ClefSpanner(
32                 clef='treble^15',
33             ),
34             selector=selectortools.Selector(),
35         ),
36         text_spanner=consort.AttachmentExpression(
37             attachments=consort.ComplexTextSpanner(
38                 markup=Markup(r'\concat { \vstrut prepared }')
39                 .italic()
40                 .pad_around(0.5)
41                 .box(),
42                 overrides={
43                     'note_head__style': 'cross',
44                 },
45             ),
46             selector=selectortools.Selector(),
47         ),
48         staccato=consort.AttachmentExpression(
49             attachments=indicatortools.Articulation('.'),
50             selector=staccato_selector,
51         ),
52         trill_spanner=consort.AttachmentExpression(
53             attachments=spannertools.ComplexTrillSpanner(
54                 interval='m2',
55             ),
56             selector=sustain_selector,
57         ),
58     ),
59     labels='pedaled',

```

```

60     pitch_handler=consort.AbsolutePitchHandler(
61         pitchSpecifier=pitchtools.PitchSegment(
62             'C7 B7 D7 C#7 E7 D#7 F7 G7 A#7 F#7 A7',
63             ),
64         ),
65     rhythm_maker=zaira.materials.undergrowth_rhythm_maker,
66 )

```

### B.2.38 ZAIRA.MATERIALS.PROPORTIONS

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 proportions = consortium.Proportions(
7     [
8         [2],          # A
9         [1, 6, 3],   # B
10        [7],         # C
11        [1, 2, 1],   # D
12        [1, 8, 4],   # E
13        [1, 14, 7],  # F
14        [1, 4, 2],   # G
15        [5],         # H
16        [1, 15],    # I
17        [2],         # J
18        [3],         # K
19    ]
20 )

```

### B.2.39 ZAIRA.MATERIALS.REGISTER\_SPECIFIER\_INVENTORY

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import mathtools
3 import consort
4
5
6 registerSpecifierInventory = consortium.RegisterSpecifierInventory(
7     [
8         consortium.RegisterSpecifier(
9             divisionInflections=consort.RegisterInflectionInventory([
10                 consortium.RegisterInflection(
11                     inflections=(-3, 3),
12                     ratio=mathtools.Ratio([1]),
13                     ),
14                 consortium.RegisterInflection(
15                     inflections=(3, 6, -3),
16                     ratio=mathtools.Ratio([2, 1]),
17                     ),
18                 consortium.RegisterInflection(
19                     inflections=(-6, 0, 6),
20                     ratio=mathtools.Ratio([1, 2]),
21                     ),
22             )
23         )
24     ]
25 )

```

```

22     ]),
23     phrase_inflections=consort.RegisterInflectionInventory([
24         consortium.RegisterInflection(
25             inflections=(6, 0, -6),
26             ratio=mathtools.Ratio([2, 1]),
27         ),
28         consortium.RegisterInflection(
29             inflections=(3, 0, -3),
30             ratio=mathtools.Ratio([1, 2]),
31         ),
32     ]),
33     segment_inflections=consort.RegisterInflectionInventory([
34         consortium.RegisterInflection(
35             inflections=(-12, 0, 12),
36             ratio=mathtools.Ratio([2, 1]),
37         ),
38     ]),
39     ),
40     consortium.RegisterSpecifier(
41         division_inflections=consort.RegisterInflectionInventory([
42             consortium.RegisterInflection(
43                 inflections=(-6, 6),
44                 ratio=mathtools.Ratio([1]),
45             ),
46             consortium.RegisterInflection(
47                 inflections=(-12, 3, 9),
48                 ratio=mathtools.Ratio([2, 1]),
49             ),
50             consortium.RegisterInflection(
51                 inflections=(6, -6),
52                 ratio=mathtools.Ratio([1]),
53             ),
54         ]),
55         phrase_inflections=consort.RegisterInflectionInventory([
56             consortium.RegisterInflection(
57                 inflections=(3, -3),
58                 ratio=mathtools.Ratio([1]),
59             ),
60             consortium.RegisterInflection(
61                 inflections=(-3, 3),
62                 ratio=mathtools.Ratio([1]),
63             ),
64         ]),
65         segment_inflections=consort.RegisterInflectionInventory([
66             consortium.RegisterInflection(
67                 inflections=(12, 6, 0, -12),
68                 ratio=mathtools.Ratio([3, 2, 1]),
69             ),
70         ]),
71     ),
72     consortium.RegisterSpecifier(
73         division_inflections=consort.RegisterInflectionInventory([
74             consortium.RegisterInflection(
75                 inflections=(3, -3),

```

```

76             ratio=mathtools.Ratio([1]),
77             ),
78         consort.RegisterInflection(
79             inflections=(3, -3),
80             ratio=mathtools.Ratio([1]),
81             ),
82         consort.RegisterInflection(
83             inflections=(-3, 3),
84             ratio=mathtools.Ratio([1]),
85             ),
86         ],
87     phrase_inflections=consort.RegisterInflectionInventory([
88         consortium.RegisterInflection(
89             inflections=(6, -6),
90             ratio=mathtools.Ratio([1]),
91             ),
92         consortium.RegisterInflection(
93             inflections=(-3, 3, 6),
94             ratio=mathtools.Ratio([2, 1]),
95             ),
96     ],
97     segment_inflections=consort.RegisterInflectionInventory([
98         consortium.RegisterInflection(
99             inflections=(-6, 6, -3),
100            ratio=mathtools.Ratio([2, 1]),
101            ),
102        ],
103    ),
104    consortium.RegisterSpecifier(
105        division_inflections=consort.RegisterInflectionInventory([
106            consortium.RegisterInflection(
107                inflections=(-3, 3),
108                ratio=mathtools.Ratio([1]),
109                ),
110            consortium.RegisterInflection(
111                inflections=(-3, 3),
112                ratio=mathtools.Ratio([1]),
113                ),
114            consortium.RegisterInflection(
115                inflections=(3, -3),
116                ratio=mathtools.Ratio([1]),
117                ),
118        ],
119        phrase_inflections=consort.RegisterInflectionInventory([
120            consortium.RegisterInflection(
121                inflections=(-3, 3),
122                ratio=mathtools.Ratio([1]),
123                ),
124            consortium.RegisterInflection(
125                inflections=(6, 3, -6),
126                ratio=mathtools.Ratio([2, 1]),
127                ),
128        ],
129        segment_inflections=consort.RegisterInflectionInventory([

```

```

130         consort.RegisterInflection(
131             inflections=(-12, 6, -6, 12),
132             ratio=mathtools.Ratio([2, 1, 3]),
133             ),
134         ],
135     ],
136 )
137 )

```

## B.2.40 ZAIRA.MATERIALS.REITERATING\_RHYTHM MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 reiterating_rhythm_maker = rhythmmakertools.EvenDivisionRhythmMaker(
6     denominators=(16,),
7     beamSpecifier=rhythmmakertools.BeamSpecifier(
8         beamEachDivision=False,
9         beamDivisionsTogether=False,
10        ),
11        durationSpellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
12            decreaseDurationsMonotonically=True,
13            forbiddenWrittenDuration=durationtools.Duration(1, 2),
14            ),
15        )

```

## B.2.41 ZAIRA.MATERIALS.SPARSE\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 sparseTimespanMaker = consort.TaleaTimespanMaker(
7     initialSilenceTalea=rhythmmakertools.Talea(
8         counts=(0, 4, 2, 6, 3),
9         denominator=16,
10        ),
11        playingTalea=rhythmmakertools.Talea(
12            counts=(4, 6, 8, 5, 6, 6, 4),
13            denominator=16,
14            ),
15        playingGroupings=(1, 1, 2, 1, 2, 1, 1, 1),
16        repeat=True,
17        silenceTalea=rhythmmakertools.Talea(
18            counts=(20, 12, 26, 10, 14, 7),
19            denominator=16,
20            ),
21        stepAnchor=Right,
22        synchronizeGroupings=False,
23        synchronizeStep=False,
24        timespanSpecifier=consort.TimespanSpecifier(
25            minimumDuration=durationtools.Duration(1, 8),

```

```
26     ),
27 )
```

### B.2.42 ZAIRA.MATERIALS.STRING\_CHORD\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 import zaira
4
5
6 string_chord_music_specifier = consort.MusicSpecifier(
7     attachment_handler=consort.AttachmentHandler(
8         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
9         )),
10    pitch_handler=consort.PitchClassPitchHandler(
11        pitch_specifier='c ef d',
12        register_specifier=consort.RegisterSpecifier(),
13        ),
14    rhythm_maker=zaira.materials.sustained_rhythm_maker,
15 )
```

### B.2.43 ZAIRA.MATERIALS.STRING\_FLOURISH\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import selectortools
4 from abjad.tools import spannertools
5 from abjad.tools.topleveltools import new
6 import consort
7 import zaira
8
9
10 string_flourish_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
13         slur=consort.AttachmentExpression(
14             attachments=spannertools.Slur(),
15             selector=selectortools.select_pitched_runs(),
16             ),
17         text_spanner=consort.AttachmentExpression(
18             attachments=consort.ComplexTextSpanner(
19                 markup=Markup(r'\concat { \vstrut flautando }')
20                 .italic()
21                 .pad_around(0.5)
22                 .box(),
23             ),
24             selector=selectortools.select_pitched_runs(),
25             ),
26             ),
27         pitch_handler=consort.PitchClassPitchHandler(
28             pitch_specifier='d f e g cs as',
29             register_specifier=zaira.materials.register_specifier_inventory[2],
30             register_spread=0,
31             ),
```

```

32     rhythm_maker=new(
33         zaira.materials.flourish_rhythm_maker,
34         tieSpecifier__tie_across_divisions=False,
35     ),
36 )

```

## B.2.44 ZAIRA.MATERIALS.STRING\_TRILLS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import markuptools
3 from abjad.tools import schemetools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 import consort
7 import zaira
8
9
10 fourth_spanner = spannertools.ComplexTrillSpanner(
11     interval='+P4',
12     overrides={
13         'trill_pitch_head__stencil': schemetools.Scheme(
14             'ly:text-interface::print',
15         ),
16         'trill_pitch_head__text': markuptools.Markup.musicglyph(
17             'noteheads.s0harmonic',
18             direction=None,
19         ),
20     },
21 )
22
23 third_spanner = spannertools.ComplexTrillSpanner(
24     interval='+m3',
25     overrides={
26         'trill_pitch_head__stencil': schemetools.Scheme(
27             'ly:text-interface::print',
28         ),
29         'trill_pitch_head__text': markuptools.Markup.musicglyph(
30             'noteheads.s0harmonic',
31             direction=None,
32         ),
33     },
34 )
35
36 string_trills_music_specifier = consort.MusicSpecifier(
37     attachment_handler=consort.AttachmentHandler(
38         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
39         trill_spanner=consort.AttachmentExpression(
40             attachments=(
41                 third_spanner,
42                 fourth_spanner,
43                 fourth_spanner,
44             ),
45             selector=selectortools.select_pitched_runs(),
46         ),

```

```

47     ),
48     pitch_handler=consort.PitchClassPitchHandler(
49         pitchSpecifier='a c b',
50         registerSpecifier=zaira.materials.registerSpecifierInventory[3],
51     ),
52     rhythm_maker=zaira.materials.sustainedRhythmMaker,
53 )

```

## B.2.45 ZAIRA.MATERIALS.STRING\_TUTTI\_OVERPRESSURE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import selectortools
5 import consort
6 import zaira
7
8
9 string_tutti_overpressure_music_specifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         text_spanner=consort.AttachmentExpression(
12             attachments=consort.ComplexTextSpanner(
13                 markup=Markup(r'\concat {\ \vstrut overpressure }')
14                     .italic()
15                     .pad_around(0.5)
16                     .box(),
17             ),
18             selector=selectortools.Selector().by_leaves(),
19         ),
20         dynamic_and_accent=consort.AttachmentExpression(
21             attachments=(
22                 (
23                     indicatortools.Dynamic('fff'),
24                     indicatortools.Articulation('accent'),
25                     indicatortools.Articulation('tenuto'),
26                 ),
27             ),
28             selector=selectortools.Selector()
29                 .by_leaves()
30                 .by_logical_tie(pitched=True)[0],
31             ),
32         ),
33     pitch_handler=consort.AbsolutePitchHandler(
34         logical_tie_expressions=(
35             consortium.ChordExpression(
36                 chord_expr=(0, 7),
37             ),
38         ),
39     ),
40     rhythm_maker=zaira.materials.sustainedRhythmMaker,
41 )

```

## B.2.46 ZAIRA.MATERIALS.STRING\_UNDERGROWTH\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 import consort
7 import zaira
8
9
10 string_undergrowth_musicSpecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.midground_dynamic_attachment_expression,
13         text_spanner=consort.AttachmentExpression(
14             attachments=consort.ComplexTextSpanner(
15                 markup=Markup(r'\concat {\vstrut overpressure }')
16                 .italic()
17                 .pad_around(0.5)
18                 .box(),
19             ),
20             selector=selectortools.Selector().by_leaves(),
21         ),
22         accent_and_tenuto=consort.AttachmentExpression(
23             attachments=(
24                 (
25                     indicatortools.Articulation('accent'),
26                     indicatortools.Articulation('tenuto'),
27                 ),
28             ),
29             selector=selectortools.Selector()
30             .by_leaves()
31             .by_logical_tie(pitched=True)[0],
32         ),
33         stem_tremolo_spanner=consort.AttachmentExpression(
34             attachments=(
35                 None,
36                 None,
37                 spannertools.StemTremoloSpanner(),
38             ),
39             selector=selectortools.Selector()
40             .by_leaves()
41             .by_logical_tie(pitched=True)
42         ),
43     ),
44     pitch_handler=consort.PitchClassPitchHandler(
45         pitchSpecifier='a c b d',
46         registerSpecifier=zaira.materials.registerSpecifierInventory[3],
47     ),
48     rhythm_maker=zaira.materials.undergrowth_rhythm_maker,
49 )
```

### B.2.47 ZAIRA.MATERIALS.STUTTERING\_RHYTHM MAKER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 stuttering_rhythm_maker = rhythmmakertools.IncisedRhythmMaker(
6     inciseSpecifier=rhythmmakertools.InciseSpecifier(
7         outerDivisionsOnly=False,
8         prefixTalea=(1, 1, 1, 2, 1, 2),
9         prefixCounts=(2, 2, 1, 2, 3, 2, 2, 2, 1),
10        suffixTalea=(1,),
11        suffixCounts=(0,),
12        taleaDenominator=16,
13        fillWithNotes=False,
14        ),
15        extraCountsPerDivision=(0, 0, 0, 1, 0, 1),
16        beamSpecifier=rhythmmakertools.BeamSpecifier(
17            beamEachDivision=False,
18            beamDivisionsTogether=False,
19            ),
20        tupletSpellingSpecifier=rhythmmakertools.TupletSpellingSpecifier(
21            avoidDots=True,
22            isDiminution=True,
23            simplifyTuplets=True,
24            ),
25    )
```

### B.2.48 ZAIRA.MATERIALS.SUSTAINED\_RHYTHM MAKER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 sustained_rhythm_maker = rhythmmakertools.NoteRhythmMaker(
6     beamSpecifier=rhythmmakertools.BeamSpecifier(
7         beamEachDivision=False,
8         beamDivisionsTogether=False,
9         ),
10        tieSpecifier=rhythmmakertools.TieSpecifier(
11            tieAcrossDivisions=(True, True, False),
12            ),
13    )
```

### B.2.49 ZAIRA.MATERIALS.SUSTAINED\_TIMESPAN MAKER

```
1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 sustainedTimespanMaker = consort.TaleaTimespanMaker(
7     initialSilenceTalea=rhythmmakertools.Talea(
8         counts=(0, 2, 1),
```

```

9     denominator=8,
10    ),
11    playing_talea=rhythmmakertools.Talea(
12      counts=(4, 5, 4, 3, 7, 6),
13      denominator=8,
14    ),
15    playing_groupings=(3, 4, 2, 2, 3),
16    repeat=True,
17    silence_talea=rhythmmakertools.Talea(
18      counts=(2, 1, 2, 1, 3, 7, 1, 2),
19      denominator=8,
20    ),
21    step_anchor=Right,
22    synchronize_groupings=False,
23    synchronize_step=False,
24    timespanSpecifier=consort.TimespanSpecifier(
25      minimum_duration=durationtools.Duration(1, 8),
26    ),
27  )

```

## B.2.50 ZAIRA.MATERIALS.TEMPI

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 tempi = indicatortools.TempoInventory(
6   [
7     indicatortools.Tempo(
8       duration=durationtools.Duration(1, 4),
9       units_per_minute=72,
10      ),
11     indicatortools.Tempo(
12       duration=durationtools.Duration(1, 4),
13       units_per_minute=48,
14      ),
15     indicatortools.Tempo(
16       duration=durationtools.Duration(1, 4),
17       units_per_minute=108,
18      ),
19   ]
20 )

```

## B.2.51 ZAIRA.MATERIALS.TIME\_SIGNATURES

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3
4
5 time_signatures = indicatortools.TimeSignatureInventory(
6   [
7     (2, 4),
8     (3, 4),
9     (3, 8),

```

```

10      (4, 4),
11      (5, 16),
12      (5, 8),
13      (6, 8),
14  ]
15 )

```

### B.2.52 ZAIRA.MATERIALS.TOTAL\_DURATION\_IN\_SECONDS

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 total_duration_in_seconds = 480

```

### B.2.53 ZAIRA.MATERIALS.TUTTI\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3 import consort
4
5
6 tutti_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=None,
8     playing_talea=rhythmmakertools.Talea(
9         counts=(4, 6, 6, 5),
10        denominator=16,
11        ),
12     playing_groupings=(1, 2, 1, 3, 2, 2),
13     repeat=True,
14     silence_talea=rhythmmakertools.Talea(
15         counts=(8, 6, 10, 7, 12),
16         denominator=8,
17         ),
18     step_anchor=Right,
19     synchronize_groupings=True,
20     synchronize_step=True,
21     timespanSpecifier=consort.TimespanSpecifier(
22         minimum_duration=durationaltools.Duration(1, 8),
23         ),
24     )

```

### B.2.54 ZAIRA.MATERIALS.UNDERGROWTH\_RHYTHM MAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 undergrowth_rhythm_maker = rhythmmakertools.TaleaRhythmMaker(
6     talea=rhythmmakertools.Talea(
7         counts=(1, -3, 1, -2, 1, -2, 2, -4, 4),
8         denominator=16,
9         ),
10    extra_counts_per_division=(0, 2, 1, 0, 1, 1, 0),

```

```

11     beamSpecifier=rhythmmakertools.BeamSpecifier(
12         beamEachDivision=False,
13         beamDivisionsTogether=False,
14     ),
15     tieSplitNotes=False,
16     tupletSpellingSpecifier=rhythmmakertools.TupletSpellingSpecifier(
17         avoidDots=True,
18         isDiminution=True,
19         simplifyTuplets=True,
20     ),
21 )

```

## B.2.55 ZAIRA.MATERIALS.WIND\_AIRTONE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import selectortools
4 import consort
5 import zaira
6
7
8 windAirtoneMusicSpecifier = consort.MusicSpecifier(
9     attachmentHandler=consort.AttachmentHandler(
10         dynamicExpression=zaira.materials.backgroundDynamicAttachmentExpression,
11         textSpanner=consort.AttachmentExpression(
12             attachments=consort.ComplexTextSpanner(
13                 markup=Markup(r'\concat {\vstrut airtone}')
14                 .italic()
15                 .padAround(0.5)
16                 .box(),
17                 overrides={
18                     'noteHeadStyle': 'slash',
19                 }
20             ),
21             selector=selectortools.Selector().byLeaves(),
22         ),
23     ),
24     pitchHandler=consort.PitchClassPitchHandler(
25         pitchSpecifier='a c b d b f gs e',
26         registerSpecifier=consort.RegisterSpecifier(),
27         registerSpread=0,
28     ),
29     rhythmMaker=zaira.materials.sustainedRhythmMaker,
30 )

```

## B.2.56 ZAIRA.MATERIALS.WIND\_KEYCLICK\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import spannertools
5 from abjad.tools import selectortools
6 import consort
7 import zaira

```

```

8
9
10 wind_keyclick_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
13         clef_spacer=consort.ClefSpanner('percussion'),
14         staff_lines_spacer=spannertools.StaffLinesSpanner(
15             lines=(4, -4),
16             overrides={
17                 'note_head__no_ledgers': True,
18                 'note_head__style': 'cross',
19             }
20         ),
21         text_spacer=consort.AttachmentExpression(
22             attachments=consort.ComplexTextSpanner(
23                 markup=Markup(r'\concat { \vstrut keyclick }')
24                 .italic()
25                 .pad_around(0.5)
26                 .box(),
27             ),
28             selector=selectortools.Selector().by_leaves(),
29         ),
30         stem_tremolo_spacer=consort.AttachmentExpression(
31             attachments=spannertools.StemTremoloSpanner(),
32             selector=selectortools.Selector(
33                 .by_logical_tie(pitched=True
34                 .by_duration('>', (1, 16))
35             ),
36             staccato=consort.AttachmentExpression(
37                 attachments=indicatortools.Articulation('.'),
38                 selector=selectortools.Selector(
39                     .by_logical_tie(pitched=True
40                     .by_duration('<', (1, 8)
41                     .by_length(1)
42                 ),
43             ),
44             pitch_handler=consort.AbsolutePitchHandler(
45                 pitchSpecifier="c' g' f g' g' c' f c' f g' c' c' f g''",
46                 pitches_are_nonsemantic=True,
47             ),
48             rhythm_maker=zaira.materials.undergrowth_rhythm_maker,
49         )

```

## B.2.57 ZAIRA.MATERIALS.WIND\_SLAP\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import Markup
3 from abjad.tools import indicatortools
4 from abjad.tools import markuptools
5 from abjad.tools import schemetools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8 import consort
9 import zaira

```

```

10
11
12 wind_slap_music_specifier = consort.MusicSpecifier(
13     attachment_handler=consort.AttachmentHandler(
14         dynamic_expression=zaira.materials.midground_dynamic_attachment_expression,
15         override_spacer=consort.AttachmentExpression(
16             attachments=spannertools.Spanner(
17                 overrides={
18                     'note_head__stencil': schemetools.Scheme(
19                         'ly:text-interface::print'
20                         ),
21                     'note_head__text': markuptools.Markup.triangle(False)
22                         .rotate(180)
23                         .scale((0.667, 0.667))
24                         .override(('thickness', 0.5))
25                         .translate((0, -0.9)),
26                     },
27                 ),
28             selector=selectortools.select_pitched_runs(),
29             ),
30             text_spacer=consort.AttachmentExpression(
31                 attachments=consort.ComplexTextSpanner(
32                     markup=Markup(r'\concat { \vstrut slap }')
33                     .italic()
34                     .pad_around(0.5)
35                     .box(),
36                 ),
37                 selector=selectortools.Selector().by_leaves(),
38                 ),
39             accent_and_staccato=consort.AttachmentExpression(
40                 attachments=(
41                     (
42                         indicatortools.Articulation('accent'),
43                         indicatortools.Articulation('staccato'),
44                     ),
45                     ),
46                 selector=selectortools.Selector()
47                     .by_leaves()
48                     .by_logical_tie(pitched=True)
49                     [0],
50                 ),
51             ),
52             pitch_handler=consort.PitchClassPitchHandler(
53                 pitchSpecifier='a c b d b f gs e d f cs',
54                 registerSpecifier=consort.RegisterSpecifier(),
55                 registerSpread=0,
56             ),
57             rhythm_maker=zaira.materials.stuttering_rhythm_maker,
58         )

```

## B.2.58 ZAIRA.MATERIALS.WIND\_TRILLS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import selectortools

```

```

3 from abjad.tools import spannertools
4 import consort
5 import zaira
6
7
8 wind_trills_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
11         trill_spinner=consort.AttachmentExpression(
12             attachments=(
13                 spannertools.ComplexTrillSpanner(interval='+P4'),
14                 spannertools.ComplexTrillSpanner(interval='+P4'),
15                 spannertools.ComplexTrillSpanner(interval='+m3'),
16                 spannertools.ComplexTrillSpanner(interval='+m3'),
17                 spannertools.ComplexTrillSpanner(interval='+P4'),
18                 spannertools.ComplexTrillSpanner(interval='+m3'),
19             ),
20             selector=selectortools.select_pitched_runs(),
21         ),
22     ),
23     pitch_handler=consort.PitchClassPitchHandler(
24         pitchSpecifier='c ef d',
25         registerSpecifier=consort.RegisterSpecifier(
26             base_pitch="g",
27         ),
28     ),
29     rhythm_maker=zaira.materials.sustained_rhythm_maker,
30 )

```

## B.2.59 ZAIRA.MATERIALS.WOOD\_AGITATION\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import pitchtools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 from abjad.tools.topleveltools import new
7 import consort
8 import zaira
9
10
11 wood_agitation_music_specifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
14         stem_tremolo_spinner=consort.AttachmentExpression(
15             attachments=spannertools.StemTremoloSpanner(),
16             selector=selectortools.Selector(
17                 ).by_logical_tie(pitched=True
18                 ).by_duration('>', (1, 16))
19             ),
20         accent=consort.AttachmentExpression(
21             attachments=(
22                 indicatortools.Articulation('accent'),
23             ),

```

```

24     selector=selectortools.Selector(
25         ).by_logical_tie(pitched=True)[0],
26     ),
27     ),
28     pitch_handler=consort.AbsolutePitchHandler(
29         pitch_specifier=pitchtools.PitchSegment(
30             items=(
31                 zaira.makers.Percussion.TAMBOURINE,
32                 zaira.makers.Percussion.GUERO,
33                 zaira.makers.Percussion.TAMBOURINE,
34                 zaira.makers.Percussion.TAMBOURINE,
35                 zaira.makers.Percussion.GUERO,
36                 zaira.makers.Percussion.GUERO,
37                 zaira.makers.Percussion.GUERO,
38                 zaira.makers.Percussion.TAMBOURINE,
39                 zaira.makers.Percussion.TAMBOURINE,
40                 zaira.makers.Percussion.GUERO,
41                 zaira.makers.Percussion.TAMBOURINE,
42                 zaira.makers.Percussion.TAMBOURINE,
43                 zaira.makers.Percussion.TAMBOURINE,
44                 zaira.makers.Percussion.TAMBOURINE,
45                 zaira.makers.Percussion.GUERO,
46                 zaira.makers.Percussion.GUERO,
47                 zaira.makers.Percussion.TAMBOURINE,
48                 zaira.makers.Percussion.GUERO,
49                 zaira.makers.Percussion.TAMBOURINE,
50                 zaira.makers.Percussion.TAMBOURINE,
51                 zaira.makers.Percussion.GUERO,
52             ),
53         ),
54     ),
55     rhythm_maker=new(
56         zaira.materials.reiterating_rhythm_maker,
57         denominators=(16, 4),
58         extra_counts_per_division=(0, 1, 0, 1, 2),
59     ),
60 )

```

## B.2.60 ZAIRA.MATERIALS.WOOD\_BAMBOO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import pitchtools
3 from abjad.tools import indicatortools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 import consort
7 import zaira
8
9
10 wood_bamboo_musicSpecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=zaira.materials.midground_dynamic_attachment_expression,
13         stem_tremolo_spanner=consort.AttachmentExpression(
14             attachments=spannertools.StemTremoloSpanner(),

```

```

15     selector=selectortools.Selector(
16         ).by_logical_tie(pitched=True
17         ).by_duration('>', (1, 16))
18     ),
19     staccato=consort.AttachmentExpression(
20         attachments=indicatortools.Articulation('.'),
21         selector=selectortools.Selector(
22             ).by_logical_tie(pitched=True
23             ).by_duration('<', (1, 8)
24             ).by_length(1)
25         ),
26     ),
27     pitch_handler=consort.AbsolutePitchHandler(
28         pitchSpecifier=pitchtools.PitchSegment(
29             items=[zaira.makers.Percussion.BAMBOO_WINDCHIMES],
30             ),
31         ),
32     rhythm_maker=zaira.materials.undergrowth_rhythm_maker,
33 )

```

## B.3 ZAIRA

### SEGMENTS

SOURCE

#### B.3.1 ZAIRA.SEGMENTS SEGMENT\_A

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 import consort
7 import zaira
8
9
10 #### SEGMENT MAKER #####
11
12
13 segment_maker = zaira.makers.ZairaSegmentMaker(
14     discard_final_silence=True,
15     permitted_time_signatures=(
16         (2, 4),
17         (3, 8),
18     ),
19     tempo=indicatortools.Tempo((1, 4), 72),
20 )
21
22 ratio = mathtools.NonreducedRatio([2])
23
24 segment_maker.desired_duration_in_seconds = (
25     durationtools.Multiplier(sum(ratio), 91) *
26     zaira.materials.total_duration_in_seconds
27 )
28
29
30 #### FANFARE SETTINGS #####

```

```

31
32
33 segment_maker.add_setting(
34     timespan_maker=consort.FloodedTimespanMaker(),
35     piano_rh=new(
36         zaira.materials.piano_fanfare_music_specifier,
37         pitch_handler__register_specifier__base_pitch="g",
38     ),
39     piano_lh=new(
40         zaira.materials.piano_fanfare_music_specifier,
41         pitch_handler__logical_tie_expressions=
42             zaira.materials.piano_fanfare_music_specifier
43                 .pitch_handler.logical_tie_expressions[:-1],
44         pitch_handler__pitchSpecifier="g c a f d f e b e",
45         pitch_handler__register_specifier__base_pitch="g,,",
46     ),
47     drums=new(
48         zaira.materials.percussion_fanfare_music_specifier,
49         pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
50     ),
51     metals=new(
52         zaira.materials.percussion_fanfare_music_specifier,
53         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
54     ),
55 )
56
57
58 ##### DEPENDENT MUSIC SETTINGS #####
59
60
61 segment_maker.add_setting(
62     timespan_maker=zaira.materials.pedals_timespan_maker,
63     piano_pedals=zaira.materials.piano_pedals_music_specifier,
64 )

```

### B.3.2 ZAIRA.SEGMENTS SEGMENT\_B

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import timespantools
7 import consort
8 import zaira
9
10
11 ##### SEGMENT MAKER #####
12
13
14 segment_maker = zaira.makers.ZairaSegmentMaker(
15     tempo=indicatortools.Tempo((1, 4), 48),
16 )
17

```

```

18 ratio = mathtools.NonreducedRatio([1, 6, 3])
19
20 segment_maker.desired_duration_in_seconds = (
21     durationtools.Multiplier(sum(ratio), 91) *
22     zaira.materials.total_duration_in_seconds
23 )
24
25 fanfare_duration = durationtools.Duration(1, 16)
26
27
28 ### WINDS SETTINGS #####
29
30
31 segment_maker.add_setting(
32     timespan_maker=zaira.materials.dense_timespan_maker,
33     timespan_identifier=consort.RatioPartsExpression(
34         parts=(0, 2, 4),
35         ratio=(1, 1, 1, 1, 1),
36         mask_timespan=timespantools.Timespan(
37             start_offset=fanfare_duration,
38         ),
39     ),
40     clarinet=zaira.materials.wind_keyclick_music_specifier,
41     flute=zaira.materials.wind_keyclick_music_specifier,
42     oboe=zaira.materials.wind_keyclick_music_specifier,
43 )
44
45
46 segment_maker.add_setting(
47     timespan_maker=zaira.materials.dense_timespan_maker,
48     timespan_identifier=consort.RatioPartsExpression(
49         parts=(1, 3),
50         ratio=(3, 1, 2, 1, 1),
51         mask_timespan=timespantools.Timespan(
52             start_offset=fanfare_duration,
53         ),
54     ),
55     oboe=zaira.materials.oboe_solo_music_specifier,
56 )
57
58
59 ### PERCUSSION SETTINGS #####
60
61
62 segment_maker.add_setting(
63     timespan_maker=new(
64         zaira.materials.dense_timespan_maker,
65         reflect=True,
66     ),
67     drums=zaira.materials.drum_tranquilo_music_specifier,
68     metals=zaira.materials.metal_tranquilo_music_specifier,
69 )
70
71

```

```

72 segment_maker.add_setting(
73     timespan_maker=new(
74         zaira.materials.sparse_timespan_maker,
75         fuse_groups=True,
76         padding=durationtools.Duration(1, 4),
77     ),
78     drums=zaira.materials.drum_brushed_music_specifier,
79     metals=zaira.materials.metal_brushed_music_specifier,
80 )
81
82
83 ### PIANO SETTINGS #####
84
85
86 segment_maker.add_setting(
87     timespan_maker=new(
88         zaira.materials.sparse_timespan_maker,
89         playing_groupings=(1,),
90         playing_talea_counts=(5, 3, 3, 3, 6, 4, 3),
91         timespan_specifier=consort.TimespanSpecifier(
92             minimum_duration=0,
93         ),
94     ),
95     timespan_identifier=consort.RatioPartsExpression(
96         parts=(1, 3, 5),
97         ratio=(1, 2, 1, 2, 1, 2, 1),
98     ),
99     piano_rh=new(
100         zaira.materials.piano_flourish_music_specifier,
101         pitch_handler__register_specifier__base_pitch="c'",
102     ),
103     piano_lh=new(
104         zaira.materials.piano_flourish_music_specifier,
105         pitch_handler__register_specifier__base_pitch="c",
106     ),
107 )
108
109
110 segment_maker.add_setting(
111     timespan_maker=zaira.materials.dense_timespan_maker,
112     timespan_identifier=consort.RatioPartsExpression(
113         parts=(0, 2, 4, 6),
114         ratio=(1, 2, 1, 2, 1, 2, 1),
115     ),
116     piano_rh=zaira.materials.piano_prepared_treble_music_specifier,
117     piano_lh=zaira.materials.piano_prepared_bass_music_specifier,
118 )
119
120
121 ### STRING SETTINGS #####
122
123
124 segment_maker.add_setting(
125     timespan_maker=new(

```

```

126     zaira.materials.sparse_timespan_maker,
127     padding=durationtools.Duration(1, 4),
128     playing_groupings=(1,),
129     playing_talea__counts=(4, 3, 2, 4, 3),
130     reflect=True,
131     ),
132     timespan_identifier=timespantools.Timespan(
133         start_offset=fanfare_duration,
134         ),
135     violin=new(
136         zaira.materials.string_flourish_music_specifier,
137         pitch_handler__registerSpecifier__base_pitch=None,
138         seed=0,
139         ),
140     viola=new(
141         zaira.materials.string_flourish_music_specifier,
142         pitch_handler__registerSpecifier__base_pitch='c',
143         seed=1,
144         ),
145     cello=new(
146         zaira.materials.string_flourish_music_specifier,
147         pitch_handler__registerSpecifier__base_pitch='c',
148         seed=2,
149         ),
150     )
151
152
153 ### FANFARE SETTINGS #####
154
155
156 segment_maker.add_setting(
157     timespan_maker=consort.FloodedTimespanMaker(),
158     timespan_identifier=timespantools.Timespan(
159         stop_offset=fanfare_duration,
160         ),
161     piano_rh=new(
162         zaira.materials.piano_fanfare_music_specifier,
163         pitch_handler__registerSpecifier__base_pitch="g",
164         ),
165     piano_lh=new(
166         zaira.materials.piano_fanfare_music_specifier,
167         pitch_handler__logical_tie_expressions=
168             zaira.materials.piano_fanfare_music_specifier
169                 .pitchHandler.logicalTieExpressions[:-1],
170         pitch_handler__pitchSpecifier="g c a f d f e b e",
171         pitch_handler__registerSpecifier__basePitch="g",
172         ),
173     drums=new(
174         zaira.materials.percussion_fanfare_music_specifier,
175         pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
176         ),
177     metals=new(
178         zaira.materials.percussion_fanfare_music_specifier,
179         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,

```

```

180     ),
181 )
182
183
184 ##### DEPENDENT MUSIC SETTINGS #####
185
186
187 segment_maker.add_setting(
188     timespan_maker=zaira.materials.pedals_timespan_maker,
189     piano_pedals=zaira.materials.piano_pedals_music_specifier,
190 )

```

### B.3.3 ZAIRA.SEGMENTS SEGMENT\_C

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespantools
8 import consort
9 import zaira
10
11
12 ##### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 72),
17 )
18
19 ratio = mathtools.NonreducedRatio([7])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multipplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24 )
25
26 fanfare_duration = durationtools.Duration(1, 4)
27
28
29 ##### WINDS SETTINGS #####
30
31
32 segment_maker.add_setting(
33     timespan_maker=zaira.materials.granular_timespan_maker,
34     timespan_identifier=timespantools.Timespan(
35         start_offset=fanfare_duration,
36     ),
37     flute=new(
38         zaira.materials.wind_trills_music_specifier,
39         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
40             operators=(

```

```

41         pitchtools.Transposition(-3),
42         pitchtools.Inversion(),
43     ),
44     ),
45     seed=0,
46 ),
47 oboe=new(
48     zaira.materials.wind_trills_music_specifier,
49     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
50         operators=(
51             pitchtools.Transposition(-3),
52             pitchtools.Inversion(),
53         ),
54     ),
55     seed=1,
56 ),
57 clarinet=new(
58     zaira.materials.wind_trills_music_specifier,
59     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
60         operators=(
61             pitchtools.Transposition(-3),
62             pitchtools.Inversion(),
63         ),
64     ),
65     seed=2,
66 ),
67 )
68
69
70 segment_maker.add_setting(
71     timespan_maker=zaira.materials.sparse_timespan_maker,
72     timespan_identifier=timespantools.Timespan(
73         start_offset=fanfare_duration,
74     ),
75     clarinet=new(
76         zaira.materials.wind_slap_music_specifier,
77         pitch_handler__registerSpecifier__base_pitch='D3',
78     ),
79     flute=zaira.materials.wind_slap_music_specifier,
80     oboe=new(
81         zaira.materials.wind_slap_music_specifier,
82         pitch_handler__registerSpecifier__base_pitch='Bb3',
83     ),
84 )
85
86
87 ### PERCUSSION SETTINGS #####
88
89
90 segment_maker.add_setting(
91     timespan_maker=zaira.materials.dense_timespan_maker,
92     timespan_identifier=timespantools.Timespan(
93         start_offset=fanfare_duration,
94     ),

```

```

95     drums=zaira.materials.drum_tranquilo_music_specifier,
96     woods=zaira.materials.wood_agitation_music_specifier,
97 )
98
99
100 segment_maker.add_setting(
101     timespan_maker=new(
102         zaira.materials.sparse_timespan_maker,
103         reflect=True,
104     ),
105     timespan_identifier=timespantools.Timespan(
106         start_offset=fanfare_duration,
107     ),
108     drums=(
109         zaira.materials.drum_brushed_music_specifier,
110         zaira.materials.drum_agitation_music_specifier,
111     ),
112 )
113
114
115 ##### PIANO SETTINGS #####
116
117
118 segment_maker.add_setting(
119     timespan_maker=zaira.materials.sparse_timespan_maker,
120     piano_rh=new(
121         zaira.materials.piano_clusters_music_specifier,
122         pitch_handler__registerSpecifier__base_pitch="g",
123     ),
124     piano_lh=new(
125         zaira.materials.piano_clusters_music_specifier,
126         pitch_handler__registerSpecifier__base_pitch="g,",
127     ),
128 )
129
130
131 segment_maker.add_setting(
132     timespan_maker=new(
133         zaira.materials.sparse_timespan_maker,
134         playing_groupings=(1,),
135         playing_talea_counts=(5, 3, 3, 3, 6, 4, 3),
136         timespanSpecifier=consort.TimespanSpecifier(
137             minimum_duration=0,
138         ),
139     ),
140     timespanIdentifier=consort.RatioPartsExpression(
141         parts=(1, 3, 5),
142         ratio=(1, 2, 1, 2, 1, 2, 1),
143     ),
144     piano_rh=new(
145         zaira.materials.piano_flourish_music_specifier,
146         pitch_handler__registerSpecifier__base_pitch="c",
147         pitch_handler__pitchOperationSpecifier=pitchtools.PitchOperation(
148             operators=(

```

```

149         pitchtools.Inversion(),
150         pitchtools.Transposition(3),
151     ),
152     ),
153 ),
154 piano_lh=new(
155     zaira.materials.piano_flourish_music_specifier,
156     pitch_handler__register_specifier__base_pitch="c",
157     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
158         operators=
159             pitchtools.Inversion(),
160             pitchtools.Transposition(3),
161         ),
162     ),
163 ),
164 )
165
166
167 segment_maker.add_setting(
168     timespan_maker=new(
169         zaira.materials.sparse_timespan_maker,
170         fuse_groups=True,
171         padding=durationtools.Duration(1, 4),
172         reflect=True,
173     ),
174     piano_rh=zaira.materials.piano_guero_music_specifier,
175     piano_lh=new(
176         zaira.materials.piano_guero_music_specifier,
177         seed=1,
178     ),
179 )
180
181
182 ### STRING SETTINGS #####
183
184
185 segment_maker.add_setting(
186     timespan_maker=new(
187         zaira.materials.sparse_timespan_maker,
188         padding=durationtools.Duration(1, 4),
189         playing_groupings=(1,),
190         playing_talea__counts=(4, 3, 2, 4, 3),
191         reflect=True,
192     ),
193     timespan_identifier=timespantools.Timespan(
194         start_offset=fanfare_duration,
195     ),
196     violin=new(
197         zaira.materials.string_flourish_music_specifier,
198         pitch_handler__register_specifier__base_pitch=None,
199         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
200             operators=
201                 pitchtools.Transposition(3),
202             ),

```

```

203         ),
204     rhythm_maker__talea__denominator=16,
205     seed=0,
206   ),
207   viola=new(
208     zaira.materials.string_flourish_music_specifier,
209     pitch_handler__register_specifier__base_pitch='c',
210     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
211       operators=(
212         pitchtools.Transposition(3),
213       ),
214     ),
215     rhythm_maker__talea__denominator=16,
216     seed=1,
217   ),
218   cello=new(
219     zaira.materials.string_flourish_music_specifier,
220     pitch_handler__register_specifier__base_pitch='c',
221     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
222       operators=(
223         pitchtools.Transposition(3),
224       ),
225     ),
226     rhythm_maker__talea__denominator=16,
227     seed=2,
228   ),
229 )
230
231
232 segment_maker.add_setting(
233   timespan_maker=new(
234     zaira.materials.tutti_timespan_maker,
235     padding=durationtools.Duration(1, 4),
236     playing_groupings=(1,),
237     reflect=True,
238   ),
239   timespan_identifier=consort.RatioPartsExpression(
240     parts=(0, 2, 4),
241     ratio=(1, 2, 1, 1, 1),
242   ),
243   violin=new(
244     zaira.materials.string_undergrowth_music_specifier,
245     pitch_handler__register_specifier__base_pitch='g',
246     seed=0,
247   ),
248   viola=new(
249     zaira.materials.string_undergrowth_music_specifier,
250     pitch_handler__register_specifier__base_pitch='c',
251     seed=1,
252   ),
253   cello=new(
254     zaira.materials.string_undergrowth_music_specifier,
255     pitch_handler__register_specifier__base_pitch='c',
256     seed=2,

```

```

257     ),
258 )
259
260
261 segment_maker.add_setting(
262     timespan_maker=zaira.materials.dense_timespan_maker,
263     timespan_identifier=consort.RatioPartsExpression(
264         parts=(1, 3),
265         ratio=(3, 1, 2, 1, 1),
266         mask_timespan=timespantools.Timespan(
267             start_offset=fanfare_duration,
268         ),
269     ),
270     cello=zaira.materials.cello_solo_music_specifier,
271 )
272
273
274 ### SHAKER SETTINGS #####
275
276
277 segment_maker.add_setting(
278     timespan_maker=new(
279         zaira.materials.sparse_timespan_maker,
280         padding=durationtools.Duration(3, 8),
281     ),
282     timespan_identifier=timespantools.Timespan(
283         start_offset=fanfare_duration,
284     ),
285     clarinet=zaira.materials.brazil_nut_music_specifier,
286     flute=zaira.materials.brazil_nut_music_specifier,
287     woods=zaira.materials.wood_bamboo_music_specifier,
288     violin=zaira.materials.brazil_nut_music_specifier,
289     viola=zaira.materials.brazil_nut_music_specifier,
290 )
291
292
293 ### FANFARE SETTINGS #####
294
295
296 segment_maker.add_setting(
297     timespan_maker=consort.FloodedTimespanMaker(),
298     timespan_identifier=timespantools.Timespan(
299         stop_offset=fanfare_duration,
300     ),
301     piano_rh=new(
302         zaira.materials.piano_fanfare_music_specifier,
303         pitch_handler__registerSpecifier__base_pitch="g",
304     ),
305     piano_lh=new(
306         zaira.materials.piano_fanfare_music_specifier,
307         pitch_handler__logical_tie_expressions=
308             zaira.materials.piano_fanfare_music_specifier
309                 .pitch_handler.logical_tie_expressions[:-1],
310         pitch_handler__pitch_specifier="g c a f d f e b e",

```

```

311     pitch_handler__register_specifier__base_pitch="g,,",
312     ),
313     drums=new(
314         zaira.materials.percussion_fanfare_music_specifier,
315         pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
316         ),
317     metals=new(
318         zaira.materials.percussion_fanfare_music_specifier,
319         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
320         ),
321     )
322
323
324 ### DEPENDENT MUSIC SETTINGS #####
325
326
327 segment_maker.add_setting(
328     timespan_maker=zaira.materials.pedals_timespan_maker,
329     piano_pedals=zaira.materials.piano_pedals_music_specifier,
330     )

```

### B.3.4 ZAIRA.SEGMENTS SEGMENT\_D

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespantools
8 import consort
9 import zaira
10
11
12 ### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 84),
17     )
18
19 ratio = mathtools.NonreducedRatio([1, 2, 1])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multiplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24     )
25
26 fanfare_duration = durationtools.Duration(1, 16)
27
28
29 ### WIND SETTINGS #####
30
31

```

```

32 segment_maker.add_setting(
33     timespan_maker=zaira.materials.granular_timespan_maker,
34     timespan_identifier=timespantools.Timespan(
35         start_offset=fanfare_duration,
36     ),
37     flute=new(
38         zaira.materials.wind_trills_music_specifier,
39         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
40             pitchtools.Inversion(),
41         ),
42         seed=0,
43     ),
44     oboe=new(
45         zaira.materials.wind_trills_music_specifier,
46         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
47             pitchtools.Inversion(),
48         ),
49         seed=1,
50     ),
51     clarinet=new(
52         zaira.materials.wind_trills_music_specifier,
53         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
54             pitchtools.Inversion(),
55         ),
56         seed=2,
57     ),
58 )
59
60
61 segment_maker.add_setting(
62     timespan_maker=zaira.materials.dense_timespan_maker,
63     timespan_identifier=consort.RatioPartsExpression(
64         parts=(1, 3),
65         ratio=(3, 1, 2, 1, 1),
66         mask_timespan=timespantools.Timespan(
67             start_offset=fanfare_duration,
68         ),
69     ),
70     oboe=new(
71         zaira.materials.oboe_solo_music_specifier,
72         seed=1,
73     ),
74 )
75
76
77 ##### PERCUSSION SETTINGS #####
78
79
80 segment_maker.add_setting(
81     timespan_maker=zaira.materials.sparse_timespan_maker,
82     metals=zaira.materials.metal_agitation_music_specifier,
83     woods=zaira.materials.wood_agitation_music_specifier,
84 )
85

```

```

86
87 ### PIANO SETTINGS #####
88
89
90 segment_maker.add_setting(
91     timespan_maker=new(
92         zaira.materials.dense_timespan_maker,
93         fuse_groups=True,
94     ),
95     piano_rh=zaira.materials.piano_drone_music_specifier,
96 )
97
98
99 segment_maker.add_setting(
100    timespan_maker=zaira.materials.dense_timespan_maker,
101    timespan_identifier=consort.RatioPartsExpression(
102        parts=(0, 2, 4),
103        ratio=(1, 2, 1, 2, 1),
104    ),
105    piano_rh=zaira.materials.piano_prepared_treble_music_specifier,
106    piano_lh=zaira.materials.piano_prepared_bass_music_specifier,
107 )
108
109
110 ### STRING SETTINGS #####
111
112
113 segment_maker.add_setting(
114     timespan_maker=new(
115         zaira.materials.granular_timespan_maker,
116         reflect=True,
117     ),
118     timespan_identifier=timespantools.Timespan(
119         start_offset=fanfare_duration,
120     ),
121     violin=new(
122         zaira.materials.string_trills_music_specifier,
123         pitch_handler__register_specifier__base_pitch="c''",
124         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
125             operators=(
126                 pitchtools.Transposition(-3),
127                 pitchtools.Inversion(),
128             ),
129         ),
130         seed=0,
131     ),
132     viola=new(
133         zaira.materials.string_trills_music_specifier,
134         pitch_handler__register_specifier__base_pitch="c''",
135         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
136             operators=(
137                 pitchtools.Transposition(-3),
138                 pitchtools.Inversion(),
139             ),

```

```

140         ),
141         seed=1,
142     ),
143     cello=new(
144         zaira.materials.string_trills_music_specifier,
145         pitch_handler__register_specifier__base_pitch='c',
146         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
147             operators=(
148                 pitchtools.Transposition(-3),
149                 pitchtools.Inversion(),
150             ),
151         ),
152         seed=2,
153     ),
154 )
155
156
157 segment_maker.add_setting(
158     timespan_maker=new(
159         zaira.materials.sparse_timespan_maker,
160         padding=durationtools.Duration(1, 4),
161         reflect=True,
162     ),
163     timespan_identifier=timespantools.Timespan(
164         start_offset=fanfare_duration,
165     ),
166     violin=new(
167         zaira.materials.string_flourish_music_specifier,
168         pitch_handler__register_specifier__base_pitch=None,
169         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
170             pitchtools.Transposition(-3),
171         ),
172         rhythm_maker__talea__denominator=16,
173         seed=0,
174     ),
175     viola=new(
176         zaira.materials.string_flourish_music_specifier,
177         pitch_handler__register_specifier__base_pitch='c',
178         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
179             pitchtools.Transposition(-3),
180         ),
181         rhythm_maker__talea__denominator=16,
182         seed=1,
183     ),
184     cello=new(
185         zaira.materials.string_flourish_music_specifier,
186         pitch_handler__register_specifier__base_pitch='c',
187         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
188             pitchtools.Transposition(-3),
189         ),
190         rhythm_maker__talea__denominator=16,
191         seed=2,
192     ),
193 )

```

```

194
195
196 ### FANFARE SETTINGS #####
197
198
199 segment_maker.add_setting(
200     timespan_maker=zaira.materials.sparse_timespan_maker,
201     piano_rh=new(
202         zaira.materials.piano_clusters_music_specifier,
203         pitch_handler__registerSpecifier__base_pitch="c''",
204         ),
205     piano_lh=new(
206         zaira.materials.piano_clusters_music_specifier,
207         pitch_handler__registerSpecifier__base_pitch="c",
208         ),
209     )
210
211
212 segment_maker.add_setting(
213     timespan_maker=consort.FloodedTimespanMaker(),
214     timespan_identifier=timespanTools.Timespan(
215         stop_offset=fanfare_duration,
216         ),
217     piano_rh=new(
218         zaira.materials.piano_fanfare_music_specifier,
219         pitch_handler__registerSpecifier__base_pitch="g",
220         ),
221     piano_lh=new(
222         zaira.materials.piano_fanfare_music_specifier,
223         pitch_handler__logical_tie_expressions=
224             zaira.materials.piano_fanfare_music_specifier
225                 .pitchHandler.logical_tie_expressions[:-1],
226         pitch_handler__pitchSpecifier="g c a f d f e b e",
227         pitch_handler__registerSpecifier__basePitch="g,,",
228         ),
229     drums=new(
230         zaira.materials.percussion_fanfare_music_specifier,
231         pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
232         ),
233     metals=new(
234         zaira.materials.percussion_fanfare_music_specifier,
235         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
236         ),
237     )
238
239
240 ### DEPENDENT MUSIC SETTING #####
241
242
243 segment_maker.add_setting(
244     timespan_maker=zaira.materials.pedals_timespan_maker,
245     piano_pedals=zaira.materials.piano_pedals_music_specifier,
246     )

```

### B.3.5 ZAIRA.SEGMENTS.SEGMENT\_E

```
1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespanools
8 import consort
9 import zaira
10
11
12 ### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 72),
17 )
18
19 ratio = mathtools.NonreducedRatio([1, 8, 4])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multiplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24 )
25
26 fanfare_duration = durationtools.Duration(3, 4)
27
28
29 ### WINDS SETTINGS #####
30
31
32 segment_maker.add_setting(
33     timespan_maker=new(
34         zaira.materials.granular_timespan_maker,
35         reflect=True,
36     ),
37     timespan_identifier=timespanools.Timespan(
38         start_offset=fanfare_duration,
39     ),
40     flute=new(
41         zaira.materials.wind_trills_music_specifier,
42         seed=0,
43     ),
44     oboe=new(
45         zaira.materials.wind_trills_music_specifier,
46         seed=1,
47     ),
48     clarinet=new(
49         zaira.materials.wind_trills_music_specifier,
50         seed=2,
51     ),
52 )
```

```

53
54
55 segment_maker.add_setting(
56     timespan_maker=new(
57         zaira.materials.dense_timespan_maker,
58         playing_groupings=(1,),
59     ),
60     timespan_identifier=consort.RatioPartsExpression(
61         parts=(1, 3, 5),
62         ratio=(1, 2, 1, 2, 1, 2, 1),
63         mask_timespan=timespantools.Timespan(
64             start_offset=fanfare_duration,
65         ),
66     ),
67     clarinet=new(
68         zaira.materials.wind_airtone_music_specifier,
69         pitch_handler__register_specifier__base_pitch='D3',
70     ),
71     flute=zaira.materials.wind_airtone_music_specifier,
72     oboe=new(
73         zaira.materials.wind_airtone_music_specifier,
74         pitch_handler__register_specifier__base_pitch='Bb3',
75     ),
76 )
77
78
79 segment_maker.add_setting(
80     timespan_maker=zaira.materials.dense_timespan_maker,
81     timespan_identifier=consort.RatioPartsExpression(
82         parts=(0, 2, 4, 6),
83         ratio=(1, 1, 3, 1, 1, 1, 2),
84         mask_timespan=timespantools.Timespan(
85             start_offset=fanfare_duration,
86         ),
87     ),
88     clarinet=zaira.materials.wind_keyclick_music_specifier,
89     flute=zaira.materials.wind_keyclick_music_specifier,
90     oboe=zaira.materials.wind_keyclick_music_specifier,
91 )
92
93
94 segment_maker.add_setting(
95     timespan_maker=zaira.materials.sparse_timespan_maker,
96     timespan_identifier=timespantools.Timespan(
97         start_offset=fanfare_duration,
98     ),
99     clarinet=new(
100         zaira.materials.wind_slap_music_specifier,
101         pitch_handler__register_specifier__base_pitch='D3',
102     ),
103     flute=zaira.materials.wind_slap_music_specifier,
104     oboe=new(
105         zaira.materials.wind_slap_music_specifier,
106         pitch_handler__register_specifier__base_pitch='Bb3',

```

```

107     ),
108 )
109
110
111 segment_maker.add_setting(
112     timespan_maker=new(
113         zaira.materials.dense_timespan_maker,
114         reflect=True,
115     ),
116     timespan_identifier=consort.RatioPartsExpression(
117         parts=(0, 2, 4),
118         ratio=(1, 1, 1, 1, 1),
119         mask_timespan=timespantools.Timespan(
120             start_offset=fanfare_duration,
121         ),
122     ),
123     oboe=zaira.materials.oboe_solo_music_specifier,
124 )
125
126
127 ### PERCUSSION SETTINGS #####
128
129
130 segment_maker.add_setting(
131     timespan_maker=zaira.materials.dense_timespan_maker,
132     drums=zaira.materials.drum_agitation_music_specifier,
133     metals=zaira.materials.metal_agitation_music_specifier,
134 )
135
136
137 segment_maker.add_setting(
138     timespan_maker=new(
139         zaira.materials.sparse_timespan_maker,
140         fuse_groups=True,
141         padding=durationtools.Duration(1, 4),
142         reflect=True,
143     ),
144     drums=zaira.materials.drum_brushed_music_specifier,
145     metals=zaira.materials.metal_brushed_music_specifier,
146 )
147
148
149 ### PIANO SETTINGS #####
150
151
152 segment_maker.add_setting(
153     timespan_maker=zaira.materials.dense_timespan_maker,
154     timespan_identifier=consort.RatioPartsExpression(
155         parts=(0, 2, 4, 6),
156         ratio=(1, 2, 1, 2, 1, 2, 1),
157     ),
158     piano_rh=zaira.materials.piano_prepared_treble_music_specifier,
159     piano_lh=zaira.materials.piano_prepared_bass_music_specifier,
160 )

```

```

161
162
163 segment_maker.add_setting(
164     timespan_maker=zaira.materials.sparse_timespan_maker,
165     timespan_identifier=consort.RatioPartsExpression(
166         parts=(1,),
167         ratio=ratio,
168     ),
169     piano_rh=new(
170         zaira.materials.piano_clusters_music_specifier,
171         pitch_handler__registerSpecifier__base_pitch="g",
172     ),
173     piano_lh=new(
174         zaira.materials.piano_clusters_music_specifier,
175         pitch_handler__registerSpecifier__base_pitch="c",
176     ),
177 )
178
179
180 segment_maker.add_setting(
181     timespan_maker=zaira.materials.dense_timespan_maker,
182     timespan_identifier=consort.RatioPartsExpression(
183         parts=(2,),
184         ratio=ratio,
185     ),
186     piano_rh=new(
187         zaira.materials.piano_clusters_music_specifier,
188         attachment_handler__dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
189         pitch_handler__registerSpecifier__base_pitch="g",
190     ),
191     piano_lh=new(
192         zaira.materials.piano_clusters_music_specifier,
193         attachment_handler__dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
194         pitch_handler__registerSpecifier__base_pitch="c",
195     ),
196 )
197
198
199 segment_maker.add_setting(
200     timespan_maker=new(
201         zaira.materials.sparse_timespan_maker,
202         fuse_groups=True,
203         padding=durationtools.Duration(1, 4),
204         reflect=True,
205     ),
206     timespan_identifier=consort.RatioPartsExpression(
207         parts=(0, 2, 4),
208         ratio=(1, 2, 1, 2, 1),
209     ),
210     piano_rh=zaira.materials.piano_guero_music_specifier,
211     piano_lh=new(
212         zaira.materials.piano_guero_music_specifier,
213         seed=1,
214     ),

```

```

215     )
216
217
218 ### STRING SETTINGS #####
219
220
221 segment_maker.add_setting(
222     timespan_maker=zaira.materials.granular_timespan_maker,
223     timespan_identifier=timespantools.Timespan(
224         start_offset=fanfare_duration,
225         ),
226     violin=new(
227         zaira.materials.string_trills_music_specifier,
228         pitch_handler__register_specifier__base_pitch="c",
229         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
230             pitchtools.Transposition(-3),
231             ),
232         seed=0,
233         ),
234     viola=new(
235         zaira.materials.string_trills_music_specifier,
236         pitch_handler__register_specifier__base_pitch="c",
237         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
238             pitchtools.Transposition(-3),
239             ),
240         seed=1,
241         ),
242     cello=new(
243         zaira.materials.string_trills_music_specifier,
244         pitch_handler__register_specifier__base_pitch='c',
245         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
246             pitchtools.Transposition(-3),
247             ),
248         seed=2,
249         ),
250     )
251
252
253 segment_maker.add_setting(
254     timespan_maker=new(
255         zaira.materials.tutti_timespan_maker,
256         padding=durationtools.Duration(1, 4),
257         playing_groupings=(1,),
258         reflect=True,
259         ),
260     timespan_identifier=consort.RatioPartsExpression(
261         parts=(0, 2, 4, 6),
262         ratio=(1, 2, 1, 1, 1, 2, 1),
263         mask_timespan=timespantools.Timespan(
264             start_offset=fanfare_duration,
265             ),
266         ),
267     violin=new(
268         zaira.materials.string_undergrowth_music_specifier,

```

```

269     pitch_handler__register_specifier__base_pitch='g',
270     seed=0,
271     ),
272     viola=new(
273         zaira.materials.string_undergrowth_musicSpecifier,
274         pitch_handler__register_specifier__base_pitch='c',
275         seed=1,
276         ),
277     cello=new(
278         zaira.materials.string_undergrowth_musicSpecifier,
279         pitch_handler__register_specifier__base_pitch='c',
280         seed=2,
281         ),
282     )
283
284
285 segment_maker.add_setting(
286     timespan_maker=new(
287         zaira.materials.tutti_timespan_maker,
288         padding=durationtools.Duration(1, 4),
289         playing_groupings=(1,),
290         playing_talea_counts=(8,),
291         reflect=True,
292         repeat=False,
293         ),
294     timespan_identifier=consort.RatioPartsExpression(
295         parts=(1, 3, 5),
296         ratio=(1, 2, 1, 1, 1, 2, 1),
297         mask_timespan=timespantools.Timespan(
298             start_offset=fanfare_duration,
299             ),
300         ),
301     violin=new(
302         zaira.materials.string_tutti_overpressure_musicSpecifier,
303         pitch_handler__pitchSpecifier='g',
304         seed=0,
305         ),
306     viola=new(
307         zaira.materials.string_tutti_overpressure_musicSpecifier,
308         pitch_handler__pitchSpecifier='c',
309         seed=1,
310         ),
311     cello=new(
312         zaira.materials.string_tutti_overpressure_musicSpecifier,
313         pitch_handler__pitchSpecifier='c',
314         seed=2,
315         ),
316     )
317
318
319 #### FANFARE SETTINGS #####
320
321
322 segment_maker.add_setting(

```

```

323     timespan_maker=consort.FloodedTimespanMaker(),
324     timespan_identifier=timespantools.Timespan(
325         stop_offset=fanfare_duration,
326     ),
327     piano_rh=new(
328         zaira.materials.piano_fanfare_music_specifier,
329         pitch_handler__registerSpecifier__base_pitch="g",
330     ),
331     piano_lh=new(
332         zaira.materials.piano_fanfare_music_specifier,
333         pitch_handler__logical_tie_expressions=
334             zaira.materials.piano_fanfare_music_specifier
335                 .pitch_handler.logical_tie_expressions[:-1],
336         pitch_handler__pitchSpecifier="g c a f d f e b e",
337         pitch_handler__registerSpecifier__base_pitch="g,,",
338     ),
339     drums=new(
340         zaira.materials.percussion_fanfare_music_specifier,
341         pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
342     ),
343     metals=new(
344         zaira.materials.percussion_fanfare_music_specifier,
345         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
346     ),
347 )
348
349
350 ##### DEPENDENT MUSIC SETTINGS #####
351
352
353 segment_maker.add_setting(
354     timespan_maker=zaira.materials.pedals_timespan_maker,
355     piano_pedals=zaira.materials.piano_pedals_music_specifier,
356 )

```

### B.3.6 ZAIRA.SEGMENTS.SEGMENT\_F1

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespantools
8 import consort
9 import zaira
10
11
12 ##### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 48),
17 )

```

```

18
19 ratio = mathtools.NonreducedRatio([1, 14])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multiplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24 )
25
26 fanfare_duration = durationtools.Duration(1, 16),
27
28
29 ##### WINDS SETTINGS #####
30
31
32 segment_maker.add_setting(
33     timespan_maker=new(
34         zaira.materials.dense_timespan_maker,
35         playing_groupings=(1,),
36     ),
37     timespan_identifier=consort.RatioPartsExpression(
38         parts=(1, 3, 5),
39         ratio=(1, 1, 2, 1, 2, 1, 2),
40         mask_timespan=timespantools.Timespan(
41             start_offset=fanfare_duration,
42         ),
43     ),
44     clarinet=new(
45         zaira.materials.wind_airtone_music_specifier,
46         pitch_handler__registerSpecifier__base_pitch='D3',
47     ),
48     flute=zaira.materials.wind_airtone_music_specifier,
49     oboe=new(
50         zaira.materials.wind_airtone_music_specifier,
51         pitch_handler__registerSpecifier__base_pitch='Bb3',
52     ),
53 )
54
55
56 segment_maker.add_setting(
57     timespan_maker=zaira.materials.sparse_timespan_maker,
58     timespan_identifier=consort.RatioPartsExpression(
59         parts=(0, 2, 4),
60         ratio=(3, 1, 2, 1, 1),
61         mask_timespan=timespantools.Timespan(
62             start_offset=fanfare_duration,
63         ),
64     ),
65     clarinet=zaira.materials.wind_keyclick_music_specifier,
66     flute=zaira.materials.wind_keyclick_music_specifier,
67     oboe=zaira.materials.wind_keyclick_music_specifier,
68 )
69
70
71 segment_maker.add_setting(

```

```

72     timespan_maker=new(
73         zaira.materials.granular_timespan_maker,
74         playing_talea__counts=(2, 2, 3, 2, 7, 1, 3, 2, 1),
75         ),
76     timespan_identifier=consort.RatioPartsExpression(
77         parts=(0, 2, 4),
78         ratio=(1, 1, 1, 1, 1),
79         mask_timespan=timespantools.Timespan(
80             start_offset=fanfare_duration,
81             ),
82         ),
83     oboe=new(
84         zaira.materials.oboe_solo_music_specifier,
85         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
86             pitchtools.Transposition(-3),
87             ),
88         ),
89     )
90
91
92 ### PERCUSSION SETTINGS #####
93
94
95 segment_maker.add_setting(
96     timespan_maker=new(
97         zaira.materials.sustained_timespan_maker,
98         fuse_groups=True,
99         ),
100    drums=new(
101        zaira.materials.percussion_brushed_music_specifier,
102        pitch_handler__pitchSpecifier=zaira.makers.Percussion.BASS_DRUM,
103        ),
104    metals=new(
105        zaira.materials.percussion_brushed_music_specifier,
106        pitch_handler__pitchSpecifier=zaira.makers.Percussion.TAM_TAM,
107        ),
108    )
109
110
111 segment_maker.add_setting(
112     timespan_maker=new(
113         zaira.materials.sparse_timespan_maker,
114         fuse_groups=True,
115         ),
116     timespan_identifier=consort.RatioPartsExpression(
117         parts=(0, 2, 4),
118         ratio=(1, 1, 1, 1, 1),
119         ),
120     drums=new(
121         zaira.materials.percussion_brushed_tremolo_music_specifier,
122         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BASS_DRUM,
123         ),
124     metals=new(
125         zaira.materials.percussion_brushed_tremolo_music_specifier,

```

```

126     pitch_handler__pitch_specifier=zaira.makers.Percussion.TAM_TAM,
127     ),
128 )
129
130
131 segment_maker.add_setting(
132     timespan_maker=new(
133         zaira.materials.sparse_timespan_maker,
134         fuse_groups=True,
135         ),
136         timespan_identifier=consort.RatioPartsExpression(
137             parts=(1, 3),
138             ratio=(1, 1, 1, 1, 1),
139             ),
140         drums=new(
141             zaira.materials.percussion_superball_music_specifier,
142             pitch_handler__pitch_specifier=zaira.makers.Percussion.BASS_DRUM,
143             ),
144         metals=new(
145             zaira.materials.percussion_superball_music_specifier,
146             pitch_handler__pitch_specifier=zaira.makers.Percussion.TAM_TAM,
147             ),
148     )
149
150
151 #### PIANO SETTINGS #####
152
153
154 segment_maker.add_setting(
155     timespan_maker=new(
156         zaira.materials.dense_timespan_maker,
157         fuse_groups=True,
158         ),
159         piano_rh=zaira.materials.piano_drone_music_specifier,
160     )
161
162
163 segment_maker.add_setting(
164     timespan_maker=new(
165         zaira.materials.sparse_timespan_maker,
166         playing_groupings=(1,),
167         playing_talea__counts=(3, 3, 3, 3, 5, 3, 4),
168         timespanSpecifier=consort.TimespanSpecifier(
169             minimum_duration=0,
170             ),
171         ),
172         timespan_identifier=consort.RatioPartsExpression(
173             parts=(1, 3, 5),
174             ratio=(3, 1, 2, 1, 3, 1, 1),
175             ),
176         piano_rh=new(
177             zaira.materials.piano_flourish_music_specifier,
178             pitch_handler__registerSpecifier__base_pitch="c'",
179             pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(

```

```

180         pitchtools.Transposition(3),
181         ),
182     seed=1,
183     ),
184 piano_lh=new(
185     zaira.materials.piano_flourish_music_specifier,
186     pitch_handler__register_specifier__base_pitch="c",
187     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
188         pitchtools.Transposition(3),
189         ),
190     seed=2,
191     ),
192 )
193
194
195 ### STRING SETTINGS #####
196
197
198 segment_maker.add_setting(
199     timespan_maker=new(
200         zaira.materials.tutti_timespan_maker,
201         padding=durationtools.Duration(1, 4),
202         playing_groupings=(1,),
203         reflect=True,
204         ),
205         timespan_identifier=consort.RatioPartsExpression(
206             parts=(0, 2, 4, 6),
207             ratio=(1, 2, 1, 1, 1, 2, 1),
208             mask_timespan=timespantools.Timespan(
209                 start_offset=fanfare_duration,
210                 ),
211             ),
212         violin=new(
213             zaira.materials.string_undergrowth_music_specifier,
214             pitch_handler__register_specifier__base_pitch='g',
215             seed=0,
216             ),
217         viola=new(
218             zaira.materials.string_undergrowth_music_specifier,
219             pitch_handler__register_specifier__base_pitch='c',
220             seed=1,
221             ),
222         cello=new(
223             zaira.materials.string_undergrowth_music_specifier,
224             pitch_handler__register_specifier__base_pitch='c',
225             seed=2,
226             ),
227     )
228
229
230 segment_maker.add_setting(
231     timespan_maker=zaira.materials.dense_timespan_maker,
232     timespan_identifier=consort.RatioPartsExpression(
233         parts=(1, 3),

```

```

234     ratio=(3, 1, 2, 1, 1),
235     mask_timespan=timespantools.Timespan(
236         start_offset=fanfare_duration,
237         ),
238     ),
239     cello=new(
240         zaira.materials.cello_solo_music_specifier,
241         seed=1,
242         ),
243     )
244
245
246 ### SHAKER SETTINGS #####
247
248
249 segment_maker.add_setting(
250     timespan_maker=new(
251         zaira.materials.tutti_timespan_maker,
252         playing_talea_counts=(3, 2, 3, 3, 2, 4),
253         playing_groupings=(1,),
254         padding=durationtools.Duration(3, 8),
255         ),
256         clarinet=zaira.materials.brazil_nut_music_specifier,
257         flute=zaira.materials.brazil_nut_music_specifier,
258         viola=zaira.materials.brazil_nut_music_specifier,
259         violin=zaira.materials.brazil_nut_music_specifier,
260     )
261
262
263 ### FANFARE SETTINGS #####
264
265
266 segment_maker.add_setting(
267     timespan_maker=consort.FloodedTimespanMaker(),
268     timespan_identifier=timespantools.Timespan(
269         stop_offset=fanfare_duration,
270         ),
271     piano_rh=new(
272         zaira.materials.piano_fanfare_music_specifier,
273         pitch_handler__registerSpecifier__basePitch="g",
274         ),
275     piano_lh=new(
276         zaira.materials.piano_fanfare_music_specifier,
277         pitchHandler__logicalTieExpressions=
278             zaira.materials.piano_fanfare_music_specifier
279                 .pitchHandler.logicalTieExpressions[:-1],
280         pitchHandler__pitchSpecifier="g c a f d f e b e",
281         pitchHandler__registerSpecifier__basePitch="g,,",
282         ),
283     drums=new(
284         zaira.materials.percussion_fanfare_music_specifier,
285         pitchHandler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
286         ),
287     metals=new(

```

```

288     zaira.materials.percussion_fanfare_music_specifier,
289     pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
290     ),
291   )
292
293
294 ### DEPENDENT MUSIC SETTINGS #####
295
296
297 segment_maker.add_setting(
298   timespan_maker=zaira.materials.pedals_timespan_maker,
299   piano_pedals=zaira.materials.piano_pedals_music_specifier,
300 )

```

### B.3.7 ZAIRA.SEGMENTS SEGMENT\_F2

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 import consort
8 import zaira
9
10
11 ### SEGMENT MAKER #####
12
13
14 segment_maker = zaira.makers.ZairaSegmentMaker(
15   tempo=indicatortools.Tempo((1, 4), 48),
16   )
17
18 ratio = mathtools.NonreducedRatio([7])
19
20 segment_maker.desired_duration_in_seconds = (
21   durationtools.Multiplier(sum(ratio), 91) *
22   zaira.materials.total_duration_in_seconds
23   )
24
25
26 ### WINDS SETTINGS #####
27
28
29 segment_maker.add_setting(
30   timespan_maker=new(
31     zaira.materials.dense_timespan_maker,
32     playing_groupings=(1,),
33     silence_talea_denominator=4,
34     ),
35   timespan_identifier=consort.RatioPartsExpression(
36     parts=(1, 3, 5),
37     ratio=(1, 1, 2, 1, 2, 1, 2),
38     ),

```

```

39     clarinet=new(
40         zaira.materials.wind_aitone_music_specifier,
41         pitch_handler__registerSpecifier__base_pitch='D3',
42         ),
43     flute=zaira.materials.wind_aitone_music_specifier,
44     oboe=new(
45         zaira.materials.wind_aitone_music_specifier,
46         pitch_handler__registerSpecifier__base_pitch='Bb3',
47         ),
48     )
49
50
51 segment_maker.add_setting(
52     timespan_maker=zaira.materials.sparse_timespan_maker,
53     timespan_identifier=consort.RatioPartsExpression(
54         parts=(0, 2, 4),
55         ratio=(1, 2, 1, 2, 1),
56         ),
57     clarinet=zaira.materials.wind_keyclick_music_specifier,
58     flute=zaira.materials.wind_keyclick_music_specifier,
59     oboe=zaira.materials.wind_keyclick_music_specifier,
60     )
61
62
63 segment_maker.add_setting(
64     timespan_maker=new(
65         zaira.materials.granular_timespan_maker,
66         playing_talea__counts=(1, 1, 1, 1, 3, 2, 1),
67         ),
68     timespan_identifier=consort.RatioPartsExpression(
69         parts=(1, 3),
70         ratio=(2, 1, 2, 1, 1),
71         ),
72     oboe=new(
73         zaira.materials.oboe_solo_music_specifier,
74         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
75             pitchtools.Transposition(-5),
76             ),
77         ),
78     )
79
80
81 ### PERCUSSION SETTINGS #####
82
83
84 segment_maker.add_setting(
85     timespan_maker=new(
86         zaira.materials.sustained_timespan_maker,
87         fuse_groups=True,
88         timespanSpecifier=consort.TimespanSpecifier(
89             minimum_duration=durationtools.Duration(1, 4),
90             ),
91         ),
92     drums=new(

```

```

93     zaira.materials.percussion_superball_music_specifier,
94     pitch_handler__pitchSpecifier=zaira.makers.Percussion.BASS_DRUM,
95   ),
96   metals=new(
97     zaira.materials.percussion_superball_music_specifier,
98     pitch_handler__pitchSpecifier=zaira.makers.Percussion.TAM_TAM,
99   ),
100 )
101
102
103 segment_maker.add_setting(
104   timespan_maker=zaira.materials.sparse_timespan_maker,
105   timespan_identifier=consort.RatioPartsExpression(
106     parts=(1, 3),
107     ratio=(1, 1, 1, 1, 1),
108   ),
109   drums=new(
110     zaira.materials.percussion_brushed_music_specifier,
111     pitch_handler__pitchSpecifier=zaira.makers.Percussion.TAM_TAM,
112   ),
113   metals=new(
114     zaira.materials.percussion_brushed_music_specifier,
115     pitch_handler__pitchSpecifier=zaira.makers.Percussion.BASS_DRUM,
116   ),
117 )
118
119
120 ### PIANO SETTINGS #####
121
122
123 segment_maker.add_setting(
124   timespan_maker=new(
125     zaira.materials.sustained_timespan_maker,
126     fuseGroups=True,
127     reflect=True,
128   ),
129   piano_rh=new(
130     zaira.materials.piano_drone_music_specifier,
131     pitchHandler__pitchSpecifier='b',
132   ),
133 )
134
135
136 ### STRING SETTINGS #####
137
138
139 segment_maker.add_setting(
140   timespan_maker=new(
141     zaira.materials.tutti_timespan_maker,
142     padding=durationtools.Duration(1, 4),
143     playingGroupings=(1, ),
144     reflect=True,
145   ),
146   timespanIdentifier=consort.RatioPartsExpression(

```

```

147     parts=(0, 2, 4),
148     ratio=(1, 2, 1, 3, 1),
149     ),
150     violin=new(
151         zaira.materials.string_undergrowth_music_specifier,
152         pitch_handler__register_specifier__base_pitch='g',
153         seed=0,
154         ),
155     viola=new(
156         zaira.materials.string_undergrowth_music_specifier,
157         pitch_handler__register_specifier__base_pitch='c',
158         seed=1,
159         ),
160     cello=new(
161         zaira.materials.string_undergrowth_music_specifier,
162         pitch_handler__register_specifier__base_pitch='c',
163         seed=2,
164         ),
165     )
166
167
168 segment_maker.add_setting(
169     timespan_maker=new(
170         zaira.materials.dense_timespan_maker,
171         reflect=True,
172         ),
173     timespan_identifier=consort.RatioPartsExpression(
174         parts=(0, 2, 4),
175         ratio=(1, 1, 1, 1, 1),
176         ),
177     cello=zaira.materials.cello_solo_music_specifier,
178     )
179
180
181 ### SHAKER SETTINGS #####
182
183
184 segment_maker.add_setting(
185     timespan_maker=new(
186         zaira.materials.tutti_timespan_maker,
187         playing_talea_counts=(3, 2, 3, 3, 2, 4),
188         playing_groupings=(1,),
189         padding=durationtools.Duration(3, 8),
190         ),
191     clarinet=zaira.materials.brazil_nut_music_specifier,
192     flute=zaira.materials.brazil_nut_music_specifier,
193     viola=zaira.materials.brazil_nut_music_specifier,
194     violin=zaira.materials.brazil_nut_music_specifier,
195     )
196
197
198 ### DEPENDENT MUSIC SETTINGS #####
199
200

```

```

201 segment_maker.add_setting(
202     timespan_maker=zaira.materials.pedals_timespan_maker,
203     piano_pedals=zaira.materials.piano_pedals_music_specifier,
204 )

```

### B.3.8 ZAIRA.SEGMENTS SEGMENT\_G

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespantools
8 import consort
9 import zaira
10
11
12 ### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 96),
17 )
18
19 ratio = mathtools.NonreducedRatio([1, 4, 2])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multiplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24 )
25
26 fanfare_duration = durationtools.Duration(13, 16)
27
28
29 ### WINDS SETTINGS #####
30
31
32 segment_maker.add_setting(
33     timespan_maker=new(
34         zaira.materials.granular_timespan_maker,
35         reflect=True,
36     ),
37     timespan_identifier=timespantools.Timespan(
38         start_offset=fanfare_duration,
39     ),
40     flute=new(
41         zaira.materials.wind_trills_music_specifier,
42         pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
43             pitchtools.Inversion(),
44         ),
45         seed=0,
46     ),
47     oboe=new(

```

```

48     zaira.materials.wind_trills_music_specifier,
49     pitch_handler__pitch_operationSpecifier=pitchtools.PitchOperation(
50         pitchtools.Inversion(),
51         ),
52         seed=1,
53         ),
54     clarinet=new(
55         zaira.materials.wind_trills_music_specifier,
56         pitch_handler__pitch_operationSpecifier=pitchtools.PitchOperation(
57             pitchtools.Inversion(),
58             ),
59             seed=2,
60             ),
61     )
62
63
64 ### PERCUSSION SETTINGS #####
65
66
67 segment_maker.add_setting(
68     timespan_maker=new(
69         zaira.materials.dense_timespan_maker,
70         reflect=True,
71         ),
72         drums=zaira.materials.drum_storm_music_specifier,
73         )
74
75
76 segment_maker.add_setting(
77     timespan_maker=new(
78         zaira.materials.sparse_timespan_maker,
79         ),
80         drums=zaira.materials.drum_agitation_music_specifier,
81         )
82
83
84 segment_maker.add_setting(
85     timespan_maker=zaira.materials.sparse_timespan_maker,
86     timespan_identifier=consort.RatioPartsExpression(
87         parts=(1, 3),
88         ratio=(2, 1, 1, 1, 1),
89         ),
90     metals=new(
91         zaira.materials.percussion_reiteration_music_specifier,
92         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BAMBOO_WINDCHIMES,
93         ),
94     )
95
96
97 ### PIANO SETTINGS #####
98
99
100 segment_maker.add_setting(
101     timespan_maker=zaira.materials.tutti_timespan_maker,

```

```

102     piano_rh=new(
103         zaira.materials.piano_clusters_music_specifier,
104         pitch_handler__register_specifier__base_pitch="g'",
105         ),
106     piano_lh=new(
107         zaira.materials.piano_clusters_music_specifier,
108         pitch_handler__register_specifier__base_pitch="g",
109         seed=1,
110         ),
111     )
112
113
114 ### STRING SETTINGS #####
115
116
117 segment_maker.add_setting(
118     timespan_maker=zaira.materials.granular_timespan_maker,
119     timespan_identifier=timespantools.Timespan(
120         start_offset=fanfare_duration,
121         ),
122     violin=new(
123         zaira.materials.string_trills_music_specifier,
124         pitch_handler__register_specifier__base_pitch="c",
125         seed=0,
126         ),
127     viola=new(
128         zaira.materials.string_trills_music_specifier,
129         pitch_handler__register_specifier__base_pitch='c',
130         seed=1,
131         ),
132     cello=new(
133         zaira.materials.string_trills_music_specifier,
134         pitch_handler__register_specifier__base_pitch='c',
135         seed=2,
136         ),
137     )
138
139
140 segment_maker.add_setting(
141     timespan_maker=new(
142         zaira.materials.tutti_timespan_maker,
143         padding=durationtools.Duration(1, 4),
144         playing_groupings=(1,),
145         playing_talea__counts=(8,),
146         repeat=False,
147         ),
148     timespan_identifier=consort.RatioPartsExpression(
149         parts=(1, 3, 5),
150         ratio=(1, 2, 1, 1, 1, 2, 1),
151         mask_timespan=timespantools.Timespan(
152             start_offset=fanfare_duration,
153             ),
154             ),
155     violin=new(

```

```

156     zaira.materials.string_tutti_overpressure_music_specifier,
157     pitch_handler__pitchSpecifier='g',
158     seed=0,
159   ),
160   viola=new(
161     zaira.materials.string_tutti_overpressure_music_specifier,
162     pitch_handler__pitchSpecifier='c',
163     seed=1,
164   ),
165   cello=new(
166     zaira.materials.string_tutti_overpressure_music_specifier,
167     pitch_handler__pitchSpecifier='c',
168     seed=2,
169   ),
170 )
171
172
173 segment_maker.add_setting(
174   timespan_maker=new(
175     zaira.materials.granular_timespan_maker,
176     playing_talea__counts=(5, 4, 6, 5, 3, 5, 9),
177   ),
178   timespan_identifier=consort.RatioPartsExpression(
179     parts=(1, 3, 5),
180     ratio=(1, 1, 1, 1, 1, 1, 1),
181     mask_timespan=timespantools.Timespan(
182       start_offset=fanfare_duration,
183     ),
184   ),
185   cello=new(
186     zaira.materials.cello_solo_music_specifier,
187     pitch_handler__pitchOperationSpecifier=pitchtools.PitchOperation(
188       pitchtools.Transposition(3),
189     ),
190   ),
191 )
192
193
194 ##### FANFARE SETTINGS #####
195
196
197 segment_maker.add_setting(
198   timespan_maker=consort.FloodedTimespanMaker(),
199   timespan_identifier=timespantools.Timespan(
200     stop_offset=fanfare_duration,
201   ),
202   piano_rh=new(
203     zaira.materials.piano_fanfare_music_specifier,
204     pitch_handler__registerSpecifier__basePitch="g",
205   ),
206   piano_lh=new(
207     zaira.materials.piano_fanfare_music_specifier,
208     pitch_handler__logicalTieExpressions=
209       zaira.materials.piano_fanfare_music_specifier

```

```

210     .pitch_handler.logical_tie_expressions[:-1],
211     pitch_handler__pitch_specifier="g c a f d f e b e",
212     pitch_handler__register_specifier__base_pitch="g,,",
213     ),
214     drums=new(
215         zaira.materials.percussion_fanfare_music_specifier,
216         pitch_handler__pitch_specifier=zaira.makers.Percussion.KICK_DRUM,
217         ),
218     metals=new(
219         zaira.materials.percussion_fanfare_music_specifier,
220         pitch_handler__pitch_specifier=zaira.makers.Percussion.BRAKE_DRUM,
221         ),
222     )
223
224
225
226 ### DEPENDENT MUSIC SETTINGS #####
227
228
229 segment_maker.add_setting(
230     timespan_maker=zaira.materials.pedals_timespan_maker,
231     piano_pedals=zaira.materials.piano_pedals_music_specifier,
232     )

```

### B.3.9 ZAIRA.SEGMENTS SEGMENT\_H

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespantools
8 import consort
9 import zaira
10
11
12 ### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 72),
17     )
18
19 ratio = mathtools.NonreducedRatio([5])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multiplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24     )
25
26 fanfare_duration = durationtools.Duration(1, 8),
27
28

```

```

29 ### WINDS SETTINGS #####
30
31
32 segment_maker.add_setting(
33     timespan_maker=new(
34         zaira.materials.sparse_timespan_maker,
35         playing_groupings=(1,),
36         reflect=True,
37         ),
38         timespan_identifier=consort.RatioPartsExpression(
39             parts=(1, 3, 5),
40             ratio=(1, 2, 1, 2, 1, 2, 1),
41             mask_timespan=timespantools.Timespan(
42                 start_offset=fanfare_duration,
43                 ),
44             ),
45             clarinet=new(
46                 zaira.materials.wind_airtone_music_specifier,
47                 pitch_handler__register_specifier__base_pitch='D3',
48                 ),
49             flute=zaira.materials.wind_airtone_music_specifier,
50             oboe=new(
51                 zaira.materials.wind_airtone_music_specifier,
52                 pitch_handler__register_specifier__base_pitch='Bb3',
53                 ),
54             )
55
56
57 segment_maker.add_setting(
58     timespan_maker=new(
59         zaira.materials.sparse_timespan_maker,
60         playing_groupings=(1,),
61         ),
62         timespan_identifier=timespantools.Timespan(
63             start_offset=fanfare_duration,
64             ),
65             clarinet=new(
66                 zaira.materials.wind_slap_music_specifier,
67                 pitch_handler__register_specifier__base_pitch='D3',
68                 ),
69             flute=zaira.materials.wind_slap_music_specifier,
70             oboe=new(
71                 zaira.materials.wind_slap_music_specifier,
72                 pitch_handler__register_specifier__base_pitch='Bb3',
73                 ),
74             )
75
76
77 segment_maker.add_setting(
78     timespan_maker=new(
79         zaira.materials.sparse_timespan_maker,
80         playing_groupings=(1,),
81         ),
82         timespan_identifier=consort.RatioPartsExpression(

```

```

83     parts=(0, 2, 4),
84     ratio=(3, 1, 2, 1, 1),
85     mask_timespan=timespantools.Timespan(
86         start_offset=fanfare_duration,
87         ),
88     ),
89     clarinet=zaira.materials.wind_keyclick_music_specifier,
90     flute=zaira.materials.wind_keyclick_music_specifier,
91     oboe=zaira.materials.wind_keyclick_music_specifier,
92     )
93
94
95 ### PERCUSSION SETTINGS #####
96
97
98 segment_maker.add_setting(
99     timespan_maker=new(
100         zaira.materials.sparse_timespan_maker,
101         fuse_groups=True,
102         reflect=True,
103         ),
104         timespan_identifier=timespantools.Timespan(
105             start_offset=fanfare_duration,
106             ),
107         drums=new(
108             zaira.materials.drum_brushed_music_specifier,
109             pitch_handler__pitch_specifier=zaira.makers.Percussion.BASS_DRUM,
110             ),
111         )
112
113
114 ### PIANO SETTINGS #####
115
116
117 segment_maker.add_setting(
118     timespan_maker=new(
119         zaira.materials.sparse_timespan_maker,
120         playing_groupings=(1,),
121         playing_talea__counts=(5, 3, 3, 3, 6, 4, 3),
122         timespanSpecifier=consort.TimespanSpecifier(
123             minimum_duration=0,
124             ),
125             ),
126         timespan_identifier=consort.RatioPartsExpression(
127             parts=(1, 3, 5),
128             ratio=(1, 2, 1, 2, 1, 2, 1),
129             mask_timespan=timespantools.Timespan(
130                 start_offset=fanfare_duration,
131                 ),
132             ),
133         piano_rh=new(
134             zaira.materials.piano_flourish_music_specifier,
135             attachment_handler__dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
136             pitch_handler__register_specifier__base_pitch="c'",
```

```

137     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
138         operators=(
139             pitchtools.Inversion(),
140             pitchtools.Transposition(-3),
141             ),
142             ),
143             ),
144 piano_lh=new(
145     zaira.materials.piano_flourish_musicSpecifier,
146     attachment_handler__dynamic_expression=zaira.materials.background_dynamic_attachment_expression,
147     pitch_handler__registerSpecifier__base_pitch="c",
148     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
149         operators=(
150             pitchtools.Inversion(),
151             pitchtools.Transposition(-3),
152             ),
153             ),
154             ),
155         )
156
157
158 segment_maker.add_setting(
159     timespan_maker=zaira.materials.sparse_timespan_maker,
160     timespan_identifier=timespantools.Timespan(
161         start_offset=fanfare_duration,
162         ),
163     piano_rh=new(
164         zaira.materials.piano_prepared_treble_musicSpecifier,
165         rhythm_maker=zaira.materials.sustained_rhythm_maker,
166         ),
167     piano_lh=new(
168         zaira.materials.piano_prepared_bass_musicSpecifier,
169         rhythm_maker=zaira.materials.sustained_rhythm_maker,
170         ),
171     )
172
173
174 segment_maker.add_setting(
175     timespan_maker=zaira.materials.tutti_timespan_maker,
176     timespan_identifier=timespantools.Timespan(
177         start_offset=fanfare_duration,
178         ),
179     piano_rh=new(
180         zaira.materials.piano_clusters_musicSpecifier,
181         attachment_handler__dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
182         pitch_handler__registerSpecifier__base_pitch="g",
183         ),
184     piano_lh=new(
185         zaira.materials.piano_clusters_musicSpecifier,
186         attachment_handler__dynamic_expression=zaira.materials.erratic_dynamic_attachment_expression,
187         pitch_handler__registerSpecifier__base_pitch="c",
188         seed=1,
189         ),
190     )

```

```

191
192
193 segment_maker.add_setting(
194     timespan_maker=new(
195         zaira.materials.sparse_timespan_maker,
196         fuse_groups=True,
197         padding=durationtools.Duration(1, 4),
198         reflect=True,
199         ),
200     timespan_identifier=consort.RatioPartsExpression(
201         parts=(0, 2, 4),
202         ratio=(1, 2, 1, 2, 1),
203         ),
204     piano_rh=zaira.materials.piano_guero_music_specifier,
205     piano_lh=new(
206         zaira.materials.piano_guero_music_specifier,
207         seed=1,
208         ),
209     )
210
211
212 ### STRING SETTINGS #####
213
214
215 segment_maker.add_setting(
216     timespan_maker=zaira.materials.granular_timespan_maker,
217     timespan_identifier=timespantools.Timespan(
218         start_offset=fanfare_duration,
219         ),
220     violin=new(
221         zaira.materials.string_trills_music_specifier,
222         pitch_handler__registerSpecifier__base_pitch="c",
223         pitch_handler__pitchOperationSpecifier=pitchtools.PitchOperation(
224             pitchtools.Inversion(),
225             ),
226         seed=0,
227         ),
228     viola=new(
229         zaira.materials.string_trills_music_specifier,
230         pitch_handler__registerSpecifier__base_pitch='c',
231         pitch_handler__pitchOperationSpecifier=pitchtools.PitchOperation(
232             pitchtools.Inversion(),
233             ),
234         seed=1,
235         ),
236     cello=new(
237         zaira.materials.string_trills_music_specifier,
238         pitch_handler__registerSpecifier__base_pitch='c',
239         pitch_handler__pitchOperationSpecifier=pitchtools.PitchOperation(
240             pitchtools.Inversion(),
241             ),
242         seed=2,
243         ),
244     )

```

```

245
246
247 segment_maker.add_setting(
248     timespan_maker=new(
249         zaira.materials.sparse_timespan_maker,
250         padding=durationtools.Duration(1, 4),
251         reflect=True,
252     ),
253     timespan_identifier=timespantools.Timespan(
254         start_offset=fanfare_duration,
255     ),
256     violin=new(
257         zaira.materials.string_flourish_music_specifier,
258         pitch_handler__registerSpecifier__base_pitch="c",
259         pitch_handler__pitchOperation_specifier=pitchtools.PitchOperation(
260             pitchtools.Transposition(3),
261         ),
262         rhythm_maker__talea__denominator=16,
263         seed=0,
264     ),
265     viola=new(
266         zaira.materials.string_flourish_music_specifier,
267         pitch_handler__registerSpecifier__base_pitch="c",
268         pitch_handler__pitchOperation_specifier=pitchtools.PitchOperation(
269             pitchtools.Transposition(3),
270         ),
271         rhythm_maker__talea__denominator=16,
272         seed=1,
273     ),
274     cello=new(
275         zaira.materials.string_flourish_music_specifier,
276         pitch_handler__registerSpecifier__base_pitch='c',
277         pitch_handler__pitchOperation_specifier=pitchtools.PitchOperation(
278             pitchtools.Transposition(3),
279         ),
280         rhythm_maker__talea__denominator=16,
281         seed=2,
282     ),
283 )
284
285
286 segment_maker.add_setting(
287     timespan_maker=new(
288         zaira.materials.tutti_timespan_maker,
289         padding=durationtools.Duration(1, 4),
290         playing_groupings=(1,),
291         playing_talea__counts=(8,),
292         repeat=False,
293     ),
294     timespan_identifier=consort.RatioPartsExpression(
295         parts=(1, 3, 5),
296         ratio=(1, 2, 1, 1, 1, 2, 1),
297         mask_timespan=timespantools.Timespan(
298             start_offset=fanfare_duration,

```

```

299         ),
300     ),
301     violin=new(
302         zaira.materials.string_tutti_overpressure_music_specifier,
303         pitch_handler__pitchSpecifier='g',
304         seed=0,
305     ),
306     viola=new(
307         zaira.materials.string_tutti_overpressure_music_specifier,
308         pitch_handler__pitchSpecifier='c',
309         seed=1,
310     ),
311     cello=new(
312         zaira.materials.string_tutti_overpressure_music_specifier,
313         pitch_handler__pitchSpecifier='c',
314         seed=2,
315     ),
316 )
317
318
319 ##### SHAKER SETTINGS #####
320
321
322 segment_maker.add_setting(
323     timespan_maker=new(
324         zaira.materials.sparse_timespan_maker,
325         padding=durationtools.Duration(3, 8),
326     ),
327     timespan_identifier=timespantools.Timespan(
328         start_offset=fanfare_duration,
329     ),
330     clarinet=zaira.materials.brazil_nut_music_specifier,
331     flute=zaira.materials.brazil_nut_music_specifier,
332     violin=zaira.materials.brazil_nut_music_specifier,
333     viola=zaira.materials.brazil_nut_music_specifier,
334     woods=zaira.materials.wood_bamboo_music_specifier,
335 )
336
337
338 ##### FANFARE SETTINGS #####
339
340
341 segment_maker.add_setting(
342     timespan_maker=consort.FloodedTimespanMaker(),
343     timespan_identifier=timespantools.Timespan(
344         stop_offset=fanfare_duration,
345     ),
346     piano_rh=new(
347         zaira.materials.piano_fanfare_music_specifier,
348         pitch_handler__registerSpecifier__basePitch="g",
349     ),
350     piano_lh=new(
351         zaira.materials.piano_fanfare_music_specifier,
352         pitch_handler__logicalTie_expressions=

```

```

353         zaira.materials.piano_fanfare_music_specifier
354             .pitch_handler.logical_tie_expressions[:-1],
355             pitch_handler__pitchSpecifier="g c a f d f e b e",
356             pitch_handler__registerSpecifier__basePitch="g,,",
357             ),
358     drums=new(
359         zaira.materials.percussion_fanfare_music_specifier,
360         pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
361         ),
362     metals=new(
363         zaira.materials.percussion_fanfare_music_specifier,
364         pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
365         ),
366     )
367
368
369 ### DEPENDENT MUSIC SETTINGS #####
370
371
372 segment_maker.add_setting(
373     timespan_maker=zaira.materials.pedals_timespan_maker,
374     piano_pedals=zaira.materials.piano_pedals_music_specifier,
375     )

```

### B.3.10 ZAIRA SEGMENTS SEGMENT\_I

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import pitchtools
7 from abjad.tools import timespantools
8 import consort
9 import zaira
10
11
12 ### SEGMENT MAKER #####
13
14
15 segment_maker = zaira.makers.ZairaSegmentMaker(
16     tempo=indicatortools.Tempo((1, 4), 48),
17     )
18
19 ratio = mathtools.NonreducedRatio([1, 15])
20
21 segment_maker.desired_duration_in_seconds = (
22     durationtools.Multiplier(sum(ratio), 91) *
23     zaira.materials.total_duration_in_seconds
24     )
25
26 fanfare_duration = durationtools.Duration(1, 16)
27
28

```

```

29 ### WINDS SETTINGS #####
30
31
32 segment_maker.add_setting(
33     timespan_maker=new(
34         zaira.materials.tutti_timespan_maker,
35         playing_groupings=(1,),
36     ),
37     timespan_identifier=consort.RatioPartsExpression(
38         parts=(1, 3, 5),
39         ratio=(1, 2, 1, 2, 1, 2, 1),
40         mask_timespan=timespantools.Timespan(
41             start_offset=fanfare_duration,
42         ),
43     ),
44     clarinet=new(
45         zaira.materials.wind_airtone_music_specifier,
46         pitch_handler__registerSpecifier__base_pitch='D3',
47     ),
48     flute=zaira.materials.wind_airtone_music_specifier,
49     oboe=new(
50         zaira.materials.wind_airtone_music_specifier,
51         pitch_handler__registerSpecifier__base_pitch='Bb3',
52     ),
53 )
54
55
56 ### PERCUSSION SETTINGS #####
57
58
59 segment_maker.add_setting(
60     timespan_maker=new(
61         zaira.materials.sustained_timespan_maker,
62         fuse_groups=True,
63         timespanSpecifier=consort.TimespanSpecifier(
64             minimum_duration=durationtools.Duration(1, 4),
65         ),
66     ),
67     metals=new(
68         zaira.materials.percussion_superball_music_specifier,
69         pitch_handler__pitchSpecifier=zaira.makers.Percussion.TAM_TAM,
70         seed=2,
71     ),
72 )
73
74
75 segment_maker.add_setting(
76     timespan_maker=zaira.materials.sparse_timespan_maker,
77     timespan_identifier=consort.RatioPartsExpression(
78         parts=(1, 3),
79         ratio=(1, 1, 1, 1, 1),
80     ),
81     metals=new(
82         zaira.materials.percussion_brushed_music_specifier,

```

```

83     pitch_handler__pitch_specifier=zaira.makers.Percussion.BASS_DRUM,
84     seed=2,
85     ),
86   )
87
88
89 segment_maker.add_setting(
90   timespan_maker=zaira.materials.sparse_timespan_maker,
91   timespan_identifier=consort.RatioPartsExpression(
92     parts=(1, 3),
93     ratio=(2, 1, 2, 1, 1),
94     ),
95   drums=zaira.materials.drum_heartbeat_music_specifier,
96   )
97
98
99 ### PIANO SETTINGS #####
100
101
102 segment_maker.add_setting(
103   timespan_maker=new(
104     zaira.materials.sustained_timespan_maker,
105     fuse_groups=True,
106     reflect=True,
107     ),
108   piano_rh=zaira.materials.piano_drone_music_specifier,
109   )
110
111
112 ### STRING SETTINGS #####
113
114
115 segment_maker.add_setting(
116   timespan_maker=new(
117     zaira.materials.tutti_timespan_maker,
118     padding=durationtools.Duration(1, 4),
119     playing_groupings=(1,),
120     reflect=True,
121     ),
122   timespan_identifier=timespantools.Timespan(
123     start_offset=fanfare_duration,
124     ),
125   violin=new(
126     zaira.materials.string_undergrowth_music_specifier,
127     pitch_handler__registerSpecifier__base_pitch='g',
128     seed=0,
129     ),
130   viola=new(
131     zaira.materials.string_undergrowth_music_specifier,
132     pitch_handler__registerSpecifier__base_pitch='c',
133     seed=1,
134     ),
135   cello=new(
136     zaira.materials.string_undergrowth_music_specifier,

```

```

137     pitch_handler__register_specifier__base_pitch='c',
138     seed=2,
139     ),
140   )
141
142
143 segment_maker.add_setting(
144   timespan_maker=new(
145     zaira.materials.granular_timespan_maker,
146     playing_talea__counts=(2, 2, 3, 2, 7, 1, 3, 2, 1),
147     ),
148   timespan_identifier=consort.RatioPartsExpression(
149     parts=(0, 2, 4),
150     ratio=(1, 1, 1, 1, 1),
151     mask_timespan=timespantools.Timespan(
152       start_offset=fanfare_duration,
153       ),
154     ),
155   cello=new(
156     zaira.materials.cello_solo_music_specifier,
157     pitch_handler__pitch_operation_specifier=pitchtools.PitchOperation(
158       pitchtools.Transposition(3),
159       ),
160     ),
161   )
162
163
164 ### SHAKER SETTINGS #####
165
166
167 segment_maker.add_setting(
168   timespan_maker=new(
169     zaira.materials.tutti_timespan_maker,
170     playing_talea__counts=(3, 2, 3, 3, 2, 4),
171     playing_groupings=(1,),
172     padding=durationtools.Duration(3, 8),
173     ),
174   clarinet=zaira.materials.brazil_nut_music_specifier,
175   flute=zaira.materials.brazil_nut_music_specifier,
176   violin=zaira.materials.brazil_nut_music_specifier,
177   viola=zaira.materials.brazil_nut_music_specifier,
178   woods=zaira.materials.wood_bamboo_music_specifier,
179   )
180
181
182 ### FANFARE SETTINGS #####
183
184
185 segment_maker.add_setting(
186   timespan_maker=consort.FloodedTimespanMaker(),
187   timespan_identifier=timespantools.Timespan(
188     stop_offset=fanfare_duration,
189     ),
190   piano_rh=new(

```

```

191     zaira.materials.piano_fanfare_music_specifier,
192     pitch_handler__registerSpecifier__basePitch="g",
193   ),
194   piano_lh=new(
195     zaira.materials.piano_fanfare_music_specifier,
196     pitchHandler__logicalTieExpressions=
197       zaira.materials.piano_fanfare_music_specifier
198         .pitchHandler.logicalTieExpressions[:-1],
199     pitchHandler__pitchSpecifier="g c a f d f e b e",
200     pitchHandler__registerSpecifier__basePitch="g,,",
201   ),
202   drums=new(
203     zaira.materials.percussion_fanfare_music_specifier,
204     pitchHandler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
205   ),
206   metals=new(
207     zaira.materials.percussion_fanfare_music_specifier,
208     pitchHandler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
209   ),
210 )
211
212
213 ### DEPENDENT MUSIC SETTING #####
214
215
216 segment_maker.add_setting(
217   timespan_maker=zaira.materials.pedals_timespan_maker,
218   piano_pedals=zaira.materials.piano_pedals_music_specifier,
219 )

```

### B.3.11 ZAIRA.SEGMENTS SEGMENT\_J

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import timespantools
7 import consort
8 import zaira
9
10
11 ##### SEGMENT MAKER #####
12
13
14 segment_maker = zaira.makers.ZairaSegmentMaker(
15   permitted_time_signatures=(
16     (2, 4),
17     (3, 8),
18   ),
19   tempo=indicatortools.Tempo((1, 4), 84),
20 )
21
22 ratio = mathtools.NonreducedRatio([2])

```

```

23
24 segment_maker.desired_duration_in_seconds = (
25     durationtools.Multiplier(sum(ratio), 91) *
26     zaira.materials.total_duration_in_seconds
27 )
28
29 fanfare_duration = durationtools.Duration(2, 16)
30
31
32 ### PERCUSSION SETTINGS #####
33
34
35 segment_maker.add_setting(
36     timespan_maker=new(
37         zaira.materials.sustained_timespan_maker,
38         fuse_groups=True,
39     ),
40     drums=new(
41         zaira.materials.percussion_superball_music_specifier,
42         pitch_handler__pitch_specifier=zaira.makers.Percussion.BASS_DRUM,
43     ),
44     metals=new(
45         zaira.materials.percussion_superball_music_specifier,
46         pitch_handler__pitch_specifier=zaira.makers.Percussion.TAM_TAM,
47     ),
48 )
49
50
51 ### PIANO SETTINGS #####
52
53
54 segment_maker.add_setting(
55     timespan_maker=zaira.materials.sparse_timespan_maker,
56     piano_rh=new(
57         zaira.materials.piano_prepared_treble_music_specifier,
58         rhythm_maker=zaira.materials.sustained_rhythm_maker,
59     ),
60     piano_lh=new(
61         zaira.materials.piano_prepared_bass_music_specifier,
62         rhythm_maker=zaira.materials.sustained_rhythm_maker,
63     ),
64 )
65
66
67 ### SHAKER SETTINGS #####
68
69
70 segment_maker.add_setting(
71     timespan_maker=consort.FloodedTimespanMaker(),
72     clarinet=new(
73         zaira.materials.brazil_nut_music_specifier,
74         rhythm_maker=zaira.materials.sustained_rhythm_maker,
75     ),
76     flute=new(

```

```

77     zaira.materials.brazil_nut_music_specifier,
78     rhythm_maker=zaira.materials.sustained_rhythm_maker,
79     ),
80 violin=new(
81     zaira.materials.brazil_nut_music_specifier,
82     rhythm_maker=zaira.materials.sustained_rhythm_maker,
83     ),
84 viola=new(
85     zaira.materials.brazil_nut_music_specifier,
86     rhythm_maker=zaira.materials.sustained_rhythm_maker,
87     ),
88 )
89
90
91 ##### FANFARE SETTINGS #####
92
93
94 segment_maker.add_setting(
95     timespan_maker=consort.FloodedTimespanMaker(),
96     timespan_identifier=timespantools.Timespan(
97         stop_offset=fanfare_duration,
98     ),
99 piano_rh=new(
100     zaira.materials.piano_fanfare_music_specifier,
101     pitch_handler__registerSpecifier__basePitch="g",
102     ),
103 piano_lh=new(
104     zaira.materials.piano_fanfare_music_specifier,
105     pitchHandler__logicalTieExpressions=
106         zaira.materials.piano_fanfare_music_specifier
107             .pitchHandler.logicalTieExpressions[:-1],
108     pitchHandler__pitchSpecifier="g c a f d f e b e",
109     pitchHandler__registerSpecifier__basePitch="g,,",
110     ),
111 drums=new(
112     zaira.materials.percussion_fanfare_music_specifier,
113     pitchHandler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
114     ),
115 metals=new(
116     zaira.materials.percussion_fanfare_music_specifier,
117     pitchHandler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
118     ),
119 )
120
121
122 ##### DEPENDENT MUSIC SETTINGS #####
123
124
125 segment_maker.add_setting(
126     timespan_maker=zaira.materials.pedals_timespan_maker,
127     piano_pedals=zaira.materials.piano_pedals_music_specifier,
128     )

```

### B.3.12 ZAIRA.SEGMENTS.SEGMENT\_K

```
1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import mathtools
6 from abjad.tools import timespantools
7 import consort
8 import zaira
9
10
11 ### SEGMENT MAKER #####
12
13
14 segment_maker = zaira.makers.ZairaSegmentMaker(
15     permitted_time_signatures=(
16         (3, 8),
17         (4, 8),
18     ),
19     tempo=indicatortools.Tempo((1, 4), 96),
20 )
21
22 ratio = mathtools.NonreducedRatio([3])
23
24 segment_maker.desired_duration_in_seconds = (
25     durationtools.Multiplier(sum(ratio), 91) *
26     zaira.materials.total_duration_in_seconds
27 )
28
29
30 ### FANFARE SETTINGS #####
31
32
33 segment_maker.add_setting(
34     timespan_maker=consort.FloodedTimespanMaker(),
35     timespan_identifier=timespantools.Timespan(
36         stop_offset=durationtools.Duration(3, 8),
37     ),
38     piano_rh=new(
39         zaira.materials.piano_fanfare_music_specifier,
40         pitch_handler__registerSpecifier__base_pitch="g",
41         rhythm_maker__denominators=[8],
42     ),
43     piano_lh=new(
44         zaira.materials.piano_fanfare_music_specifier,
45         pitch_handler__logical_tie_expressions=
46             zaira.materials.piano_fanfare_music_specifier
47                 .pitchHandler.logical_tie_expressions[:-1],
48         pitch_handler__pitchSpecifier="g c a f d f e b e",
49         pitch_handler__registerSpecifier__basePitch="g,,",
50         rhythm_maker__denominators=[8],
51     ),
52     drums=new(
```

```

53     zaira.materials.percussion_fanfare_music_specifier,
54     pitch_handler__pitchSpecifier=zaira.makers.Percussion.KICK_DRUM,
55     rhythm_maker__denominators=[8],
56   ),
57   metals=new(
58     zaira.materials.percussion_fanfare_music_specifier,
59     pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
60     rhythm_maker__denominators=[8],
61   ),
62 )
63
64
65 ### PERCUSSION SETTINGS #####
66
67
68 segment_maker.add_setting(
69   timespan_maker=consort.FloodedTimespanMaker(),
70   timespan_identifier=timespantools.Timespan(
71     start_offset=durationtools.Duration(5, 16),
72   ),
73   metals=new(
74     zaira.materials.percussion_reiteration_music_specifier,
75     pitch_handler__pitchSpecifier=zaira.makers.Percussion.BRAKE_DRUM,
76   ),
77 )
78
79
80 ### DEPENDENT MUSIC SETTINGS #####
81
82
83 segment_maker.add_setting(
84   timespan_maker=zaira.materials.pedals_timespan_maker,
85   piano_pedals=zaira.materials.piano_pedals_music_specifier,
86 )

```

## B.4 *ZAIRA*

### STYLESHEET

SOURCE

#### B.4.1 STYLESHEET.ILY

```

1 #(define-markup-command (vstrut layout props)
2   ())
3   #:category other
4   "
5 @cindex creating vertical space in text
6
7 Create a box of the same height as the current font."
8 (let ((ref-mrkp (interpret-markup layout props "fp")))
9   (ly:make-stencil (ly:stencil-expr empty-stencil)
10     empty-interval
11     (ly:stencil-extent ref-mrkp Y))))
12
13 afterGraceFraction = #(cons 127 128)
14 #(set-global-staff-size 12)
15

```

```

16 \layout {
17
18     ragged-right = ##t
19
20     %%% COMMON %%%
21
22     \context {
23         \Voice
24             \consists Horizontal_bracket_engraver
25             \remove Forbid_line_break_engraver
26     }
27
28     \context {
29         \Staff
30             \remove Time_signature_engraver
31     }
32
33     \context {
34         \Dynamics
35             \remove Bar_engraver
36     }
37
38     %%% TIME SIGNATURE CONTEXT %%%
39
40     \context {
41         \name TimeSignatureContext
42         \type Engraver_group
43         \consists Axis_group_engraver
44         \consists Bar_number_engraver
45         \consists Mark_engraver
46         \consists Metronome_mark_engraver
47         \consists Time_signature_engraver
48
49         \override BarNumber.X-extent = #'(0 . 0)
50         \override BarNumber.Y-extent = #'(0 . 0)
51         \override BarNumber.extra-offset = #'(-8 . -4)
52         \override BarNumber.font-name = "Didot Italic"
53         \override BarNumber.font-size = 2
54         \override BarNumber.stencil = #(make-stencil-circler 0.1 0.7 ly:text-interface::print)
55
56         \override MetronomeMark.X-extent = #'(0 . 0)
57         \override MetronomeMark.X-offset = 5
58         \override MetronomeMark.Y-offset = -2.5
59         \override MetronomeMark.break-align-symbols = #'(time-signature)
60         \override MetronomeMark.font-size = 3
61
62         \override RehearsalMark.X-extent = #'(0 . 0)
63         \override RehearsalMark.Y-offset = 8
64         \override RehearsalMark.break-align-symbols = #'(time-signature)
65         \override RehearsalMark.break-visibility = #end-of-line-invisible
66         \override RehearsalMark.font-name = "Didot"
67         \override RehearsalMark.font-size = 10
68         \override RehearsalMark.outside-staff-priority = 500
69         \override RehearsalMark.self-alignment-X = #CENTER

```

```

70
71     \override TimeSignature.X-extent = #'(0 . 0)
72     \override TimeSignature.break-align-symbols = #'(staff-bar)
73     \override TimeSignature.break-visibility = #end-of-line-invisible
74     \override TimeSignature.font-size = 3
75     \override TimeSignature.style = #'numbered
76
77     \override VerticalAxisGroup.staff-staff-spacing = #'(
78         (basic-distance . 8)
79         (minimum-distance . 8)
80         (padding . 8)
81         (stretchability . 0)
82     )
83 }
84
85 %%% WINDS %%%
86
87 \context {
88     \Staff
89     \name FluteStaff
90     \type Engraver_group
91     \alias Staff
92 }
93
94 \context {
95     \Staff
96     \name ClarinetInBFlatStaff
97     \type Engraver_group
98     \alias Staff
99 }
100
101 \context {
102     \Staff
103     \name OboeStaff
104     \type Engraver_group
105     \alias Staff
106 }
107
108 %%% DRUMS %%%
109
110 \context {
111     \Staff
112     \name MetalsStaff
113     \type Engraver_group
114     \alias Staff
115 }
116
117 \context {
118     \Staff
119     \name WoodsStaff
120     \type Engraver_group
121     \alias Staff
122     \override BarLine.bar-extent = #'(-1 . 1)
123     \override StaffSymbol.line-count = 3

```

```

124     }
125
126     \context {
127         \Staff
128         \name DrumsStaff
129         \type Engraver_group
130         \alias Staff
131         \override BarLine.bar-extent = #'(-1 . 1)
132         \override StaffSymbol.line-count = 3
133     }
134
135     %%% PIANO %%%
136
137     \context {
138         \Staff
139         \name PianoUpperStaff
140         \type Engraver_group
141         \alias Staff
142     }
143
144     \context {
145         \Staff
146         \name PianoLowerStaff
147         \type Engraver_group
148         \alias Staff
149     }
150
151     \context {
152         \PianoStaff
153         \name PianoPerformerGroup
154         \type Engraver_group
155         \alias PianoStaff
156         \accepts Staff
157         \accepts PianoUpperStaff
158         \accepts PianoLowerStaff
159         \override SystemStartBracket.stencil = ##f
160     }
161
162     \context {
163         \PianoStaff
164         \accepts PianoUpperStaff
165         \accepts PianoLowerStaff
166     }
167
168     %%% STRINGS %%%
169
170     \context {
171         \Staff
172         \name StringStaff
173         \type Engraver_group
174         \alias Staff
175     }
176
177     \context {

```

```

178     \StaffGroup
179     \name StringPerformerGroup
180     \type Engraver_group
181     \alias StaffGroup
182     \accepts BowingStaff
183     \accepts FingeringStaff
184     \accepts StringStaff
185 }
186
187 %% ANNOTATIONS %%
188
189 \context {
190     \Voice
191     \name InnerAnnotation
192     \type Engraver_group
193     \alias Voice
194     \override Accidental.stencil = ##f
195     \override Dots.stencil = ##f
196     \override Flag.stencil = ##f
197     \override NoteColumn.ignore-collision = ##t
198     \override NoteHead.no-ledgers = ##t
199     \override NoteHead.stencil = ##f
200     \override Stem.stencil = ##f
201     \override TupleBracket.dash-fraction = 0.125
202     \override TupleBracket.dash-period = 1.0
203     \override TupleBracket.outside-staff-padding = 1
204     \override TupleBracket.outside-staff-priority = 999
205     \override TupleBracket.style = #'dashed-line
206     \override TupleNumber.stencil = ##f
207 }
208
209 \context {
210     \Voice
211     \name OuterAnnotation
212     \type Engraver_group
213     \alias Voice
214     \override Accidental.stencil = ##f
215     \override Dots.stencil = ##f
216     \override Flag.stencil = ##f
217     \override NoteHead.no-ledgers = ##t
218     \override NoteHead.stencil = ##f
219     \override Stem.stencil = ##f
220     \override TupleNumber.stencil = ##f
221     \override TupleBracket.outside-staff-padding = 1
222     \override TupleBracket.outside-staff-priority = 1000
223     \override NoteColumn.ignore-collision = ##t
224 }
225
226 \context {
227     \Staff
228     \accepts InnerAnnotation
229     \accepts OuterAnnotation
230 }
231

```

```

232     %%% SINGLE PERFORMER GROUP %%%
233
234     \context {
235         \StaffGroup
236         \name PerformerGroup
237         \type Engraver_group
238         \alias StaffGroup
239         \accepts Staff
240         \accepts FluteStaff
241         \accepts OboeStaff
242         \accepts ClarinetInBFlatStaff
243         \accepts DrumsStaff
244         \accepts WoodsStaff
245         \accepts MetalsStaff
246         \accepts StringStaff
247     }
248
249     %%% MULTIPLE PERFORMER GROUP %%%
250
251     \context {
252         \StaffGroup
253         \name EnsembleGroup
254         \type Engraver_group
255         \alias StaffGroup
256         \accepts PerformerGroup
257         \accepts StringPerformerGroup
258         \accepts PianoPerformerGroup
259     }
260
261     %%% SCORE %%%
262
263     \context {
264         \Score
265         \accepts TimeSignatureContext
266         \accepts PerformerGroup
267         \accepts EnsembleGroup
268         \remove Metronome_mark_engraver
269         \remove Mark_engraver
270         \remove Bar_number_engraver
271         \override BarLine.bar-extent = #'(-2 . 2)
272         \override BarLine.hair-thickness = 0.5
273         \override BarLine.space-alist = #'(
274             (time-signature extra-space . 0.0)
275             (custos minimum-space . 0.0)
276             (clef minimum-space . 0.0)
277             (key-signature extra-space . 0.0)
278             (key-cancellation extra-space . 0.0)
279             (first-note fixed-space . 0.0)
280             (next-note semi-fixed-space . 0.0)
281             (right-edge extra-space . 0.0)
282         )
283         \override Beam.beam-thickness = 0.75
284         \override Beam.breakable = ##t
285         \override Beam.length-fraction = 1.5

```

```

286 \override DynamicLineSpanner.add-stem-support = ##t
287 \override DynamicLineSpanner.outside-staff-padding = 1
288 \override Glissando.breakable = ##t
289 \override Glissando.thickness = 3
290 \override GraceSpacing.common-shortest-duration = #(ly:make-moment 1 16)
291 \override NoteCollision.merge-differently-dotted = ##t
292 \override NoteColumn.ignore-collision = ##t
293 \override OttavaBracket.add-stem-support = ##t
294 \override OttavaBracket.padding = 2
295 \override SpacingSpanner.base-shortest-duration = #(ly:make-moment 1 64)
296 \override SpacingSpanner.strict-grace-spacing = ##f
297 \override SpacingSpanner.strict-note-spacing = ##f
298 \override SpacingSpanner.uniform-stretching = ##t
299 \override Stem.details.beamed-lengths = #'(6)
300 \override Stem.details.lengths = #'(6)
301 \override Stem.stemlet-length = 1.5
302 \override StemTremolo.beam-width = 1.5
303 \override StemTremolo.flag-count = 4
304 \override StemTremolo.slope = 0.5
305 \override StemTremolo.style = #'default
306 \override SustainPedal.self-alignment-X = #CENTER
307 \override SustainPedalLineSpanner.padding = 2
308 \override SustainPedalLineSpanner.to-barline = ##t
309 \override TextScript.add-stem-support = ##t
310 \override TextScript.outside-staff-padding = 1
311 \override TextScript.padding = 1
312 \override TextScript.staff-padding = 1
313 \override TextSpanner.bound-details.right.padding = 2.5
314 \override TrillPitchAccidental.avoid-slur = #'ignore
315 \override TrillPitchAccidental.layer = 1000
316 \override TrillPitchAccidental.whiteout = ##t
317 \override TrillPitchHead.layer = 1000
318 \override TrillPitchHead.whiteout = ##t
319 \override TrillSpanner.outside-staff-padding = 1
320 \override TrillSpanner.padding = 1
321 \override TupletBracket.avoid-scripts = ##t
322 \override TupletBracket.full-length-to-extent = ##t
323 \override TupletBracket.outside-staff-padding = 2
324 \override TupletBracket.padding = 2
325 \override TupletNumber.font-size = 1
326 \override TupletNumber.text = #tuplet-number::calc-fraction-text
327 \override VerticalAxisGroup.staff-staff-spacing = #'(
328     (basic-distance . 8)
329     (minimum-distance . 14)
330     (padding . 4)
331     (stretchability . 0)
332 )
333 autoBeaming = ##f
334 pedalSustainStyle = #'mixed
335 proportionalNotationDuration = #(ly:make-moment 1 32)
336 tupletFullLength = ##t
337 }
338
339 }
```

```

340
341 \paper {
342
343     %%% MARGINS %%%
344
345     % bottom-margin = 10\mm
346     left-margin = 30\mm
347     right-margin = 10\mm
348     top-margin = 10\mm
349
350     %%% HEADERS AND FOOTERS %%%
351
352     evenFooterMarkup = \markup \fill-line {
353         \concat {
354             \bold \fontsize #3
355             \on-the-fly #print-page-number-check-first
356             \fromproperty #'page:page-number-string
357             \%hspace #18
358         }
359         " "
360     }
361     evenHeaderMarkup = \markup \fill-line { " " }
362     oddFooterMarkup = \markup \fill-line {
363         " "
364         \concat {
365             \bold \fontsize #3
366             \on-the-fly #print-page-number-check-first
367             \fromproperty #'page:page-number-string
368             \%hspace #18
369         }
370     }
371     oddHeaderMarkup = \markup \fill-line { " " }
372     print-first-page-number = ##f
373     print-page-number = ##t
374
375     %%% PAGE BREAKING %%%
376
377     page-breaking = #ly:optimal-breaking
378     ragged-bottom = ##f
379     ragged-last-bottom = ##f
380
381     %%% SPACING DETAILS %%%
382
383     markup-system-spacing = #'(
384         (basic-distance . 0)
385         (minimum-distance . 12)
386         (padding . 0)
387         (stretchability . 0)
388     )
389     system-system-spacing = #'(
390         (basic-distance . 8)
391         (minimum-distance . 12)
392         (padding . 4)
393         (stretchability . 0)

```

```
394    )
395    top-markup-spacing = #'(
396      (basic-distance . 0)
397      (minimum-distance . 0)
398      (padding . 8)
399      (stretchability . 0)
400    )
401    top-system-spacing = #'(
402      (basic-distance . 0)
403      (minimum-distance . 10)
404      (padding . 0)
405      (stretchability . 0)
406    )
407
408    %% ETC %%
409
410    % system-separator-markup = \slashSeparator
411
412 }
```

This page intentionally left blank.

# C

*armilla* source code

## C.1 *ARMILLA*

MAKERS

SOURCE

### C.1.1 ARMILLA.MAKERS.ARMILLASCORETEMPLATE

```
1 # -*- encoding: utf-8 -*-
2 from abjad import attach
3 from abjad import indicatortools
4 from abjad import instrumenttools
5 from abjad import markuptools
6 from abjad import scoretools
7 import consort
8
9
10 class ArmillaScoreTemplate(scoretools.ScoreTemplate):
11     r'''A score template.
12
13     :::
14
15     >>> import armilla
16     >>> template = armilla.makers.ArmillaScoreTemplate()
17     >>> score = template()
18     >>> print(format(score))
19     \context Score = "Armilla Score" <-
20         \tag #'time
21         \context TimeSignatureContext = "Time Signature Context" {
22             }
23             \tag #'viola-1
24             \context StringPerformerGroup = "Viola 1 Performer Group" \with {
25                 instrumentName = \markup {
26                     \hcenter-in
27                         #10
28                         "Viola 1"
```

```

29         }
30     shortInstrumentName = \markup {
31         \hcenter-in
32             #10
33             "Va. 1"
34         }
35     } <<
36     \context BowingStaff = "Viola 1 Bowing Staff" {
37         \clef "percussion"
38         \context Voice = "Viola 1 Bowing Voice" {
39             }
40         }
41         \context FingeringStaff = "Viola 1 Fingering Staff" {
42             \clef "alto"
43             \context Voice = "Viola 1 Fingering Voice" {
44                 }
45             }
46     >>
47     \tag #'viola-2
48     \context StringPerformerGroup = "Viola 2 Performer Group" \with {
49         instrumentName = \markup {
50             \hcenter-in
51                 #10
52                 "Viola 2"
53             }
54         shortInstrumentName = \markup {
55             \hcenter-in
56                 #10
57                 "Va. 2"
58             }
59     } <<
60         \context BowingStaff = "Viola 2 Bowing Staff" {
61             \clef "percussion"
62             \context Voice = "Viola 2 Bowing Voice" {
63                 }
64             }
65             \context FingeringStaff = "Viola 2 Fingering Staff" {
66                 \clef "alto"
67                 \context Voice = "Viola 2 Fingering Voice" {
68                     }
69                 }
70             >>
71     >>
72 ::

73 >>> for item in sorted(template.context_name_abbreviations.items()):
74 ...     item
75 ...
76 ('viola_1', 'Viola 1 Performer Group')
77 ('viola_1_lh', 'Viola 1 Fingering Voice')
78 ('viola_1_rh', 'Viola 1 Bowing Voice')
79 ('viola_2', 'Viola 2 Performer Group')
80 ('viola_2_lh', 'Viola 2 Fingering Voice')

```

```

83     ('viola_2_rh', 'Viola 2 Bowing Voice')
84
85     :::
86
87     >>> for item in sorted(template.composite_context_pairs.items()):
88         ...     item
89         ...
90         ('viola_1', ('viola_1_rh', 'viola_1_lh'))
91         ('viola_2', ('viola_2_rh', 'viola_2_lh'))
92
93     """
94
95     ### CLASS VARIABLES ###
96
97     __slots__ = ()
98
99     ### SPECIAL METHODS ###
100
101    def __call__(self):
102
103        manager = consort.ScoreTemplateManager
104
105        time_signature_context = manager.make_time_signature_context()
106
107        viola_one = manager.make_single_string_performer(
108            clef=indicatortools.Clef('alto'),
109            instrument=instrumenttools.Viola(
110                instrument_name='viola 1',
111                instrument_name_markup=markuptools.Markup(
112                    'Viola 1').hcenter_in(10),
113                short_instrument_name='va. 1',
114                short_instrument_name_markup=markuptools.Markup(
115                    'Va. 1').hcenter_in(10)
116                ),
117                split=True,
118                score_template=self,
119            )
120
121        viola_two = manager.make_single_string_performer(
122            clef=indicatortools.Clef('alto'),
123            instrument=instrumenttools.Viola(
124                instrument_name='viola 2',
125                instrument_name_markup=markuptools.Markup(
126                    'Viola 2').hcenter_in(10),
127                short_instrument_name='va. 2',
128                short_instrument_name_markup=markuptools.Markup(
129                    'Va. 2').hcenter_in(10)
130                ),
131                split=True,
132                score_template=self,
133            )
134
135        score = scoretools.Score(
136            [

```

```

137     time_signature_context,
138     viola_one,
139     viola_two,
140     ],
141     name='Armilla Score',
142 )
143
144 attach(
145     indicatortools.Tuning(pitches=('C3', 'G3', 'D4', 'A4')),
146     score['Viola 1 Fingering Staff'],
147     scope=scoretools.Voice,
148 )
149 attach(
150     indicatortools.Tuning(pitches=('C3', 'G3', 'D4', 'A4')),
151     score['Viola 2 Fingering Staff'],
152     scope=scoretools.Voice,
153 )
154
155 return score

```

## C.1.2 ARMILLA.MAKERS.ARMILLASEGMENTMAKER

```

1 # -*- encoding: utf-8 -*-
2 from abjad import attach
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import markuptools
6 from abjad.tools import scoretools
7 import consort
8
9
10 class ArmillaSegmentMaker(consort.SegmentMaker):
11     r'''Armilla segment maker.
12
13     :::
14
15     >>> import armilla
16     >>> segment_maker = armilla.ArmillaSegmentMaker()
17     >>> print(format(segment_maker))
18     armilla.makers.ArmillaSegmentMaker(
19         permitted_time_signatures=indicatortools.TimeSignatureInventory(
20             [
21                 indicatortools.TimeSignature((2, 4)),
22                 indicatortools.TimeSignature((3, 4)),
23                 indicatortools.TimeSignature((3, 8)),
24                 indicatortools.TimeSignature((4, 8)),
25                 indicatortools.TimeSignature((5, 8)),
26                 indicatortools.TimeSignature((6, 8)),
27             ]
28         ),
29         score_template=armilla.makers.ArmillaScoreTemplate(),
30         timespan_quantization=durationtools.Duration(1, 8),
31     )
32

```

```

33     """
34
35     """ CLASS VARIABLES """
36
37     __slots__ = (
38         '_repeat',
39     )
40
41     """ INITIALIZER """
42
43     def __init__(
44         self,
45         annotate_colors=None,
46         annotate_phrasing=None,
47         annotate_timespans=None,
48         desired_duration_in_seconds=None,
49         discard_final_silence=None,
50         maximum_meter_run_length=None,
51         name=None,
52         omit_stylesheets=None,
53         permitted_time_signatures=None,
54         repeat=None,
55         score_template=None,
56         settings=None,
57         tempo=None,
58         timespan_quantization=None,
59     ):
60         import armilla
61         permitted_time_signatures = (
62             permitted_time_signatures or
63             armilla.materials.time_signatures
64         )
65         score_template = (
66             score_template or
67             armilla.makers.ArmillaScoreTemplate()
68         )
69         timespan_quantization = (
70             timespan_quantization or
71             durationtools.Duration(1, 8),
72         )
73         consort.SegmentMaker.__init__(
74             self,
75             annotate_colors=annotate_colors,
76             annotate_phrasing=annotate_phrasing,
77             annotate_timespans=annotate_timespans,
78             desired_duration_in_seconds=desired_duration_in_seconds,
79             discard_final_silence=discard_final_silence,
80             maximum_meter_run_length=maximum_meter_run_length,
81             name=name,
82             omit_stylesheets=omit_stylesheets,
83             permitted_time_signatures=permitted_time_signatures,
84             repeat=repeat,
85             score_template=score_template,
86             settings=settings,

```

```

87         tempo=tempo,
88         timespan_quantization=timespan_quantization,
89     )
90     self.repeat = repeat
91
92     ##### PUBLIC METHODS #####
93
94     def postprocess_breath_marks(self, score):
95         breath_mark = indicatortools.BreathMark()
96         leaves = score['Viola 1 Bowing Voice'].select_leaves()
97         if isinstance(leaves[-1], scoretools.Note):
98             attach(breath_mark, leaves[-1], name='breath_mark')
99         leaves = score['Viola 2 Bowing Voice'].select_leaves()
100        if isinstance(leaves[-1], scoretools.Note):
101            attach(breath_mark, leaves[-1], name='breath_mark')
102
103    def postprocess_left_hand_staff(self, staff):
104        voice = staff[0]
105        finger_pitches_voice = self.copy_voice(
106            voice,
107            attachment_names=(
108                'clef_spanner',
109                'staff_lines_spanner',
110                'trill_spanner',
111            ),
112            new_voice_name=voice.name.replace('Fingering', 'LH Pitches'),
113            new_context_name='FingeringPitchesVoice',
114        )
115        finger_spanner_voice = self.copy_voice(
116            voice,
117            attachment_names=(
118                'bend_after',
119                'glissando',
120            ),
121            new_voice_name=voice.name.replace('Fingering', 'LH Spanner'),
122            new_context_name='FingeringSpannerVoice',
123            remove_grace_containers=True,
124            remove_ties=True,
125            replace_rests_with_skips=True,
126        )
127        voice_index = staff.index(voice)
128        staff[voice_index:voice_index + 1] = [
129            finger_pitches_voice,
130            finger_spanner_voice
131        ]
132        staff.is_simultaneous = True
133
134    def postprocess_right_hand_staff(self, staff):
135        voice = staff[0]
136        string_contact_voice = self.copy_voice(
137            voice,
138            attachment_names=(
139                'string_contact',
140            ),

```

```

141     new_voice_name=voice.name.replace('Bowing', 'RH String Contact'),
142     new_context_name='StringContactVoice',
143     remove_ties=True,
144     replace_rests_with_skips=True,
145     )
146 bow_contact_voice = self.copy_voice(
147     voice,
148     attachment_names=(
149         'articulations',
150         'bow_contact_spanner',
151         'bow_contact_point',
152         'bow_motion_technique',
153         'breath_mark',
154         ),
155     new_voice_name=voice.name.replace('Bowing', 'RH Bow Contact'),
156     new_context_name='BowContactVoice',
157     remove_ties=True,
158     replace_rests_with_skips=True,
159     )
160 bow_beaming_voice = self.copy_voice(
161     voice,
162     attachment_names=(
163         'beam',
164         'stem_tremolo',
165         ),
166     new_voice_name=voice.name.replace('Bowing', 'RH Beaming'),
167     new_context_name='BowBeamingVoice',
168     remove_ties=True,
169     )
170 bow_dynamics_voice = self.copy_voice(
171     voice,
172     attachment_names=(
173         'dynamic_expression',
174         ),
175     new_voice_name=voice.name.replace('Bowing', 'RH Dynamics'),
176     new_context_name='Dynamics',
177     remove_ties=True,
178     replace_rests_with_skips=True,
179     )
180 voice_index = staff.index(voice)
181 staff[voice_index:voice_index + 1] = [
182     string_contact_voice,
183     bow_contact_voice,
184     bow_beaming_voice,
185     bow_dynamics_voice,
186     ]
187 staff.is_simultaneous = True
188
189 def configure_score(self):
190     self.postprocess_breath_marks(self.score)
191     self.postprocess_right_hand_staff(self.score['Viola 1 Bowing Staff'])
192     self.postprocess_right_hand_staff(self.score['Viola 2 Bowing Staff'])
193     self.postprocess_left_hand_staff(self.score['Viola 1 Fingering Staff'])
194     self.postprocess_left_hand_staff(self.score['Viola 2 Fingering Staff'])

```

```

195     consort.SegmentMaker.configure_score(self)
196
197     ### PUBLIC PROPERTIES ###
198
199     @property
200     def final_markup(self):
201         portland = markuptools.Markup('Portland, OR')
202         queens = markuptools.Markup('Fresh Meadows, NY')
203         date = markuptools.Markup('September 2014 - January 2015')
204         null = markuptools.Markup.null()
205         contents = [
206             null,
207             null,
208             null,
209             portland,
210             queens,
211             date,
212         ]
213         markup = markuptools.Markup.right_column(contents)
214         markup = markup.italic()
215         return markup
216
217     @property
218     def repeat(self):
219         return self._repeat
220
221     @repeat.setter
222     def repeat(self, repeat):
223         if repeat is not None:
224             repeat = bool(repeat)
225         self._repeat = repeat
226
227     @property
228     def score_package_name(self):
229         return 'armilla'

```

## C.2 *ARMILLA*

### MATERIALS

SOURCE

#### C.2.1 ARMILLA.MATERIALS.DENSE\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4
5
6 dense_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=rhythmmakertools.Talea(
8         counts=(1, 0),
9         denominator=8,
10        ),
11     playing_talea=rhythmmakertools.Talea(
12         counts=(5, 7, 4, 5),
13         denominator=8,
14        ),

```

```

15     playing_groupings=(3, 4, 2, 4),
16     silence_talea=rhythmmakertools.Talea(
17         counts=(1, 1, 1, 2, 1, 1, 2),
18         denominator=8,
19         ),
20     )

```

### C.2.2 ARMILLA.MATERIALS.INTERMITTENT\_ACCENTS

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import selectortools
5
6
7 intermittent_accents = consort.AttachmentExpression(
8     attachments=indicatortools.Articulation('>', 'down'),
9     selector=selectortools.Selector().by_leaves()[:-1].by_counts(
10         [-3, 1, -2, 2, -1, 3, -4, 5],
11         ).flatten(),
12     )

```

### C.2.3 ARMILLA.MATERIALS.INTERMITTENT\_CIRCULAR

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import selectortools
5
6
7 intermittent_circular = consort.AttachmentExpression(
8     attachments=indicatortools.BowMotionTechnique('circular'),
9     selector=selectortools.Selector().by_leaves()[:-1].by_counts(
10         [-3, 1, -4, 2, -1, 3, -5, 2],
11         ).flatten(),
12     )

```

### C.2.4 ARMILLA.MATERIALS.INTERMITTENT\_GLISSANDI

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import selectortools
4 from abjad.tools import spannertools
5
6
7 intermittent_glissandi = consort.AttachmentExpression(
8     attachments=spannertools.Glissando(
9         allow_repeated_pitches=False,
10        allow_ties=False,
11        parenthesize_repeated_pitches=True,
12        ),
13     selector=selectortools.Selector()\
14     .by_leaves()\
15     [:-1]\

```

```

16     .append_callback(consort.AfterGraceSelectorCallback())\
17     .by_counts(
18         [-3, 4, -2, 3, -1, 4],
19         cyclic=True,
20         overhang=True,
21         fuse_overhang=True,
22         rotate=True,
23         ),
24 )

```

## C.2.5 ARMILLA.MATERIALS.INTERMITTENT\_TREMOLI

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import selectortools
4 from abjad.tools import spannertools
5
6
7 intermittent_tremoli = consort.AttachmentExpression(
8     attachments=spannertools.StemTremoloSpanner(),
9     selector=selectortools.Selector().by_leaves()[:-1].by_counts(
10        [-7, 2, -3, 1],
11        ),
12    )

```

## C.2.6 ARMILLA.MATERIALS.INTERMITTENT\_TRILLS

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import markuptools
4 from abjad.tools import schemetools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7
8
9 harmonic_fourth_trill = consort.ConsortTrillSpanner(
10     interval='+P4',
11     overrides={
12         'trill_pitch_head__stencil': schemetools.Scheme(
13             'ly:text-interface::print',
14             ),
15         'trill_pitch_head__text': markuptools.Markup.musicglyph(
16             'noteheads.s0harmonic',
17             direction=None,
18             ),
19         },
20     )
21
22 harmonic_third_trill = consort.ConsortTrillSpanner(
23     interval='+m3',
24     overrides={
25         'trill_pitch_head__stencil': schemetools.Scheme(
26             'ly:text-interface::print',
27             ),

```

```

28     'trill_pitch_head__text': markuptools.Markup.musicglyph(
29         'noteheads.s0harmonic',
30         direction=None,
31     ),
32 },
33 )
34
35 stopped_third_trill = consort.ConsortTrillSpanner(
36     interval='+m3',
37 )
38
39 intermittent_trills = consort.AttachmentExpression(
40     attachments=(
41         harmonic_fourth_trill,
42         harmonic_third_trill,
43         harmonic_fourth_trill,
44         stopped_third_trill,
45         stopped_third_trill,
46     ),
47     selector=selectortools.Selector()\
48         .by_leaves()\\
49         [:-1]\\
50         .append_callback(consort.AfterGraceSelectorCallback())\
51         .by_counts(
52             [-3, 2, -2, 1, -1, 2, -1, 1],
53             cyclic=True,
54             overhang=False,
55             fuse_overhang=False,
56             rotate=True,
57         ),
58     )

```

## C.2.7 ARMILLA.MATERIALS.LEFT\_HAND\_DIADS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import pitchtools
4 from abjad.tools import rhythmmakertools
5
6
7 left_hand_diads_music_specifier = consort.MusicSpecifier(
8     attachment_handler=consort.AttachmentHandler(),
9     pitch_handler=consort.AbsolutePitchHandler(
10        deviations=(0, 0, 0, 0, 0.5, 0),
11        forbid_repetitions=True,
12        logical_tie_expressions=(
13            consort.ChordExpression(chord_expr=(0, 8)),
14            consort.ChordExpression(chord_expr=(0, 5)),
15            consort.ChordExpression(chord_expr=(0, 8)),
16            consort.ChordExpression(chord_expr=(0, 8)),
17            consort.ChordExpression(chord_expr=(0, 5)),
18            consort.ChordExpression(chord_expr=(0, 8)),
19            consort.ChordExpression(chord_expr=(0, 5)),
20            consort.ChordExpression(chord_expr=(0, 5)),

```

```

21     consort.ChordExpression(chord_expr=(0, 10)),
22     consort.ChordExpression(chord_expr=(0, 8)),
23     consort.ChordExpression(chord_expr=(0, 8)),
24     consort.ChordExpression(chord_expr=(0, 10)),
25     ),
26     pitchSpecifier=consort.PitchSpecifier(
27         pitchSegments=(
28             "a c' c' a a c' a a g a c' c",
29             "a c' c' c' gs g c",
30             "a c' c' a a c' a g fs a a c' c",
31             ),
32             ratio=(1, 1, 1),
33             ),
34     pitchOperationSpecifier=consort.PitchOperationSpecifier(
35         pitchOperations=(
36             None,
37             pitchtools.PitchOperation(pitchtools.Rotation(1)),
38             None,
39             pitchtools.PitchOperation(pitchtools.Rotation(1)),
40             None,
41             ),
42             ratio=(2, 1, 2, 2, 1),
43             ),
44     ),
45     rhythmMaker=rhythmmakertools.NoteRhythmMaker(
46         tieSpecifier=rhythmmakertools.TieSpecifier(
47             tie_across_divisions=False,
48             ),
49         ),
50     )

```

## C.2.8 ARMILLA.MATERIALS.LEFT\_HAND\_DIETRO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import selectortools
5
6
7 left_hand_dietro_music_specifier = consort.MusicSpecifier(
8     attachmentHandler=consort.AttachmentHandler(
9         clefSpanner=consort.AttachmentExpression(
10            attachments=consort.ClefSpanner(
11                clef='percussion',
12                overrides={
13                    'note_head_style': 'cross',
14                    },
15                    ),
16                    selector=selectortools.Selector().by_leaves(),
17                    ),
18                    ),
19                    pitchHandler=consort.AbsolutePitchHandler(
20                        logicalTieExpressions=(
21                            consort.ChordExpression("g b"),

```

```

22         consort.ChordExpression("b d'"),
23         consort.ChordExpression("d' f'"),
24     ),
25 ),
26 rhythm_maker=rhythmmakertools.NoteRhythmMaker(
27     tieSpecifier=rhythmmakertools.TieSpecifier(
28         tie_across_divisions=False,
29     ),
30 ),
31 )

```

### C.2.9 ARMILLA.MATERIALS.LEFT\_HAND\_GLISSANDI\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7
8
9 left_hand_glissandi_music_specifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         glissando=consort.AttachmentExpression(
12             attachments=spannertools.Glissando(
13                 allow_repeated_pitches=False,
14                 allow_ties=False,
15                 parenthesize_repeated_pitches=True,
16             ),
17             selector=selectortools.Selector().by_leaves(
18                 ).append_callback(consort.AfterGraceSelectorCallback())
19         ),
20     ),
21     grace_handler=consort.GraceHandler(
22         counts=(0, 1, 2, 0, 0, 0),
23         only_if_preceded_by_nonsilence=True,
24     ),
25     pitch_handler=consort.AbsolutePitchHandler(
26         deviations=(0, 0, 0, 0, 0.5, 0),
27         forbid_repetitions=True,
28         grace_expressions=(
29             consort.HarmonicExpression('P4'),
30             consort.HarmonicExpression('M3'),
31             consort.HarmonicExpression('P5'),
32         ),
33         pitchSpecifier=consort.PitchSpecifier(
34             pitch_segments=(
35                 "a c' a a c'",
36                 "a g c' gs d'",
37                 "a c",
38             ),
39             ratio=(1, 1, 1),
40         ),
41     ),

```

```

42     rhythm_maker=consort.CompositeRhythmMaker(
43         last=rhythmmakertools.NoteRhythmMaker(),
44         default=rhythmmakertools.EvenDivisionRhythmMaker(
45             denominators=(4,),
46             extra_counts_per_division=(0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1),
47             duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
48                 decrease_durations_monotonically=True,
49                 forbidden_written_duration=durationaltools.Duration(1, 4),
50             ),
51         ),
52     ),
53 )

```

### C.2.10 ARMILLA.MATERIALS.LEFT\_HAND\_PIZZICATI\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6
7
8 left_hand_pizzicati_musicSpecifier = consort.MusicSpecifier(
9     attachmentHandler=consort.AttachmentHandler(
10         arpeggio=indicatortools.Arpeggio(),
11         clefSpanner=consort.AttachmentExpression(
12             attachments=consort.ClefSpanner('treble'),
13             selector=selectortools.Selector().by_leaves(),
14         ),
15     ),
16     pitchHandler=consort.AbsolutePitchHandler(
17         logicalTieExpressions=(
18             consortium.ChordExpression("cs' b' g'' ef'''"),
19             consortium.ChordExpression("fs e' c' gs'''"),
20             consortium.ChordExpression("f ef' b' g'''"),
21             consortium.ChordExpression("fs e' c' gs'''"),
22             consortium.ChordExpression("c' bf' fs' d'''"),
23             consortium.ChordExpression("cs' b' g' ef'''"),
24             consortium.ChordExpression("f ef' b' g'''"),
25             consortium.ChordExpression("fs e' c' gs'''"),
26             consortium.ChordExpression("d' c' gs' e'''"),
27         ),
28     ),
29     rhythmMaker=rhythmmakertools.IncisedRhythmMaker(
30         inciseSpecifier=rhythmmakertools.InciseSpecifier(
31             fillWithNotes=False,
32             prefixCounts=(1, 1, 1, 1, 2, 1),
33             prefixTalea=(1,),
34             suffixTalea=(1,),
35             suffixCounts=(0,),
36             taleaDenominator=16,
37         ),
38     ),
39 )

```

### C.2.11 ARMILLA.MATERIALS.LEFT\_HAND\_STASIS\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6
7
8 left_hand_stasis_musicSpecifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         trill_spanner=consort.AttachmentExpression(
11             attachments=(
12                 consort.ConsortTrillSpanner(interval='+m3'),
13                 consort.ConsortTrillSpanner(interval='+P4'),
14                 consort.ConsortTrillSpanner(interval='+m3'),
15             ),
16             selector=selectortools.Selector().by_leaves(),
17         ),
18     ),
19     minimum_phrase_duration=durationtools.Duration(1, 4),
20     pitch_handler=consort.AbsolutePitchHandler(
21         deviations=(0, 0, 0.5),
22         pitch_specifier="gs",
23     ),
24     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
25         duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
26             forbid_meter_rewriting=True,
27         ),
28     ),
29 )
```

### C.2.12 ARMILLA.MATERIALS.RIGHT\_HAND\_CIRCULAR\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad import durationtools
4 from abjad import indicatortools
5 from abjad import rhythmmakertools
6 from abjad import selectortools
7 from abjad import scoretools
8 from abjad import spannertools
9
10
11 right_hand_circular_musicSpecifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         bow_contact_points=consort.AttachmentExpression(
14             attachments=(
15                 indicatortools.BowContactPoint(0),
16                 indicatortools.BowContactPoint(1),
17                 indicatortools.BowContactPoint((4, 5)),
18                 indicatortools.BowContactPoint((4, 5)),
19                 indicatortools.BowContactPoint((3, 5)),
20                 indicatortools.BowContactPoint((4, 5)),
```

```

21     indicatortools.BowContactPoint((4, 5)),
22     indicatortools.BowContactPoint((3, 5)),
23     indicatortools.BowContactPoint((4, 5)),
24     indicatortools.BowContactPoint((3, 5)),
25     indicatortools.BowContactPoint((3, 5)),
26     indicatortools.BowContactPoint((4, 5)),
27     indicatortools.BowContactPoint((3, 5)),
28     indicatortools.BowContactPoint((3, 5)),
29     indicatortools.BowContactPoint((4, 5)),
30     ),
31     selector=selectortools.Selector().by_leaves().flatten(),
32     ),
33 bow_contact_spinner=spannertools.BowContactSpanner(),
34 bow_motion_techniques=consort.AttachmentExpression(
35     attachments=indicatortools.BowMotionTechnique('circular'),
36     selector=selectortools.Selector().by_leaves().flatten(),
37     ),
38 dynamic_expressions=consort.DynamicExpression(
39     dynamic_tokens='p ppp p ppp p ppp',
40     ),
41 string_contact_points=consort.AttachmentExpression(
42     attachments=(
43         None,
44         indicatortools.StringContactPoint('ordinario'),
45         indicatortools.StringContactPoint('sul ponticello'),
46         indicatortools.StringContactPoint('ordinario'),
47         indicatortools.StringContactPoint('ordinario'),
48         indicatortools.StringContactPoint('molto sul ponticello'),
49         ),
50         scope=scoretools.Voice,
51         selector=selectortools.Selector().append_callback(
52             consort.PhrasedSelectorCallback()).flatten(),
53         ),
54 string_contact_spinner=consort.StringContactSpanner(),
55     ),
56 rhythm_maker=consort.CompositeRhythmMaker(
57     last=rhythmmakertools.IncisedRhythmMaker(
58         inciseSpecifier=rhythmmakertools.InciseSpecifier(
59             prefix_counts=[0],
60             suffix_talea=[1],
61             suffix_counts=[1],
62             talea_denominator=16,
63             )),
64     ),
65 default=rhythmmakertools.EvenDivisionRhythmMaker(
66     denominators=(4,),
67     extra_counts_per_division=(0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1),
68     duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
69         decrease_durations_monotonically=True,
70         forbidden_written_duration=durationtools.Duration(1, 4),
71         forbid_meter_rewriting=True,
72         )),
73     ),
74 ),

```

75 )

### C.2.13 ARMILLA.MATERIALS.RIGHT\_HAND\_JETE\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad import durationtools
4 from abjad import indicatortools
5 from abjad import rhythmmakertools
6 from abjad import scoretools
7 from abjad import selectortools
8 from abjad import spannertools
9
10
11 right_hand_jete_musicSpecifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         bow_contact_points=consort.AttachmentExpression(
14             attachments=(
15                 indicatortools.BowContactPoint((4, 5)),
16                 indicatortools.BowContactPoint((3, 5)),
17             ),
18             selector=selectortools.Selector().by_leaves().flatten(),
19         ),
20         bow_contact_spinner=spannertools.BowContactSpinner(),
21         bow_motion_techniques=consort.AttachmentExpression(
22             attachments=indicatortools.BowMotionTechnique('jete'),
23             selector=selectortools.Selector()
24                 .by_leaves()
25                 .get_slice(stop=-1)
26                 .by_counts(
27                     [2, 1, 2],
28                     overhang=True,
29                     fuse_overhang=True,
30                     rotate=True,
31                 )
32                 .get_item(0, apply_to_each=False)
33                 .flatten(),
34         ),
35         dynamic_expressions=consort.AttachmentExpression(
36             attachments=(
37                 indicatortools.Dynamic('mf'),
38                 indicatortools.Dynamic('mp'),
39             ),
40             selector=selectortools.Selector().by_leaves()[0]
41         ),
42         string_contact_points=consort.AttachmentExpression(
43             attachments=(
44                 indicatortools.StringContactPoint('ordinario'),
45                 indicatortools.StringContactPoint('sul ponticello'),
46                 indicatortools.StringContactPoint('molto sul ponticello'),
47                 indicatortools.StringContactPoint('sul tasto'),
48             ),
49             scope=scoretools.Voice,
50             selector=selectortools.Selector()
```

```

51     .append_callback(consort.PhrasedSelectorCallback())
52     .by_counts(
53         [1, -1,],
54         cyclic=True,
55         nonempty=True,
56         overhang=True,
57         )
58     .flatten()
59     ),
60     string_contact_spanner=consort.StringContactSpanner(),
61     ),
62 rhythm_maker=consort.CompositeRhythmMaker(
63     last=rhythmmakertools.IncisedRhythmMaker(
64         inciseSpecifier=rhythmmakertools.InciseSpecifier(
65             prefix_counts=[0],
66             suffix_talea=[1],
67             suffix_counts=[1],
68             talea_denominator=16,
69             ),
70             ),
71     default=rhythmmakertools.EvenDivisionRhythmMaker(
72         denominators=(8, 8, 4),
73         extra_counts_per_division=(0, 0, 1, 0, 1, 2),
74         duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
75             decrease_durations_monotonically=True,
76             forbidden_written_duration=durationtools.Duration(1, 4),
77             forbid_meter_rewriting=True,
78             ),
79             ),
80         ),
81     )

```

#### C.2.14 ARMILLA.MATERIALS.RIGHT\_HAND\_OVERPRESSURE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import scoretools
7 from abjad.tools import selectortools
8 from abjad.tools import spannertools
9
10
11 right_hand_overpressure_musicSpecifier = consortium.MusicSpecifier(
12     attachmentHandler=consort.AttachmentHandler(
13         bow_contact_points=consort.AttachmentExpression(
14             attachments=(
15                 indicatortools.BowContactPoint(0),
16                 indicatortools.BowContactPoint((1, 8)),
17                 indicatortools.BowContactPoint((1, 4)),
18                 indicatortools.BowContactPoint((3, 8)),
19                 indicatortools.BowContactPoint((1, 2)),
20                 indicatortools.BowContactPoint((5, 8)),

```

```

21     indicatortools.BowContactPoint((3, 4)),
22     indicatortools.BowContactPoint((7, 8)),
23     indicatortools.BowContactPoint(1),
24     indicatortools.BowContactPoint(0),
25     indicatortools.BowContactPoint((1, 8)),
26     indicatortools.BowContactPoint(0),
27     indicatortools.BowContactPoint((1, 8)),
28     indicatortools.BowContactPoint((1, 4)),
29     indicatortools.BowContactPoint((1, 8)),
30     indicatortools.BowContactPoint((3, 8)),
31     indicatortools.BowContactPoint((1, 4)),
32     indicatortools.BowContactPoint((3, 8)),
33     indicatortools.BowContactPoint((1, 2)),
34     indicatortools.BowContactPoint((5, 8)),
35     indicatortools.BowContactPoint((3, 4)),
36     indicatortools.BowContactPoint((7, 8)),
37     indicatortools.BowContactPoint(1),
38     ),
39     selector=selectortools.Selector().by_leaves().flatten(),
40     ),
41 bow_contact_spinner=spannertools.BowContactSpanner(),
42 dynamic_expressions=consort.DynamicExpression(
43     dynamic_tokens='p ppp p ppp p ppp f',
44     ),
45 string_contact_points=consort.AttachmentExpression(
46     attachments=(
47         indicatortools.StringContactPoint('ordinario'),
48         indicatortools.StringContactPoint('sul tasto'),
49         indicatortools.StringContactPoint('molto sul tasto'),
50         ),
51     scope=scoretools.Voice,
52     selector=selectortools.Selector(
53         ).append_callback(
54             consort.PhrasedSelectorCallback()
55         ).by_counts(
56             [1, -2, 1, -2, 1, -1],
57             cyclic=True,
58             nonempty=True,
59             overhang=True,
60             ).flatten()
61         ),
62     string_contact_spinner=consort.StringContactSpanner(),
63     ),
64 rhythm_maker=consort.CompositeRhythmMaker(
65     last=rhythmmakertools.IncisedRhythmMaker(
66         inciseSpecifier=rhythmmakertools.InciseSpecifier(
67             prefix_counts=[0],
68             suffix_talea=[1],
69             suffix_counts=[1],
70             talea_denominator=16,
71             ),
72         ),
73     default=rhythmmakertools.EvenDivisionRhythmMaker(
74         denominators=(4,),
```

```

75     extra_counts_per_division=(0, 0, 0, 1, 0, 0, 1, 0, 1),
76     duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
77         decrease_durations_monotonically=True,
78         forbidden_written_duration=durationtools.Duration(1, 4),
79         forbid_meter_rewriting=True,
80     ),
81     ),
82   ),
83 )

```

### C.2.15 ARMILLA.MATERIALS.RIGHT\_HAND\_PIZZICATI\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import scoretools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8
9
10 right_hand_pizzicati_musicSpecifier = consort.MusicSpecifier(
11     attachmentHandler=consort.AttachmentHandler(
12         bow_contact_points=consort.AttachmentExpression(
13             attachments=(
14                 indicatortools.BowContactPoint(None),
15             ),
16             selector=selectortools.Selector().by_leaves().flatten(),
17         ),
18         bow_contact_spinner=spannertools.BowContactSpinner(),
19         dynamic_expressions=consort.DynamicExpression(
20             dynamic_tokens='mf',
21         ),
22         string_contact_points=consort.AttachmentExpression(
23             attachments=indicatortools.StringContactPoint('pizzicato'),
24             scope=scoretools.Voice,
25             selector=selectortools.Selector(
26                 ).by_leaves(
27             )[0]
28         ),
29         string_contact_spinner=consort.StringContactSpinner(),
30     ),
31     rhythm_maker=rhythmmakertools.IncisedRhythmMaker(
32         inciseSpecifier=rhythmmakertools.InciseSpecifier(
33             fill_with_notes=False,
34             prefix_counts=(1, 1, 1, 1, 2, 1),
35             prefix_talea=(1,),
36             suffix_talea=(1,),
37             suffix_counts=(0,),
38             talea_denominator=16,
39         ),
40     ),
41 )

```

### C.2.16 ARMILLA.MATERIALS.RIGHT\_HAND\_STASIS\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import scoretools
7 from abjad.tools import selectortools
8 from abjad.tools import spannertools
9
10
11 right_hand_stasis_music_specifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         bow_contact_points=consort.AttachmentExpression(
14             attachments=(
15                 indicatortools.BowContactPoint(1),
16                 indicatortools.BowContactPoint(1),
17                 indicatortools.BowContactPoint((4, 5)),
18                 indicatortools.BowContactPoint(1),
19                 indicatortools.BowContactPoint((4, 5)),
20                 indicatortools.BowContactPoint(1),
21                 indicatortools.BowContactPoint(1),
22                 indicatortools.BowContactPoint((2, 5)),
23             ),
24             selector=selectortools.Selector().by_leaves().flatten(),
25         ),
26         bow_contact_spinner=spannertools.BowContactSpinner(),
27         dynamic_expressions=consort.DynamicExpression(
28             dynamic_tokens='p ppp',
29         ),
30         stem_tremolo_spinner=spannertools.StemTremoloSpinner(),
31         string_contact_points=consort.AttachmentExpression(
32             attachments=(
33                 indicatortools.StringContactPoint('ordinario'),
34                 indicatortools.StringContactPoint('sul tasto'),
35                 indicatortools.StringContactPoint('molto sul tasto'),
36             ),
37             scope=scoretools.Voice,
38             selector=selectortools.Selector(
39                 ).append_callback(
40                     consort.PhrasedSelectorCallback()
41                 ).by_counts(
42                     [1, -2, 1, -2, 1, -1],
43                     cyclic=True,
44                     nonempty=True,
45                     overhang=True,
46                     ).flatten()
47             ),
48             string_contact_spinner=consort.StringContactSpinner(),
49         ),
50         minimum_phrase_duration=durationtools.Duration(1, 4),
51         rhythm_maker=consort.CompositeRhythmMaker(
52             last=rhythmmakertools.IncisedRhythmMaker(
```

```

53     inciseSpecifier=rhythmmakertools.InciseSpecifier(
54         prefixCounts=[0],
55         suffixTalea=[1],
56         suffixCounts=[1],
57         taleaDenominator=16,
58         ),
59     ),
60     default=rhythmmakertools.EvenDivisionRhythmMaker(
61         denominators=(4,),
62         extraCountsPerDivision=(0, 0, 0, 1, 0, 0, 1, 0, 1),
63         durationSpellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
64             decreaseDurationsMonotonically=True,
65             forbiddenWrittenDuration=durationtools.Duration(1, 4),
66             forbidMeterRewriting=True,
67             ),
68         ),
69     ),
70 )

```

### C.2.17 ARMILLA.MATERIALS.SPARSE\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5
6
7 sparseTimespanMaker = consort.TaleaTimespanMaker(
8     initialSilenceTalea=rhythmmakertools.Talea(
9         counts=(1, 0),
10        denominator=8,
11        ),
12        padding=durationtools.Duration(1, 8),
13        playingTalea=rhythmmakertools.Talea(
14            counts=(1, 1, 2),
15            denominator=8,
16            ),
17            playingGroupings=(1, 2, 1, 2, 2, 3),
18            silenceTalea=rhythmmakertools.Talea(
19                counts=(1, 1, 1, 2, 1, 1, 2),
20                denominator=8,
21                ),
22                timespanSpecifier=consort.TimespanSpecifier(
23                    minimumDuration=0,
24                    ),
25        )

```

### C.2.18 ARMILLA.MATERIALS.SUSTAINED\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4
5

```

```

6 sustained_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=rhythmmakertools.Talea(
8         counts=(5, 0),
9         denominator=8,
10    ),
11    playing_talea=rhythmmakertools.Talea(
12        counts=(5, 7, 4, 5),
13        denominator=8,
14    ),
15    playing_groupings=(5, 6, 4, 8, 9),
16    silence_talea=rhythmmakertools.Talea(
17        counts=(3, 5, 3, 2, 5, 3),
18        denominator=8,
19    ),
20 )

```

### C.2.19 ARMILLA.MATERIALS.SYNCHRONIZED\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5
6
7 synchronized_timespan_maker = consort.TaleaTimespanMaker(
8     padding=durationtools.Duration(1, 8),
9     playing_talea=rhythmmakertools.Talea(
10        counts=(1, 1, 2, 1, 2, 2, 3),
11        denominator=8,
12    ),
13    playing_groupings=(1, 2),
14    silence_talea=rhythmmakertools.Talea(
15        counts=(3, 4, 5, 2, 7, 9),
16        denominator=8,
17    ),
18    synchronize_step=True,
19    timespanSpecifier=consort.TimespanSpecifier(
20        minimum_duration=0,
21    ),
22 )

```

### C.2.20 ARMILLA.MATERIALS.TIME\_SIGNATURES

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 time_signatures = indicatortools.TimeSignatureInventory([
6     (2, 4),
7     (3, 4),
8     (3, 8),
9     (4, 8),
10    (5, 8),

```

```
11     (6, 8),  
12 )]
```

### C.3 ARMILLA

#### SEGMENTS

#### SOURCE

##### C.3.1 ARMILLA.SEGMENTS SEGMENT\_A\_FAR\_SORR

```
1 # -*- encoding: utf-8 -*-  
2 import armilla  
3 import consort  
4 from abjad import new  
5 from abjad.tools import durationtools  
6 from abjad.tools import indicatortools  
7 from abjad.tools import scoretools  
8 from abjad.tools import selector tools  
9  
10  
11 ### SEGMENT MAKER ###  
12  
13 segment_maker = armilla.ArmillaSegmentMaker(  
14     desired_duration_in_seconds=90,  
15     discard_final_silence=True,  
16     name='Far Sorr',  
17     repeat=False,  
18     tempo=indicatortools.Tempo((1, 4), 36),  
19 )  
20  
21 ### ATTACHMENTS ###  
22  
23 dietro_ponticello = consort.AttachmentExpression(  
24     attachments=indicatortools.StringContactPoint('diestro ponticello'),  
25     scope=scoretools.Voice,  
26     selector=selector tools.Selector().by_leaves(),  
27 )  
28 intermittent_accents = armilla.materials.intermittent_accents  
29 intermittent_circular = armilla.materials.intermittent_circular  
30 intermittent_glissandi = armilla.materials.intermittent_glissandi  
31 intermittent_tremoli = armilla.materials.intermittent_tremoli  
32  
33 ### MUSIC SPECIFIERS ###  
34  
35 lh_diads = new(  
36     armilla.materials.left_hand_diads_music_specifier,  
37     minimum_phrase_duration=durationtools.Duration(1, 4),  
38 )  
39 lh_dietro = new(  
40     armilla.materials.left_hand_dietro_music_specifier,  
41     minimum_phrase_duration=durationtools.Duration(1, 4),  
42 )  
43 lh_pizzicati = armilla.materials.left_hand_pizzicati_music_specifier  
44 rh_overpressure = new(  
45     armilla.materials.right_hand_overpressure_music_specifier,  
46     minimum_phrase_duration=durationtools.Duration(1, 4),  
47 )
```

```

48 rh_pizzicati = armilla.materials.right_hand_pizzicati_musicSpecifier
49
50 ### OVERPRESSURE ###
51
52 segment_maker.add_setting(
53     timespan_maker=armilla.materials.sustained_timespan_maker,
54     timespan_identifier=consort.RatioPartsExpression(
55         parts=(0,),
56         ratio=(1, 1, 1),
57     ),
58     viola_1=consort.CompositeMusicSpecifier(
59         primary_music_specifier=new(
60             rh_overpressure,
61             attachment_handler__string_contact_points=dietro_ponticello,
62         ),
63         secondary_music_specifier=lh_dietro,
64     ),
65     viola_2=consort.CompositeMusicSpecifier(
66         primary_music_specifier=rh_overpressure,
67         secondary_music_specifier=lh_diads,
68     ),
69 )
70
71 segment_maker.add_setting(
72     timespan_maker=armilla.materials.dense_timespan_maker,
73     timespan_identifier=consort.RatioPartsExpression(
74         parts=(1,),
75         ratio=(1, 1, 1),
76     ),
77     viola_1=consort.CompositeMusicSpecifier(
78         primary_music_specifier=new(
79             rh_overpressure,
80             attachment_handler__articulations=intermittent_accents,
81             attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
82             rhythm_maker__default_denominators=(4, 4, 4, 8),
83         ),
84         rotation_indices=(1, 0, 1, 0, -1),
85         secondary_music_specifier=new(
86             lh_diads,
87             attachment_handler__glissando=intermittent_glissandi,
88         ),
89     ),
90
91     viola_2=consort.CompositeMusicSpecifier(
92         primary_music_specifier=new(
93             rh_overpressure,
94             attachment_handler__articulations=intermittent_accents,
95             attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
96             rhythm_maker__default_denominators=(4, 4, 4, 8, 4, 8),
97         ),
98         rotation_indices=(1, 0, 1, 0, -1),
99         secondary_music_specifier=lh_diads,
100     ),
101 )

```

```

102
103 segment_maker.add_setting(
104     timespan_maker=armilla.materials.sustained_timespan_maker,
105     timespan_identifier=consort.RatioPartsExpression(
106         parts=(2,),
107         ratio=(1, 1, 1),
108     ),
109     viola_1=consort.CompositeMusicSpecifier(
110         primary_music_specifier=new(
111             rh_overpressure,
112             attachment_handler__articulations=intermittent_accents,
113             rhythm_maker__default_denominators=(4, 4, 4, 16, 4, 16),
114         ),
115         rotation_indices=(1, 0, 1, 0, -1),
116         secondary_music_specifier=new(
117             lh_diads,
118             attachment_handler__glissando=intermittent_glissandi,
119         ),
120     ),
121     viola_2=consort.CompositeMusicSpecifier(
122         primary_music_specifier=new(
123             rh_overpressure,
124             attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
125             attachment_handler__bow_motion_technique_x=intermittent_circular,
126         ),
127         rotation_indices=(1, 0, 1, 0, -1),
128         secondary_music_specifier=new(
129             lh_diads,
130             attachment_handler__glissando=intermittent_glissandi,
131         ),
132     ),
133 )
134
135 ### PIZZICATI ###
136
137 segment_maker.add_setting(
138     timespan_maker=armilla.materials.sparse_timespan_maker,
139     timespan_identifier=consort.RatioPartsExpression(
140         parts=(1,),
141         ratio=(5, 1),
142     ),
143     viola_2=consort.CompositeMusicSpecifier(
144         primary_music_specifier=rh_pizzicati,
145         secondary_music_specifier=lh_pizzicati,
146     ),
147 )

```

### C.3.2 ARMILLA.SEGMENTS SEGMENT\_B\_SELIDOR\_A

```

1 # -*- encoding: utf-8 -*-
2 import armilla
3 import consort
4 from abjad import new
5 from abjad.tools import indicatortools

```

```

6
7
8 ### SEGMENT MAKER ###
9
10 segment_maker = armilla.ArmillaSegmentMaker(
11     desired_duration_in_seconds=20 / 2,
12     discard_final_silence=True,
13     name='Selidor (i)',
14     repeat=True,
15     tempo=indicatortools.Tempo((1, 4), 72),
16 )
17
18 ### ATTACHMENTS ###
19
20 intermittent_trills = armilla.materials.intermittent_trills
21
22 ### MUSIC SPECIFIERS ###
23
24 rh_circular = armilla.materials.right_hand_circular_music_specifier
25 lh_glissandi = armilla.materials.left_hand_glissandi_music_specifier
26
27 ### SETTINGS ###
28
29 segment_maker.add_setting(
30     timespan_maker=armilla.materials.dense_timespan_maker,
31     viola_1=consort.CompositeMusicSpecifier(
32         primary_music_specifier=rh_circular,
33         rotation_indices=(1, 0, 1, 0, -1),
34         secondary_music_specifier=new(
35             lh_glissandi,
36             attachment_handler__trill_spanner=intermittent_trills,
37             ),
38         ),
39     viola_2=consort.CompositeMusicSpecifier(
40         primary_music_specifier=rh_circular,
41         rotation_indices=(1, 0, 1, 0, -1),
42         secondary_music_specifier=new(
43             lh_glissandi,
44             attachment_handler__trill_spanner=intermittent_trills,
45             ),
46         ),
47     )

```

### C.3.3 ARMILLA.SEGMENTS.SEGMENT\_C\_WELLOGY

```

1 # -*- encoding: utf-8 -*-
2 from abjad import new
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 import armilla
7 import consort
8
9

```

```

10 ### SEGMENT MAKER ###
11
12 segment_maker = armilla.ArmillSegmentMaker(
13     desired_duration_in_seconds=30 / 2,
14     discard_final_silence=True,
15     name='Wellogy',
16     repeat=True,
17     tempo=indicatortools.Tempo((1, 4), 108),
18 )
19
20 ### MUSIC SPECIFIERS ###
21
22 lh_glissandi = new(
23     armilla.materials.left_hand_glissandi_music_specifier,
24     attachment_handler__bend_after=consort.AttachmentExpression(
25         attachments=indicatortools.BendAfter(4),
26         selector=selectortools.Selector().by_leaves()[-1],
27     ),
28     pitch_handler__pitchSpecifier="fs' gs' as",
29     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
30         duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
31             forbid_meter_rewriting=True,
32         ),
33     ),
34 )
35 lh_stasis = armilla.materials.left_hand_stasis_music_specifier
36 rh_jete = armilla.materials.right_hand_jete_music_specifier
37 rh_stasis = armilla.materials.right_hand_stasis_music_specifier
38
39 ### SETTINGS ###
40
41 segment_maker.add_setting(
42     timespan_maker=armilla.materials.sustained_timespan_maker,
43     timespan_identifier=consort.RatioPartsExpression(
44         ratio=(1, 2),
45         parts=(1,),
46     ),
47     viola_1=consort.CompositeMusicSpecifier(
48         discard_inner_offsets=True,
49         primary_music_specifier=rh_stasis,
50         secondary_music_specifier=lh_stasis,
51     ),
52     viola_2=consort.CompositeMusicSpecifier(
53         discard_inner_offsets=True,
54         primary_music_specifier=rh_stasis,
55         secondary_music_specifier=lh_stasis,
56     ),
57 )
58
59 segment_maker.add_setting(
60     timespan_maker=new(
61         armilla.materials.sparse_timespan_maker,
62         playing_talea__counts=(2, 1, 2, 1, 1),
63     ),

```

```

64     timespan_identifier=consort.RatioPartsExpression(
65         ratio=(1, 2),
66         parts=(0,),
67     ),
68     viola_1=consort.CompositeMusicSpecifier(
69         primary_music_specifier=rh_jete,
70         secondary_music_specifier=lh_glissandi,
71     ),
72     viola_2=consort.CompositeMusicSpecifier(
73         primary_music_specifier=rh_jete,
74         secondary_music_specifier=lh_glissandi,
75     ),
76 )
77
78 segment_maker.add_setting(
79     timespan_maker=new(
80         armilla.materials.sparse_timespan_maker,
81         playing_talea_counts=(2, 1, 2, 1, 1),
82         playing_groupings=(1, 1, 2),
83         silence_talea_denominator=2,
84     ),
85     timespan_identifier=consort.RatioPartsExpression(
86         ratio=(2, 1),
87         parts=(1,),
88     ),
89     viola_1=consort.CompositeMusicSpecifier(
90         primary_music_specifier=rh_jete,
91         secondary_music_specifier=lh_glissandi,
92     ),
93     viola_2=consort.CompositeMusicSpecifier(
94         primary_music_specifier=rh_jete,
95         secondary_music_specifier=lh_glissandi,
96     ),
97 )

```

### C.3.4 ARMILLA.SEGMENTS SEGMENT\_D THE\_LONG\_DUNE\_A

```

1 # -*- encoding: utf-8 -*-
2 import armilla
3 import consort
4 from abjad import new
5 from abjad.tools import indicatortools
6 from abjad.tools import scoretools
7 from abjad.tools import selectortools
8
9
10 ### SEGMENT MAKER ###
11
12 segment_maker = armilla.ArmillaSegmentMaker(
13     desired_duration_in_seconds=120,
14     discard_final_silence=True,
15     name='The Long Dune (i)',
16     repeat=False,
17     tempo=indicatortools.Tempo((1, 4), 36),

```

```

18     )
19
20 ### ATTACHMENTS ###
21
22 dietro_ponticello = consort.AttachmentExpression(
23     attachments=indicatortools.StringContactPoint('dietro ponticello'),
24     scope=scoretools.Voice,
25     selector=selectortools.Selector().by_leaves(),
26 )
27 dynamics_a = dynamic_expressions = consort.DynamicExpression(
28     dynamic_tokens='p mf p ppp f p ff',
29 )
30 dynamics_b = dynamic_expressions = consort.DynamicExpression(
31     dynamic_tokens='f p f mf ff p fff f fff mf fff',
32 )
33 intermittent_accents = armilla.materials.intermittent_accents
34 intermittent_circular = armilla.materials.intermittent_circular
35 intermittent_tremoli = armilla.materials.intermittent_tremoli
36
37 ### MUSIC SPECIFIERS ###
38
39 lh_diads = armilla.materials.left_hand_diads_musicSpecifier
40 lh_dietro = armilla.materials.left_hand_dietro_musicSpecifier
41 rh_overpressure = armilla.materials.right_hand_overpressure_musicSpecifier
42
43 ### OVERPRESSURE ###
44
45 segment_maker.add_setting(
46     timespan_maker=armilla.materials.sustained_timespan_maker,
47     timespan_identifier=consort.RatioPartsExpression(
48         parts=(0,),
49         ratio=(3, 2, 1),
50     ),
51     viola_1=consort.CompositeMusicSpecifier(
52         discard_inner_offsets=True,
53         primary_musicSpecifier=new(
54             rh_overpressure,
55             attachment_handler__string_contact_points=dietro_ponticello,
56         ),
57         secondary_musicSpecifier=lh_dietro,
58     ),
59     viola_2=consort.CompositeMusicSpecifier(
60         discard_inner_offsets=True,
61         primary_musicSpecifier=new(
62             rh_overpressure,
63             attachment_handler__string_contact_points=dietro_ponticello,
64         ),
65         secondary_musicSpecifier=lh_dietro,
66     ),
67 )
68
69 segment_maker.add_setting(
70     timespan_maker=armilla.materials.dense_timespan_maker,
71     timespan_identifier=consort.RatioPartsExpression(

```

```

72     parts=(1,),
73     ratio=(3, 2, 1),
74     ),
75     viola_1=consort.CompositeMusicSpecifier(
76         discard_inner_offsets=True,
77         primary_music_specifier=new(
78             rh_overpressure,
79             attachment_handler__articulations=intermittent_accents,
80             attachment_handler__dynamic_expressions=dynamics_a,
81             attachment_handler__string_contact_points=dietro_ponticello,
82             rhythm_maker__default__denominators=(4, 4, 8, 8, 4, 8),
83             ),
84             secondary_music_specifier=lh_dietro,
85             ),
86     viola_2=consort.CompositeMusicSpecifier(
87         primary_music_specifier=new(
88             rh_overpressure,
89             attachment_handler__articulations=intermittent_accents,
90             attachment_handler__bow_motion_technique_x=intermittent_circular,
91             attachment_handler__dynamic_expressions=dynamics_a,
92             attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
93             rhythm_maker__default__denominators=(4, 4, 4, 8, 4, 8, 16),
94             ),
95             secondary_music_specifier=lh_diads,
96             ),
97     )
98
99 segment_maker.add_setting(
100     timespan_maker=armilla.materials.sustained_timespan_maker,
101     timespan_identifier=consort.RatioPartsExpression(
102         parts=(2,),
103         ratio=(3, 2, 1),
104         ),
105     viola_1=consort.CompositeMusicSpecifier(
106         primary_music_specifier=new(
107             rh_overpressure,
108             attachment_handler__articulations=intermittent_accents,
109             attachment_handler__bow_motion_technique_x=intermittent_circular,
110             attachment_handler__dynamic_expressions=dynamics_b,
111             attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
112             rhythm_maker__default__denominators=(4, 4, 4, 16, 4, 16),
113             ),
114             rotation_indices=(1, 0, 1, 0, -1),
115             secondary_music_specifier=lh_diads,
116             ),
117     viola_2=consort.CompositeMusicSpecifier(
118         primary_music_specifier=new(
119             rh_overpressure,
120             attachment_handler__articulations=intermittent_accents,
121             attachment_handler__bow_motion_technique_x=intermittent_circular,
122             attachment_handler__dynamic_expressions=dynamics_b,
123             attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
124             rhythm_maker__default__denominators=(4, 4, 16, 4, 16, 16),
125             ),

```

```

126     rotation_indices=(1, 0, 1, 0, -1),
127     secondary_music_specifier=lh_diads,
128   ),
129 )

```

### C.3.5 ARMILLA SEGMENTS SEGMENT\_E\_SELIDOR\_B

```

1 # -*- encoding: utf-8 -*-
2 import armilla
3 import consort
4 from abjad import new
5 from abjad.tools import indicatortools
6
7
8 ### SEGMENT MAKER ###
9
10 segment_maker = armilla.ArmillaSegmentMaker(
11     desired_duration_in_seconds=40 / 2,
12     discard_final_silence=True,
13     name='Selidor (ii)',
14     repeat=True,
15     tempo=indicatortools.Tempo((1, 4), 72),
16 )
17
18 ### ATTACHMENTS ###
19
20 intermittent_trills = armilla.materials.intermittent_trills
21
22 ### MUSIC SPECIFIERS ###
23
24 lh_glissandi = armilla.materials.left_hand_glissandi_musicSpecifier
25 rh_circular = armilla.materials.right_hand_circular_musicSpecifier
26
27 ### SETTINGS ###
28
29 segment_maker.add_setting(
30     timespan_maker=armilla.materials.dense_timespan_maker,
31     viola_1=consort.CompositeMusicSpecifier(
32         primary_musicSpecifier=rh_circular,
33         rotation_indices=(1, 0, 1, 0, -1),
34         secondary_musicSpecifier=new(
35             lh_glissandi,
36             attachment_handler__trill_spanner=intermittent_trills,
37         ),
38     ),
39     viola_2=consort.CompositeMusicSpecifier(
40         primary_musicSpecifier=rh_circular,
41         rotation_indices=(1, 0, 1, 0, -1),
42         secondary_musicSpecifier=new(
43             lh_glissandi,
44             attachment_handler__trill_spanner=intermittent_trills,
45         ),
46     )
47 )

```

### C.3.6 ARMILLA.SEGMENTS SEGMENT\_F THE ISLE\_OF\_THE\_EAR

```
1 # -*- encoding: utf-8 -*-
2 import armilla
3 import consort
4 from abjad import new
5 from abjad.tools import durationtools
6 from abjad.tools import indicatortools
7 from abjad.tools import rhythmmakertools
8 from abjad.tools import selectortools
9
10
11 ### SEGMENT MAKER ###
12
13 segment_maker = armilla.ArmillaSegmentMaker(
14     desired_duration_in_seconds=40 / 2,
15     discard_final_silence=True,
16     name='The Isle of the Ear',
17     repeat=True,
18     tempo=indicatortools.Tempo((1, 4), 108),
19 )
20
21 ### ATTACHMENTS ###
22
23 ### MUSIC SPECIFIERS ###
24
25 lh_glissandi = new(
26     armilla.materials.left_hand_glissandi_musicSpecifier,
27     attachment_handler__bend_after=consort.AttachmentExpression(
28         attachments=(
29             indicatortools.BendAfter(4),
30             indicatortools.BendAfter(4),
31             indicatortools.BendAfter(-4),
32             ),
33             selector=selectortools.Selector().by_leaves()[-1],
34             ),
35     pitch_handler__pitchSpecifier="fs' gs' as''",
36     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
37         duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
38             forbid_meter_rewriting=True,
39             ),
40             ),
41         )
42 lh_stasis = armilla.materials.left_hand_stasis_musicSpecifier
43 rh_jete = armilla.materials.right_hand_jete_musicSpecifier
44 rh_stasis = armilla.materials.right_hand_stasis_musicSpecifier
45
46 ### SETTINGS ###
47
48 segment_maker.add_setting(
49     timespan_maker=armilla.materials.sustained_timespan_maker,
50     timespan_identifier=consort.RatioPartsExpression(
51         ratio=(1, 2),
52         parts=(1,),
```

```

53     ),
54     viola_1=consort.CompositeMusicSpecifier(
55         discard_inner_offsets=True,
56         primary_music_specifier=rh_stasis,
57         rotation_indices=(1, 0, 1, 0, -1),
58         secondary_music_specifier=lh_stasis,
59         ),
60     viola_2=consort.CompositeMusicSpecifier(
61         discard_inner_offsets=True,
62         primary_music_specifier=rh_stasis,
63         rotation_indices=(1, 0, 1, 0, -1),
64         secondary_music_specifier=lh_stasis,
65         ),
66     )
67
68 segment_maker.add_setting(
69     timespan_maker=armilla.materials.synchronized_timespan_maker,
70     timespan_identifier=consort.RatioPartsExpression(
71         ratio=(1, 3),
72         parts=(0,),
73         ),
74     viola_1=consort.CompositeMusicSpecifier(
75         primary_music_specifier=rh_jete,
76         secondary_music_specifier=lh_glissandi,
77         ),
78     viola_2=consort.CompositeMusicSpecifier(
79         primary_music_specifier=rh_jete,
80         secondary_music_specifier=lh_glissandi,
81         ),
82     )
83
84 segment_maker.add_setting(
85     timespan_maker=new(
86         armilla.materials.sparse_timespan_maker,
87         padding=durationtools.Duration(1, 8),
88         playing_talea_counts=(2, 1, 2, 1, 1),
89         playing_groupings=(1, 1, 2),
90         silence_talea_denominator=4,
91         ),
92     timespan_identifier=consort.RatioPartsExpression(
93         ratio=(2, 1, 1),
94         parts=(2,),
95         ),
96     viola_1=consort.CompositeMusicSpecifier(
97         primary_music_specifier=rh_jete,
98         secondary_music_specifier=lh_glissandi,
99         ),
100    viola_2=consort.CompositeMusicSpecifier(
101        primary_music_specifier=rh_jete,
102        secondary_music_specifier=lh_glissandi,
103        ),
104    )

```

### C.3.7 ARMILLA.SEGMENTS SEGMENT\_G\_SELIDOR\_C

```
1 # -*- encoding: utf-8 -*-
2 import armilla
3 import consort
4 from abjad import new
5 from abjad.tools import indicatortools
6 from abjad.tools import spannertools
7 from abjad.tools import selectortools
8
9
10 ### SEGMENT MAKER ###
11
12 segment_maker = armilla.ArmillaSegmentMaker(
13     desired_duration_in_seconds=60 / 2,
14     discard_final_silence=True,
15     name='Selidor (iii)',
16     repeat=True,
17     tempo=indicatortools.Tempo((1, 4), 72),
18 )
19
20 ### ATTACHMENTS ###
21
22 intermittent_trills = armilla.materials.intermittent_trills
23
24 ### MUSIC SPECIFIERS ###
25
26 rh_circular = new(
27     armilla.materials.right_hand_circular_music_specifier,
28     attachment_handler__stem_tremolo_spanner=consort.AttachmentExpression(
29         attachments=(
30             None,
31             spannertools.StemTremoloSpanner(),
32         ),
33         selector=selectortools.Selector().by_leaves().by_counts(
34             [5, 1, 4, 2, 6, 3], cyclic=True,
35         )
36     ),
37 )
38
39 lh_glissandi = new(
40     armilla.materials.left_hand_glissandi_music_specifier,
41     pitch_handler__pitch_specifier=consort.PitchSpecifier(
42         pitch_segments=(
43             "a c' a a c",
44             "c' ef' g c' c' ef",
45             "a' c' a' c' c' g' a",
46         ),
47         ratio=(1, 1, 2),
48     ),
49     attachment_handler__trill_spanner=intermittent_trills,
50 )
51
52 ### SETTINGS ###
```

```

53
54 segment_maker.add_setting(
55     timespan_maker=armilla.materials.dense_timespan_maker,
56     viola_1=consort.CompositeMusicSpecifier(
57         primary_music_specifier=rh_circular,
58         rotation_indices=(1, 0, 1, 0, -1),
59         secondary_music_specifier=lh_glissandi,
60     ),
61     viola_2=consort.CompositeMusicSpecifier(
62         primary_music_specifier=rh_circular,
63         rotation_indices=(1, 0, 1, 0, -1),
64         secondary_music_specifier=lh_glissandi,
65     ),
66 )

```

### C.3.8 ARMILLA.SEGMENTS SEGMENT\_H THE\_LONG\_DUNE\_B

```

1 # -*- encoding: utf-8 -*-
2 import armilla
3 import consort
4 from abjad import new
5 from abjad.tools import durationtools
6 from abjad.tools import indicatortools
7 from abjad.tools import scoretools
8 from abjad.tools import selectortools
9
10
11 ### SEGMENT MAKER ###
12
13 segment_maker = armilla.ArmillaSegmentMaker(
14     desired_duration_in_seconds=60,
15     discard_final_silence=False,
16     name='The Long Dune (ii)',
17     repeat=False,
18     tempo=indicatortools.Tempo((1, 4), 36),
19 )
20
21 ### ATTACHMENTS ###
22
23 dietro_ponticello = consort.AttachmentExpression(
24     attachments=indicatortools.StringContactPoint('dietro ponticello'),
25     scope=scoretools.Voice,
26     selector=selectortools.Selector().by_leaves(),
27 )
28 dynamics_a = dynamic_expressions = consort.DynamicExpression(
29     dynamic_tokens='p mf p ppp f p ff',
30 )
31 dynamics_b = dynamic_expressions = consort.DynamicExpression(
32     dynamic_tokens='f p f mf ff p fff f fff mf fff',
33 )
34 intermittent_accents = armilla.materials.intermittent_accents
35 intermittent_circular = armilla.materials.intermittent_circular
36 intermittent_glissandi = armilla.materials.intermittent_glissandi
37 intermittent_tremoli = armilla.materials.intermittent_tremoli

```

```

38
39 ### MUSIC SPECIFIERS ###
40
41 lh_dietro = new(
42     armilla.materials.left_hand_dietro_music_specifier,
43     minimum_phrase_duration=durationtools.Duration(1, 4),
44 )
45 lh_diads = new(
46     armilla.materials.left_hand_diads_music_specifier,
47     minimum_phrase_duration=durationtools.Duration(1, 4),
48 )
49 rh_overpressure = new(
50     armilla.materials.right_hand_overpressure_music_specifier,
51     minimum_phrase_duration=durationtools.Duration(1, 4),
52 )
53 rh_pizzicati = armilla.materials.right_hand_pizzicati_music_specifier
54 lh_pizzicati = armilla.materials.left_hand_pizzicati_music_specifier
55
56 ### OVERPRESSURE ###
57
58 segment_maker.add_setting(
59     timespan_maker=armilla.materials.sustained_timespan_maker,
60     timespan_identifier=consort.RatioPartsExpression(
61         parts=(0,),
62         ratio=(2, 2, 1),
63     ),
64     viola_1=consort.CompositeMusicSpecifier(
65         discard_inner_offsets=True,
66         primary_music_specifier=new(
67             rh_overpressure,
68             attachment_handler__dynamic_expressions=dynamics_b,
69         ),
70         rotation_indices=(1, 0, 1, 0, -1),
71         secondary_music_specifier=lh_diads,
72     ),
73     viola_2=consort.CompositeMusicSpecifier(
74         discard_inner_offsets=True,
75         primary_music_specifier=new(
76             rh_overpressure,
77             attachment_handler__dynamic_expressions=dynamics_b,
78         ),
79         rotation_indices=(1, 0, 1, 0, -1),
80         secondary_music_specifier=lh_diads,
81     ),
82 )
83
84 segment_maker.add_setting(
85     timespan_maker=armilla.materials.dense_timespan_maker,
86     timespan_identifier=consort.RatioPartsExpression(
87         parts=(1,),
88         ratio=(2, 2, 1),
89     ),
90     viola_1=consort.CompositeMusicSpecifier(
91         primary_music_specifier=new(

```

```

92     rh_overpressure,
93     attachment_handler__articulations=intermittent_accents,
94     attachment_handler__dynamic_expressions=dynamics_a,
95     ),
96     rotation_indices=(1, 0, 1, 0, -1),
97     secondary_music_specifier=lh_diads,
98     ),
99     viola_2=consort.CompositeMusicSpecifier(
100     discard_inner_offsets=True,
101     primary_music_specifier=new(
102         rh_overpressure,
103         attachment_handler__articulations=intermittent_accents,
104         attachment_handler__dynamic_expressions=dynamics_a,
105         seed=1,
106         ),
107         rotation_indices=(1, 0, 1, 0, -1),
108         secondary_music_specifier=lh_diads,
109         ),
110     )
111
112 segment_maker.add_setting(
113     timespan_maker=armilla.materials.dense_timespan_maker,
114     timespan_identifier=consort.RatioPartsExpression(
115         parts=(2,),
116         ratio=(2, 2, 1),
117         ),
118     viola_1=consort.CompositeMusicSpecifier(
119         primary_music_specifier=new(
120             rh_overpressure,
121             attachment_handler__articulations=consort.AttachmentExpression(
122                 attachments=indicatortools.Articulation('>', 'down'),
123                 selector=selectortools.Selector().by_leaves()[:-1].flatten(),
124                 ),
125                 attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
126                 attachment_handler__bow_motion_technique_x=intermittent_circular,
127                 rhythm_maker__default_denominators=(4, 16, 4, 4, 4),
128                 seed=1,
129                 ),
130                 rotation_indices=(1, 0, 1, 0, -1),
131                 secondary_music_specifier=new(
132                     lh_diads,
133                     attachment_handler__glissando=intermittent_glissandi,
134                     ),
135                     ),
136     viola_2=consort.CompositeMusicSpecifier(
137         primary_music_specifier=new(
138             rh_overpressure,
139             attachment_handler__articulations=consort.AttachmentExpression(
140                 attachments=indicatortools.Articulation('>', 'down'),
141                 selector=selectortools.Selector().by_leaves()[:-1].flatten(),
142                 ),
143                 attachment_handler__stem_tremolo_spanner=intermittent_tremoli,
144                 attachment_handler__string_contact_points=dietro_ponticello,
145                 seed=2,

```

```

146         ),
147         rotation_indices=(1, 0, 1, 0, -1),
148         secondary_music_specifier=lh_dietro,
149     ),
150 )
151
152 ### PIZZICATI ###
153
154 segment_maker.add_setting(
155     timespan_maker=armilla.materials.sparse_timespan_maker,
156     timespan_identifier=consort.RatioPartsExpression(
157         parts=(1,),
158         ratio=(7, 1),
159     ),
160     viola_2=consort.CompositeMusicSpecifier(
161         primary_music_specifier=rh_pizzicati,
162         secondary_music_specifier=lh_pizzicati,
163     ),
164 )

```

## C.4 *ARMILLA*

### STYLESHEET

SOURCE

#### C.4.1 STYLESHEET.ILY

```

1 #(define-markup-command (vstrut layout props)
2   ())
3   #:category other
4   "
5 @cindex creating vertical space in text
6
7 Create a box of the same height as the current font."
8 (let ((ref-mrkp (interpret-markup layout props "fp")))
9   (ly:make-stencil (ly:stencil-expr empty-stencil)
10                 empty-interval
11                 (ly:stencil-extent ref-mrkp Y))))
12
13 afterGraceFraction = #(cons 1023 1024)
14 #(set-default-paper-size "17x11" 'landscape)
15 #(set-global-staff-size 11.5)
16
17 \paper {
18   bottom-margin = 10\mm
19   left-margin = 10\mm
20   right-margin = 10\mm
21   top-margin = 10\mm
22   evenFooterMarkup = \markup \fill-line {
23     \concat {
24       \bold \fontsize #3
25       \on-the-fly #print-page-number-check-first
26       \fromproperty #'page:page-number-string
27     }
28     "
29   }
30   evenHeaderMarkup = \markup \fill-line { " " }

```

```

31 oddFooterMarkup = \markup \fill-line {
32   " "
33   \concat {
34     \bold \fontsize #3
35     \on-the-fly #print-page-number-check-first
36     \fromproperty #'page:page-number-string
37   }
38 }
39 oddHeaderMarkup = \markup \fill-line { " " }
40 print-first-page-number = ##f
41 print-page-number = ##t
42 page-breaking = #ly:optimal-breaking
43 ragged-bottom = ##f
44 ragged-last-bottom = ##t
45 markup-system-spacing = #'(
46   (basic-distance . 0)
47   (minimum-distance . 12)
48   (padding . 0)
49   (stretchability . 0)
50 )
51 system-system-spacing = #'(
52   (basic-distance . 12)
53   (minimum-distance . 18)
54   (padding . 12)
55   (stretchability . 100)
56 )
57 top-markup-spacing = #'(
58   (basic-distance . 0)
59   (minimum-distance . 0)
60   (padding . 8)
61   (stretchability . 0)
62 )
63 top-system-spacing = #'(
64   (basic-distance . 0)
65   (minimum-distance . 10)
66   (padding . 0)
67   (stretchability . 0)
68 )
69 }

70
71 \layout {
72   \accidentalStyle forget
73   ragged-right = ##t
74
75   %% TIME SIGNATURE CONTEXT %%
76
77   \context {
78     \name TimeSignatureContext
79     \type Engraver_group
80     \consists Axis_group_engraver
81     \consists Bar_number_engraver
82     \consists Mark_engraver
83     \consists Metronome_mark_engraver
84     \consists Script_engraver

```

```

85  \consists Text_engraver
86  \consists Text_spacer_engraver
87  \consists Time_signature_engraver
88  \override BarNumber.Y-extent = #'(0 . 0)
89  \override BarNumber.Y-offset = 0
90  \override BarNumber.extra-offset = #'(-6 . 0)
91  \override BarNumber.font-name = "Didot Italic"
92  \override BarNumber.font-size = 1
93  \override BarNumber.padding = 4
94  \override BarNumber.stencil = #(make-stencil-circler 0.1 0.7 ly:text-interface::print)
95  \override MetronomeMark.X-extent = #'(0 . 0)
96  \override MetronomeMark.Y-extent = #'(0 . 0)
97  \override MetronomeMark.break-align-symbols = #'(left-edge)
98  \override MetronomeMark.extra-offset = #'(0 . 4)
99  \override MetronomeMark.font-size = 3
100 \override RehearsalMark.X-extent = #'(0 . 0)
101 \override RehearsalMark.X-offset = 6
102 \override RehearsalMark.Y-offset = -2.25
103 \override RehearsalMark.break-align-symbols = #'(time-signature)
104 \override RehearsalMark.break-visibility = #end-of-line-invisible
105 \override RehearsalMark.font-name = "Didot"
106 \override RehearsalMark.font-size = 8
107 \override RehearsalMark.outside-staff-priority = 500
108 \override RehearsalMark.self-alignment-X = #center
109 \override Script.extra-offset = #'(4 . -9)
110 \override Script.font-size = 6
111 \override TextScript.font-size = 3
112 \override TextScript.outside-staff-priority = 600
113 \override TextScript.padding = 6
114 \override TextSpanner.bound-details.right.attach-dir = #left
115 \override TextSpanner.padding = 6.75
116 \override TimeSignature.X-extent = #'(0 . 0)
117 \override TimeSignature.break-align-symbol = #'left-edge
118 \override TimeSignature.break-visibility = #end-of-line-invisible
119 \override TimeSignature.font-size = 3
120 \override TimeSignature.space-alist.clef = #'(extra-space . 0.5)
121 \override TimeSignature.style = #'numbered
122 \override VerticalAxisGroup.default-staff-staff-spacing = #'(
123   (basic-distance . 0)
124   (minimum-distance . 14)
125   (padding . 0)
126   (stretchability . 0)
127 )
128 \override VerticalAxisGroup.minimum-Y-extent = #'(-20 . 20)
129 }
130
131 %% DEFAULTS %%
132
133 \context {
134   \Voice
135   \remove Forbid_line_break_engraver
136 }
137 \context {
138   \Staff

```

```

139     \remove Time_signature_engraver
140 }
141
142 %%% BOWING %%%
143
144 \context {
145     \Voice
146     \name StringContactVoice
147     \type Engraver_group
148     \alias Voice
149     \override Beam.stencil = ##f
150     \override Dots.stencil = ##f
151     \override Flag.stencil = ##f
152     \override NoteHead.stencil = ##f
153     \override Rest.stencil = ##f
154     \override Stem.stencil = ##f
155     \override TextScript.staff-padding = 7
156     \override TextScript.self-alignment-X = #center
157     \override TextSpanner.staff-padding = 7
158     \override TextSpanner #'bound-details #'left #'attach-dir = 0
159     \override TextSpanner #'bound-details #'right #'attach-dir = 0
160     \override TupleBracket.stencil = ##f
161     \override TupleNumber.stencil = ##f
162 }
163 \context {
164     \Voice
165     \name BowContactVoice
166     \type Engraver_group
167     \alias Voice
168     \override Beam.stencil = ##f
169     \override Dots.stencil = ##f
170     \override Flag.stencil = ##f
171     \override NoteHead.extra-offset = #'(0.05 . 0)
172     \override Rest.stencil = ##f
173     \override Script.staff-padding = 2.5
174     \override Stem.stencil = ##f
175     \override TupleBracket.stencil = ##f
176     \override TupleNumber.stencil = ##f
177 }
178 \context {
179     \Voice
180     \name BowBeamingVoice
181     \type Engraver_group
182     \alias Voice
183     \override Beam.direction = #down
184     \override Beam.positions = #'(-11 . -11)
185     \override Dots.X-offset = -8
186     \override Dots.staff-position = -1
187     \override Flag.Y-offset = -10
188     \override NoteHead.Y-offset = -5
189     \override NoteHead.stencil = ##f
190     \override Stem.direction = #down
191     \override Stem.length = 9
192     \override Stem.stem-begin-position = -11

```

```

193     \override TupletBracket.positions = #'(-13 . -13)
194 }
195 \context {
196     \Dynamics
197     \remove Bar_engraver
198     \override DynamicLineSpanner.staff-padding = 11.5
199     \override DynamicText.self-alignment-X = -1
200     \override Hairpin.bound-padding = 1.5
201     \%override Hairpin.minimum-length = 5
202     \override VerticalAxisGroup.nonstaff-relatedstaff-spacing = #'(
203         (basic-distance . 5)
204         (padding . 2.5)
205     )
206 }
207 \context {
208     \Staff
209     \name BowingStaff
210     \type Engraver_group
211     \alias Staff
212     \accepts BowBeamingVoice
213     \accepts BowContactVoice
214     \accepts StringContactVoice
215     \override Glissando.bound-details.left.padding = 0.75
216     \override Glissando.bound-details.right.padding = 0.75
217     \override Glissando.thickness = 2
218     \override Glissando.zigzag-length = 1.5
219     \override Glissando.zigzag-width = 0.75
220     \override ParenthesesItem.font-size = 1
221     \override ParenthesesItem.padding = 0.1
222     \override StaffSymbol.transparent = ##t
223 }
224
225 %%% FINGERING %%%
226
227 \context {
228     \Voice
229     \name FingeringPitchesVoice
230     \type Engraver_group
231     \alias Voice
232     \override Beam.direction = #down
233     \override Beam.positions = #'(-9 . -9)
234     \override Stem.direction = #down
235     \%override Tie.stencil = ##f
236     \override Glissando.stencil = ##f
237     \override TupletBracket.positions = #'(-11 . -11)
238 }
239 \context {
240     \Voice
241     \name FingeringSpannerVoice
242     \type Engraver_group
243     \alias Voice
244     \override Beam.direction = #down
245     \override Beam.stencil = ##f
246     \override Dots.transparent = ##t

```

```

247     \override Flag.stencil = ##f
248     \override NoteHead.transparent = ##t
249     \override Stem.direction = #down
250     \override Stem.stencil = ##f
251     \override Tie.stencil = ##f
252     \override TupletBracket.stencil = ##f
253     \override TupletNumber.stencil = ##f
254 }
255 \context {
256     \Staff
257     \name FingeringStaff
258     \type Engraver_group
259     \alias Staff
260     \accepts FingeringPitchesVoice
261     \accepts FingeringSpannerVoice
262     \override Glissando.bound-details.left.padding = 1.5
263     \override Glissando.bound-details.right.padding = 1.5
264     \override Glissando.thickness = 2
265     \override StaffSymbol.color = #(x11-color 'grey50)
266 }
267
268 %%% STRING PERFORMER GROUP %%%%
269
270 \context {
271     \StaffGroup
272     \name StringPerformerGroup
273     \type Engraver_group
274     \alias StaffGroup
275     \accepts BowingStaff
276     \accepts FingeringStaff
277 }
278
279 %%% SCORE %%%
280
281 \context {
282     \Score
283     \accepts TimeSignatureContext
284     \accepts StringPerformerGroup
285     \remove Bar_number_engraver
286     \remove Mark_engraver
287     \remove Metronome_mark_engraver
288     \override BarLine.bar-extent = #'(-2 . 2)
289     \override BarLine.hair-thickness = 0.5
290     \override BarLine.space-alist = #'(
291         (time-signature extra-space . 0.0)
292         (custos minimum-space . 0.0)
293         (clef minimum-space . 0.0)
294         (key-signature extra-space . 0.0)
295         (key-cancellation extra-space . 0.0)
296         (first-note fixed-space . 0.0)
297         (next-note semi-fixed-space . 0.0)
298         (right-edge extra-space . 0.0)
299     )
300     \override Beam.beam-thickness = 0.75

```

```

301 \override Beam.breakable = ##t
302 \override Beam.length-fraction = 1.5
303 \override DynamicLineSpanner.add-stem-support = ##t
304 \override DynamicLineSpanner.outside-staff-padding = 1
305 \override Glissando.after-line-breaking = ##t
306 \override Glissando.breakable = ##t
307 \override Glissando.thickness = 3
308 \override GraceSpacing.common-shortest-duration = #(ly:make-moment 1 8)
309 \override NoteCollision.merge-differently-dotted = ##t
310 \override NoteColumn.ignore-collision = ##t
311 \override OttavaBracket.add-stem-support = ##t
312 \override OttavaBracket.padding = 2
313 \override SpacingSpanner.base-shortest-duration = #(ly:make-moment 1 64)
314 \override SpacingSpanner.strict-grace-spacing = ##f
315 \override SpacingSpanner.strict-note-spacing = ##f
316 \override SpacingSpanner.uniform-stretching = ##t
317 \override Stem.details.beamed-lengths = #'(6)
318 \override Stem.details.lengths = #'(6)
319 \override Stem.stemlet-length = 1.5
320 \override StemTremolo.beam-width = 1.5
321 \override StemTremolo.flag-count = 4
322 \override StemTremolo.slope = 0.5
323 \override StemTremolo.style = #'default
324 \override SustainPedal.self-alignment-X = #CENTER
325 \override SustainPedalLineSpanner.padding = 2
326 \override SustainPedalLineSpanner.to-barline = ##t
327 \override TextScript.add-stem-support = ##t
328 \override TextScript.outside-staff-padding = 1
329 \override TextScript.padding = 1
330 \override TextScript.staff-padding = 1
331 \override TextSpanner.bound-details.right.padding = 2.5
332 \override TrillPitchAccidental.avoid-slur = #'ignore
333 \override TrillPitchAccidental.layer = 1000
334 \override TrillPitchAccidental.whiteout = ##t
335 \override TrillPitchHead.layer = 1000
336 \override TrillPitchHead.whiteout = ##t
337 \override TrillSpanner.bound-details.right.padding = 1
338 \override TrillSpanner.outside-staff-padding = 1
339 \override TrillSpanner.padding = 1
340 \override TupletBracket.avoid-scripts = ##t
341 \override TupletBracket.direction = #down
342 \override TupletBracket.full-length-to-extent = ##t
343 \override TupletBracket.outside-staff-padding = 2
344 \override TupletBracket.padding = 2
345 \override TupletNumber.font-size = 1
346 \override TupletNumber.text = #tuple-number::calc-fraction-text
347 \override VerticalAxisGroup.staff-staff-spacing = #'(
348     (basic-distance . 8)
349     (minimum-distance . 14)
350     (padding . 4)
351     (stretchability . 0)
352 )
353 autoBeaming = ##f
354 doubleRepeatType = #":|.|:"
```

```
355     proportionalNotationDuration = #(ly:make-moment 1 16)
356     tupletFullLength = ##t
357 }
358 }
```

# D

## *ersilia* source code

### D.1 *ERSILIA*

MAKERS

SOURCE

#### D.1.1 ERSILIA.MAKERS.ERSILIASCORETEMPLATE

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad import attach
4 from abjad.tools import instrumenttools
5 from abjad.tools import markuptools
6 from abjad.tools import scoretools
7
8
9 class ErsiliaScoreTemplate(consort.ScoreTemplate):
10     r'''Ersilia score template.
11
12     :::
13
14     >>> import ersilia
15     >>> template = ersilia.makers.ErsiliaScoreTemplate()
16     >>> score = template()
17     >>> print(format(score))
18     \context Score = "Ersilia Score" <-
19         \tag #'time
20         \context TimeSignatureContext = "Time Signature Context" {
21             }
22         \context WindSectionStaffGroup = "Wind Section Staff Group" <-
23             \tag #'flute
24             \context FluteStaff = "Flute Staff" {
25                 \clef "treble"
26                 \set Staff.instrumentName = \markup { Flute }
27                 \set Staff.shortInstrumentName = \markup { Fl. }
28                 \context Voice = "Flute Voice" {
```

```

29         }
30     }
31     \tag #'oboe
32     \context OboeStaff = "Oboe Staff" {
33         \clef "treble"
34         \set Staff.instrumentName = \markup { Oboe }
35         \set Staff.shortInstrumentName = \markup { Ob. }
36         \context Voice = "Oboe Voice" {
37             }
38         }
39     \tag #'clarinet
40     \context ClarinetStaff = "Clarinet Staff" {
41         \clef "treble"
42         \set Staff.instrumentName = \markup {
43             \right-column
44             {
45                 Bass
46                 Clarinet
47             }
48         }
49         \set Staff.shortInstrumentName = \markup { Bass cl. }
50         \context Voice = "Clarinet Voice" {
51             }
52         }
53     \tag #'saxophone
54     \context SaxophoneStaff = "Saxophone Staff" {
55         \clef "treble"
56         \set Staff.instrumentName = \markup {
57             \right-column
58             {
59                 Baritone
60                 Saxophone
61             }
62         }
63         \set Staff.shortInstrumentName = \markup { Bar. sax. }
64         \context Voice = "Saxophone Voice" {
65             }
66         }
67     >>
68     \context PercussionSectionStaffGroup = "Percussion Section Staff Group" <<
69     \tag #'guitar
70     \context GuitarStaffGroup = "Guitar Staff Group" <<
71     \context PitchPipes = "Guitar Pitch Pipe Staff" {
72         \clef "percussion"
73         \set Staff.instrumentName = \markup {
74             \right-column
75             {
76                 Pitch
77                 Pipes
78             }
79         }
80         \set Staff.shortInstrumentName = \markup { Pp. }
81         \context Voice = "Guitar Pitch Pipe Voice" {
82             }

```

```

83
84     }
85     \context GuitarStaff = "Guitar Staff" {
86         \clef "treble_8"
87         \set Staff.instrumentName = \markup { Guitar }
88         \set Staff.shortInstrumentName = \markup { Gt. }
89         \context Voice = "Guitar Voice" {
90             }
91         }
92     >>
93     \tag #'piano
94     \context PianoStaffGroup = "Piano Staff Group" <<
95         \context PitchPipes = "Piano Pitch Pipe Staff" {
96             \clef "percussion"
97             \set Staff.instrumentName = \markup {
98                 \right-column
99                 {
100                     Pitch
101                     Pipes
102                 }
103             }
104             \set Staff.shortInstrumentName = \markup { Pp. }
105             \context Voice = "Piano Pitch Pipe Voice" {
106                 }
107             }
108             \context PianoStaff = "Piano Staff" <<
109                 \set PianoStaff.instrumentName = \markup { Piano }
110                 \set PianoStaff.shortInstrumentName = \markup { Pf. }
111                 \context PianoUpperStaff = "Piano Upper Staff" {
112                     \clef "treble"
113                     \context Voice = "Piano Upper Voice" {
114                         }
115                     }
116                     \context PianoLowerStaff = "Piano Lower Staff" {
117                         \clef "bass"
118                         \context Voice = "Piano Lower Voice" {
119                             }
120                         \context Dynamics = "Piano Pedals Voice" {
121                             }
122                     >>
123     >>
124     \tag #'percussion
125     \context PercussionStaffGroup = "Percussion Staff Group" <<
126         \context PitchPipes = "Percussion Pitch Pipe Staff" {
127             \clef "percussion"
128             \set Staff.instrumentName = \markup {
129                 \right-column
130                 {
131                     Pitch
132                     Pipes
133                 }
134             }
135             \set Staff.shortInstrumentName = \markup { Pp. }
136             \context Voice = "Percussion Pitch Pipe Voice" {

```

```

137         }
138     }
139     \context PercussionStaff = "Percussion Staff" {
140         \clef "percussion"
141         \set Staff.instrumentName = \markup { Percussion }
142         \set Staff.shortInstrumentName = \markup { Perc. }
143         \context Voice = "Percussion Voice" {
144             }
145         }
146     >>
147     >>
148     \context StringSectionStaffGroup = "String Section Staff Group" <<
149         \tag #'violin
150         \context ViolinStaff = "Violin Staff" {
151             \clef "treble"
152             \set Staff.instrumentName = \markup { Violin }
153             \set Staff.shortInstrumentName = \markup { Vn. }
154             \context Voice = "Violin Voice" {
155                 }
156             }
157         \tag #'viola
158         \context ViolaStaff = "Viola Staff" {
159             \clef "alto"
160             \set Staff.instrumentName = \markup { Viola }
161             \set Staff.shortInstrumentName = \markup { Va. }
162             \context Voice = "Viola Voice" {
163                 }
164             }
165         \tag #'cello
166         \context CelloStaff = "Cello Staff" {
167             \clef "bass"
168             \set Staff.instrumentName = \markup { Cello }
169             \set Staff.shortInstrumentName = \markup { Vc. }
170             \context Voice = "Cello Voice" {
171                 }
172             }
173         \tag #'contrabass
174         \context ContrabassStaffGroup = "Contrabass Staff Group" <<
175             \context PitchPipes = "Contrabass Pitch Pipe Staff" {
176                 \clef "percussion"
177                 \set Staff.instrumentName = \markup {
178                     \right-column
179                     {
180                         Pitch
181                         Pipes
182                     }
183                 }
184                 \set Staff.shortInstrumentName = \markup { Pp. }
185                 \context Voice = "Contrabass Pitch Pipe Voice" {
186                     }
187                 }
188             \context ContrabassStaff = "Contrabass Staff" {
189                 \clef "bass_8"
190                 \set Staff.instrumentName = \markup { Contrabass }

```

```

191         \set Staff.shortInstrumentName = \markup { Cb. }
192         \context Voice = "Contrabass Voice" {
193             }
194         }
195     >>
196     >>
197     >>
198     :: 
199
200
201     >>> for item in sorted(template.context_name_abbreviations.items()):
202         ...
203         item
204         ...
205         ('bass', 'Contrabass Voice')
206         ('bass_pp', 'Contrabass Pitch Pipe Voice')
207         ('cello', 'Cello Voice')
208         ('clarinet', 'Clarinet Voice')
209         ('flute', 'Flute Voice')
210         ('guitar', 'Guitar Voice')
211         ('guitar_pp', 'Guitar Pitch Pipe Voice')
212         ('oboe', 'Oboe Voice')
213         ('percussion', 'Percussion Voice')
214         ('percussion_pp', 'Percussion Pitch Pipe Voice')
215         ('piano_lh', 'Piano Lower Voice')
216         ('piano_pedals', 'Piano Pedals Voice')
217         ('piano_pp', 'Piano Pitch Pipe Voice')
218         ('piano_rh', 'Piano Upper Voice')
219         ('saxophone', 'Saxophone Voice')
220         ('viola', 'Viola Voice')
221         ('violin', 'Violin Voice')
222     :: 
223
224     >>> for item in template.composite_context_pairs.items():
225         ...
226         item
227         ...
228     ...
229
230     ### CLASS VARIABLES ###
231
232     __slots__ = ()
233
234     ### SPECIAL METHODS ###
235
236     def __call__(self):
237
238         pitch_pipes = instrumenttools.Percussion(
239             instrument_name='pitch pipes',
240             instrument_name_markup=markuptools.Markup.right_column(
241                 ['Pitch', 'Pipes'],
242                 direction=None,
243                 ),
244             short_instrument_name='pp.',
```

```

245     )
246
247     time_signature_context = scoretools.Context(
248         context_name='TimeSignatureContext',
249         name='Time Signature Context',
250         )
251     self._attach_tag('time', time_signature_context)
252
253     flute_staff = self._make_staff(
254         'Flute', 'treble',
255         instrument=instrumenttools.Flute(),
256         tag='flute',
257         )
258
259     oboe_staff = self._make_staff(
260         'Oboe', 'treble',
261         instrument=instrumenttools.Oboe(),
262         tag='oboe',
263         )
264
265     clarinet_staff = self._make_staff(
266         'Clarinet', 'treble',
267         instrument=instrumenttools.BassClarinet(
268             instrument_name_markup=markuptools.Markup.right_column(
269                 ['Bass', 'Clarinet'],
270                 direction=None,
271                 ),
272                 ),
273         tag='clarinet',
274         )
275
276     saxophone_staff = self._make_staff(
277         'Saxophone', 'treble',
278         instrument=instrumenttools.BaritoneSaxophone(
279             instrument_name_markup=markuptools.Markup.right_column(
280                 ['Baritone', 'Saxophone'],
281                 direction=None,
282                 ),
283                 ),
284         tag='saxophone',
285         )
286
287     wind_section_staff_group = scoretools.StaffGroup(
288         [
289             flute_staff,
290             oboe_staff,
291             clarinet_staff,
292             saxophone_staff,
293             ],
294             context_name='WindSectionStaffGroup',
295             name='Wind Section Staff Group',
296             )
297
298     guitar_staff = self._make_staff(

```

```

299     'Guitar', 'treble_8',
300     instrument=instrumenttools.Guitar(),
301 )
302 guitar_aux_staff = self._make_staff(
303     'Guitar Pitch Pipe', 'percussion',
304     abbreviation='guitar_pp',
305     context_name='Pitch Pipes',
306     instrument=pitch_pipes,
307 )
308 guitar_staff_group = scoretools.StaffGroup(
309     [guitar_aux_staff, guitar_staff],
310     name='Guitar Staff Group',
311     context_name='GuitarStaffGroup',
312 )
313 self._attach_tag('guitar', guitar_staff_group)
314
315 piano_aux_staff = self._make_staff(
316     'Piano Pitch Pipe', 'percussion',
317     abbreviation='piano_pp',
318     context_name='Pitch Pipes',
319     instrument=pitch_pipes,
320 )
321 piano_rh_staff = self._make_staff(
322     'Piano Upper', 'treble',
323     abbreviation='piano_rh',
324 )
325 piano_lh_staff = self._make_staff(
326     'Piano Lower', 'bass',
327     abbreviation='piano_lh',
328 )
329 piano_pedals = self._make_voice(
330     'Piano Pedals',
331     context_name='Dynamics',
332 )
333 piano_staff = scoretools.StaffGroup(
334     [piano_rh_staff, piano_lh_staff, piano_pedals],
335     context_name='PianoStaff',
336     name='Piano Staff',
337 )
338 attach(instrumenttools.Piano(), piano_staff)
339 piano_staff_group = scoretools.StaffGroup(
340     [piano_aux_staff, piano_staff],
341     context_name='PianoStaffGroup',
342     name='Piano Staff Group',
343 )
344 self._attach_tag('piano', piano_staff_group)
345
346 percussion_staff = self._make_staff(
347     'Percussion', 'percussion',
348     instrument=instrumenttools.Percussion(),
349 )
350 percussion_aux_staff = self._make_staff(
351     'Percussion Pitch Pipe', 'percussion',
352     abbreviation='percussion_pp',

```

```

353     context_name='Pitch Pipes',
354     instrument=pitch_pipes,
355     )
356 percussion_staff_group = scoretools.StaffGroup(
357     [percussion_aux_staff, percussion_staff],
358     name='Percussion Staff Group',
359     context_name='PercussionStaffGroup',
360     )
361 self._attach_tag('percussion', percussion_staff_group)
362
363 percussion_section_staff_group = scoretools.StaffGroup(
364     [
365         guitar_staff_group,
366         piano_staff_group,
367         percussion_staff_group,
368         ],
369         context_name='PercussionSectionStaffGroup',
370         name='Percussion Section Staff Group',
371         )
372
373 violin_staff = self._make_staff(
374     'Violin', 'treble',
375     instrument=instrumenttools.Violin(),
376     tag='violin',
377     )
378
379 viola_staff = self._make_staff(
380     'Viola', 'alto',
381     instrument=instrumenttools.Viola(),
382     tag='viola',
383     )
384
385 cello_staff = self._make_staff(
386     'Cello', 'bass',
387     instrument=instrumenttools.Cello(),
388     tag='cello',
389     )
390
391 contrabass_aux_staff = self._make_staff(
392     'Contrabass Pitch Pipe',
393     'percussion',
394     abbreviation='bass_pp',
395     context_name='Pitch Pipes',
396     instrument=pitch_pipes,
397     )
398 contrabass_staff = self._make_staff(
399     'Contrabass', 'bass_8',
400     abbreviation='bass',
401     instrument=instrumenttools.Contrabass(
402         pitch_range='[E1, G4]',
403         ),
404     )
405 contrabass_staff_group = scoretools.StaffGroup(
406     [contrabass_aux_staff, contrabass_staff],

```

```

407         name='Contrabass Staff Group',
408         context_name='ContrabassStaffGroup',
409         )
410     self._attach_tag('contrabass', contrabass_staff_group)
411
412     string_section_staff_group = scoretools.StaffGroup(
413     [
414         violin_staff,
415         viola_staff,
416         cello_staff,
417         contrabass_staff_group,
418         ],
419         context_name='StringSectionStaffGroup',
420         name='String Section Staff Group',
421         )
422
423     score = scoretools.Score(
424     [
425         time_signature_context,
426         wind_section_staff_group,
427         percussion_section_staff_group,
428         string_section_staff_group,
429         ],
430         name='Ersilia Score',
431         )
432
433     return score

```

### D.1.2 ERSILIA.MAKERS.ERSILIASegmentMaker

```

1 # -*- encoding: utf-8 -*-
2 import collections
3 from abjad import iterate
4 from abjad.tools import durationtools
5 from abjad.tools import markuptools
6 from abjad.tools import scoretools
7 from abjad.tools import systemtools
8 import consort
9
10
11 class ErsiliaSegmentMaker(consort.SegmentMaker):
12
13     ### CLASS VARIABLES ###
14
15     __slots__ = ()
16
17     ### INITIALIZER ###
18
19     def __init__(
20         self,
21         annotate_colors=None,
22         annotate_phrasing=None,
23         annotate_timepans=None,
24         desired_duration_in_seconds=None,

```

```

25     discard_final_silence=None,
26     maximum_meter_run_length=None,
27     name=None,
28     permitted_time_signatures=None,
29     repeat=None,
30     score_template=None,
31     settings=None,
32     tempo=None,
33     ):
34     import ersilia
35     score_template = score_template or ersilia.ErsiliaScoreTemplate()
36     consort.SegmentMaker.__init__(
37         self,
38         annotate_colors=annotate_colors,
39         annotate_phrasing=annotate_phrasing,
40         annotate_timespans=annotate_timespans,
41         desired_duration_in_seconds=desired_duration_in_seconds,
42         discard_final_silence=discard_final_silence,
43         maximum_meter_run_length=maximum_meter_run_length,
44         name=name,
45         permitted_time_signatures=permitted_time_signatures,
46         repeat=repeat,
47         score_template=score_template,
48         settings=settings,
49         tempo=tempo,
50         timespan_quantization=durationtools.Duration(1, 8),
51     )
52
53     ### PUBLIC METHODS ###
54
55     @staticmethod
56     def validate_score(score, verbose=True):
57         consort.SegmentMaker.validate_score(score, verbose=verbose)
58         component = score['Piano Staff']
59         progress_indicator = systemtools.ProgressIndicator(
60             is_warning=True,
61             message='    coloring piano conflicts',
62             verbose=verbose,
63         )
64         with progress_indicator:
65             for vertical_moment in iterate(component).by_vertical_moment():
66                 pitch_numbers = collections.Counter()
67                 notes_and_chords = vertical_moment.notes_and_chords
68                 for note_or_chord in notes_and_chords:
69                     if isinstance(note_or_chord, scoretools.Note):
70                         pitch_number = note_or_chord.written_pitch.pitch_number
71                         pitch_number = float(pitch_number)
72                         pitch_numbers[pitch_number] += 1
73                     else:
74                         for pitch in note_or_chord.written_pitches:
75                             pitch_number = pitch.pitch_number
76                             pitch_number = float(pitch_number)
77                             pitch_numbers[pitch_number] += 1
78                 conflict_pitch_numbers = set()

```

```

79         for pitch_number, count in pitch_numbers.iteritems():
80             if 1 < count:
81                 conflict_pitch_numbers.add(pitch_number)
82             if not conflict_pitch_numbers:
83                 continue
84             for note_or_chord in notes_and_chords:
85                 if isinstance(note_or_chord, scoretools.Note):
86                     pitch_number = note_or_chord.written_pitch.pitch_number
87                     pitch_number = float(pitch_number)
88                     if pitch_number in conflict_pitch_numbers:
89                         #note_or_chord.note_head.tweak.color = 'red'
90                         progress_indicator.advance()
91                     else:
92                         for note_head in note_or_chord.note_heads:
93                             pitch_number = note_head.written_pitch.pitch_number
94                             pitch_number = float(pitch_number)
95                             if pitch_number in conflict_pitch_numbers:
96                                 #note_head.tweak.color = 'red'
97                                 progress_indicator.advance()
98
99     ### PUBLIC PROPERTIES ###
100
101     @property
102     def final_markup(self):
103         portland = markuptools.Markup('Portland, OR')
104         date = markuptools.Markup('January 2015 - April 2015')
105         null = markuptools.Markup.null()
106         contents = [
107             null,
108             null,
109             null,
110             portland,
111             date,
112         ]
113         markup = markuptools.Markup.right_column(contents)
114         markup = markup.italic()
115         return markup
116
117     @property
118     def score_package_name(self):
119         return 'ersilia'
```

### D.1.3 ERSILIA.MAKERS.PERCUSION

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import abctools
3 from abjad.tools import pitchtools
4
5
6 class Percussion(abctools.AbjadObject):
7
8     WOOD_BLOCK_5 = pitchtools.NamedPitch('G4')
9     WOOD_BLOCK_4 = pitchtools.NamedPitch('E4')
10    WOOD_BLOCK_3 = pitchtools.NamedPitch('C4')
```

```

11 WOOD_BLOCK_2 = pitchtools.NamedPitch('A3')
12 WOOD_BLOCK_1 = pitchtools.NamedPitch('F3')

13
14 TOM_4 = pitchtools.NamedPitch('F4')
15 TOM_3 = pitchtools.NamedPitch('D4')
16 TOM_2 = pitchtools.NamedPitch('B3')
17 TOM_1 = pitchtools.NamedPitch('G3')
18
19 BAMBOO_WIND_CHIMES = pitchtools.NamedPitch('F4')
20 SNARE_DRUM = pitchtools.NamedPitch('D4')
21 TAM_TAM = pitchtools.NamedPitch('B3')
22 BASS_DRUM = pitchtools.NamedPitch('G3')

```

## D.2 *ERSILIA* MATERIALS SOURCE

### D.2.1 ERSILIA.MATERIALS.ABBREVIATIONS

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import markuptools
5 from abjad.tools import pitchtools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8
9
10 laissez_vibrer = consort.AttachmentExpression(
11     attachments=[
12         [
13             indicatortools.LaissezVibrer(),
14             markuptools.Markup('L.V', Up)
15             .caps()
16             .tiny()
17             .pad_around(0.5)
18             .box()
19             .pad_around(0.5)
20         ],
21     ],
22     selector=selectortools.Selector()
23     .by_logical_tie(pitched=True)
24     .by_contiguity()
25     .by_length('==', 1)
26     .by_leaves()
27     [0]
28 )
29
30 black_keys_spanner = spannertools.make_solid_text_spanner_with_nib(
31     markuptools.Markup.flat().vcenter(),
32 )
33
34
35 chromatic_keys_spanner = spannertools.make_solid_text_spanner_with_nib(
36     markuptools.Markup.concat([
37         markuptools.Markup.natural(),

```

```

38     markuptools.Markup.hspace(0.1),
39     markuptools.Markup.flat(),
40   ]).vcenter(),
41 )
42
43
44 white_keys_spinner = spanertools.make_solid_text_spinner_with_nib(
45   markuptools.Markup.natural().vcenter(),
46 )
47
48
49 percussion_staff = consort.AttachmentExpression(
50   attachments=[
51     [
52       spanertools.StaffLinesSpanner(
53         lines=[-4, 4],
54         overrides={
55           'note_head__style': 'cross',
56           'note_head__no_ledgers': True,
57         },
58       ),
59       consort.ClefSpanner('percussion'),
60     ],
61   ],
62 )
63
64 agitato_pitch_specifier = consort.PitchSpecifier(
65   pitch_segments=[
66     pitchtools.PitchClassSegment([0, 3, 2, 5, 11, 1]),
67     pitchtools.PitchClassSegment([11, 9]),
68     pitchtools.PitchClassSegment([2, 4, 5, 8]),
69     pitchtools.PitchClassSegment([0, 3, 5]),
70     pitchtools.PitchClassSegment([2, 4, 5, 8]),
71   ],
72   ratio=[1, 2, 1, 2, 1],
73 )
74
75
76 pitch_operation_specifier = consort.PitchOperationSpecifier(
77   pitch_operations=[
78     pitchtools.Rotation(1),
79     None,
80     pitchtools.PitchOperation([
81       pitchtools.Transposition(1),
82       pitchtools.Inversion(),
83     ]),
84     None,
85     pitchtools.Rotation(-1),
86     pitchtools.Retrogression(),
87   ],
88   ratio=(1, 3, 1, 1, 2, 1),
89 )
90
91

```

```

92 def make_text_markup(text):
93     markup = markuptools.Markup.concat([
94         markuptools.Markup(r'\vstrut'),
95         markuptools.Markup(text),
96     ])
97     markup = markup.smaller().italic().pad_around(0.5).whiteout().box()
98     markup = markuptools.Markup(markup, Up)
99     return markup
100
101
102 def make_text_spanner(text):
103     markup_contents = make_text_markup(text).contents
104     markup = markuptools.Markup(markup_contents)
105     text_spanner = consort.ComplexTextSpanner(markup=markup)
106     return text_spanner
107
108
109 guitar_chords = tuple(
110     pitchtools.PitchSegment(_) for _ in (
111         "d c' f' a''",
112         "df bf e' a''",
113         "c g bf ef' a''",
114         "b, gf a d' af''",
115         "c g b e' a''",
116         "f bf ef' g' c'''",
117         "e a d' fs' b''",
118         "ef af df' f' bf''",
119         "d g c' e' a''",
120         "d b d' f' a''",
121         "d f c' d' g''",
122         "d f b d' g''",
123     )
124 )
125
126 __all__ = [
127     'agitato_pitchSpecifier',
128     'black_keysSpanner',
129     'chromatic_keysSpanner',
130     'guitar_chords',
131     'laissez_vibrer',
132     'make_text_markup',
133     'make_text_spanner',
134     'percussion_staff',
135     'pitch_operationSpecifier',
136     'white_keysSpanner',
137 ]

```

## D.2.2 ERSILIA.MATERIALS.DENSE\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad import *
4
5

```

```

6 dense_timespan_maker = consort.TaleaTimespanMaker(
7     initial_silence_talea=rhythmmakertools.Talea(
8         counts=(0, 1, 2, 1, 3),
9         denominator=8,
10    ),
11    playing_talea=rhythmmakertools.Talea(
12        counts=(3, 4, 2, 2, 3, 3, 2),
13        denominator=8,
14    ),
15    playing_groupings=(2, 1, 2, 3, 1, 1, 2, 2),
16    repeat=True,
17    silence_talea=rhythmmakertools.Talea(
18        counts=(1, 2, 3, 1, 2, 5),
19        denominator=8,
20    ),
21    step_anchor=Right,
22    synchronize_groupings=False,
23    synchronize_step=False,
24    timespanSpecifier=consort.TimespanSpecifier(
25        minimum_duration=durationaltools.Duration(1, 8),
26    ),
27 )

```

### D.2.3 ERSILIA.MATERIALS.GUITAR\_AGITATO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import scoretools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8 import consort
9
10
11 guitar_agitato_music_specifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         accents=consort.AttachmentExpression(
14             attachments=indicatortools.Articulation('accent'),
15             selector=selectortools.Selector()
16             .by_logical_tie(pitched=True)
17             .by_duration('==', (1, 8), preprolated=True)
18             [0],
19         ),
20         dynamic_expressions=consort.DynamicExpression(
21             dynamic_tokens='mf mp fff',
22             start_dynamic_tokens='f',
23             stop_dynamic_tokens='mf',
24         ),
25         mordent=consort.AttachmentExpression(
26             attachments=indicatortools.Articulation('mordent'),
27             selector=selectortools.Selector()
28             .by_logical_tie(pitched=True)
29             .by_duration('>', (1, 8), preprolated=True)

```

```

30     .by_class(scoretools.Note)
31     [0],
32   ),
33 snappizzicato=consort.AttachmentExpression(
34   attachments=indicatortools.Articulation('snappizzicato'),
35   selector=selectortools.Selector()
36   .by_logical_tie(pitched=True)
37   .by_duration('==', (1, 16), preprolated=True)
38   .by_contiguity()
39   .by_leaves()
40   [0]
41   .flatten()
42   ),
43 staccati=consort.AttachmentExpression(
44   attachments=indicatortools.Articulation('staccato'),
45   selector=selectortools.Selector()
46   .by_logical_tie(pitched=True)
47   .by_duration('==', (1, 16), preprolated=True)
48   [0],
49   ),
50 tremoli=consort.AttachmentExpression(
51   attachments=spannertools.StemTremoloSpanner(),
52   selector=selectortools.Selector()
53   .by_logical_tie(pitched=True)
54   .by_duration('>=', (1, 8), preprolated=True)
55   .by_class(scoretools.Chord)
56   [0],
57   ),
58   ),
59 color='magenta',
60 labels=[],
61 pitch_handler=consort.PitchClassPitchHandler(
62   forbid_repetitions=True,
63   leap_constraint=12,
64   logical_tie_expressions=[
65     None,
66     consort.ChordExpression([0, 3]),
67     None,
68     None,
69     consort.ChordExpression([0, 5]),
70     ],
71   pitchSpecifier=abbreviations.agitato_pitchSpecifier,
72   registerSpecifier=consort.RegisterSpecifier(
73     base_pitch='A2',
74     phrase_inflections=consort.RegisterInflection
75     .zigzag(6)
76     .reverse()
77     .align(),
78     segment_inflections=consort.RegisterInflection
79     .descending(width=12)
80     .align()
81     ),
82   ),
83 rhythm_maker=rhythmmakertools.TaleaRhythmMaker(

```

```

84     burnishSpecifier=rhythmmakertools.BurnishSpecifier(
85         left_classes=[scoretools.Rest],
86         left_counts=[1, 0],
87         right_classes=[scoretools.Rest],
88         right_counts=[1],
89         ),
90     extra_counts_per_division=[0, 0, 1],
91     output_masks=[
92         rhythmmakertools.SustainMask(
93             indices=[0],
94             period=3,
95             ),
96         ],
97     talea=rhythmmakertools.Talea(
98         counts=[1, 1, 1, 1, 2],
99         denominator=16,
100        ),
101    ),
102 )

```

#### D.2.4 ERSILIA.MATERIALS.GUITAR\_CONTINUO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import scoretools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8
9
10 guitar_continuo_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         accents=consort.AttachmentExpression(
13             attachments=indicatortools.Articulation('accent'),
14             selector=selectortools.Selector()
15             .by_leaves()
16             .by_run(scoretools.Note)
17             .by_counts(
18                 [3, 3, 4],
19                 cyclic=True,
20                 overhang=True,
21                 fuse_overhang=True,
22                 )
23             [0],
24             ),
25         dynamic_expressions=consort.DynamicExpression(
26             start_dynamic_tokens='p mp',
27             only_first=True,
28             ),
29         slur=consort.AttachmentExpression(
30             attachments=spannertools.Slur(),
31             selector=selectortools.Selector()
32             .by_leaves())

```

```

33     .by_run(scoretools.Note)
34     .by_counts(
35         [3, 3, 4],
36         cyclic=True,
37         overhang=True,
38         fuse_overhang=True,
39         )
40     ),
41     ),
42 pitch_handler=consort.AbsolutePitchHandler(
43     deviations=[0, 2, 0, 3, 0, 3, 0, 2, 0, 5, 0, 3, 0, 5],
44     pitchSpecifier="d' f' df",
45     pitchApplicationRate='phrase',
46     ),
47 rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
48     burnishSpecifier=rhythmmakertools.BurnishSpecifier(
49         outerDivisionsOnly=True,
50         leftClasses=[scoretools.Rest],
51         leftCounts=[1, 1, 0],
52         rightClasses=[scoretools.Rest],
53         rightCounts=[1, 0],
54         ),
55     denominators=[16],
56     extraCountsPerDivision=(0, 0, 1, 2, 0, 1),
57     )
58 )

```

## D.2.5 ERSILIA.MATERIALS.GUITAR\_POINTILLIST\_HARMONICS\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import scoretools
5 from ersilia.materials import abbreviations
6
7
8 guitar_pointillist_harmonics_musicSpecifier = consort.MusicSpecifier(
9     attachmentHandler=consort.AttachmentHandler(
10        dynamicExpressions=consort.DynamicExpression(
11            dynamicTokens='p mp',
12            ),
13        ),
14        color=None,
15        labels=[],
16        pitchHandler=consort.PitchClassPitchHandler(
17            forbidRepetitions=True,
18            leapConstraint=6,
19            logicalTieExpressions=[
20                consort.HarmonicExpression('P4'),
21                consort.HarmonicExpression('P5'),
22                consort.HarmonicExpression('P8'),
23                consort.HarmonicExpression('P5'),
24                ],
25        pitchSpecifier=abbreviations.agitato_pitchSpecifier,

```

```

26     registerSpecifier=consort.RegisterSpecifier(
27         base_pitch='E2',
28         phrase_inflections=consort.RegisterInflection
29             .zigzag(6)
30             .reverse()
31             .align(),
32         segment_inflections=consort.RegisterInflection
33             .descending(width=12)
34             .align()
35         ),
36     ),
37     rhythm_maker=consort.CompositeRhythmMaker(
38         default=rhythmmakertools.EvenDivisionRhythmMaker(
39             burnishSpecifier=rhythmmakertools.BurnishSpecifier(
40                 left_classes=[scoretools.Rest],
41                 left_counts=[1, 0],
42                 right_classes=[scoretools.Rest],
43                 right_counts=[1],
44             ),
45             denominators=[8, 8, 16],
46             extra_counts_per_division=[0, 0, 1, 0, 1, 2],
47         ),
48         last=rhythmmakertools.IncisedRhythmMaker(
49             inciseSpecifier=rhythmmakertools.InciseSpecifier(
50                 fill_with_notes=False,
51                 prefix_counts=[1],
52                 prefix_talea=[1],
53                 talea_denominator=8,
54             ),
55         ),
56     ),
57 )

```

#### D.2.6 ERSILIA.MATERIALS.GUITAR\_STRUMMED\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import lilypondnametools
5 from abjad.tools import markuptools
6 from abjad.tools import rhythmmakertools
7 from abjad.tools import scoretools
8 from abjad.tools import selectortools
9 from ersilia.materials import abbreviations
10
11
12 guitar_strummed_musicSpecifier = consort.MusicSpecifier(
13     attachmentHandler=consort.AttachmentHandler(
14         damped=consort.AttachmentExpression(
15             attachments=consort.LeafExpression(
16                 leaf=scoretools.Note("f'4"),
17                 attachments=[
18                     lilypondnametools.LilyPondGrobOverride(
19                         grob_name='NoteHead',

```

```

20         is_once=True,
21         property_path='transparent',
22         value=True,
23     ),
24     markuptools.Markup.musicglyph('scripts.coda'),
25     indicatortools.Articulation('accent'),
26     indicatortools.Dynamic('sfz'),
27 ],
28 ),
29     is_destructive=True,
30     selector=selectortools.Selector()
31     .by_logical_tie(pitched=True)
32     .by_contiguity()
33     .by_length('>', 1)
34     .by_leaves()
35     [-1]
36 ),
37 dynamic_expressions=consort.DynamicExpression(
38     dynamic_tokens='p ppp p ppp mf p',
39     only_first=True,
40 ),
41 laissez_vibrer=abbreviations.laissez_vibrer,
42 ),
43 pitch_handler=consort.AbsolutePitchHandler(
44     logical_tie_expressions=[
45         consortium.ChordExpression(
46             chord_expr=_,
47             arpeggio_direction=Center,
48             ) for _ in abbreviations.guitar_chords
49         ],
50     ),
51 rhythm_maker=rhythmmakertools.IncisedRhythmMaker(
52     inciseSpecifier=rhythmmakertools.InciseSpecifier(
53         fill_with_notes=False,
54         prefix_counts=[1, 1, 1, 2, 1, 2, 3],
55         prefix_talea=[1],
56         talea_denominator=16,
57         ),
58     ),
59 )

```

#### D.2.7 ERSILIA.MATERIALS.GUITAR\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 guitar_tremolo_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(

```

```

12     accents=consort.AttachmentExpression(
13         attachments=indicatortools.Articulation('accent'),
14         selector=selectortools.select_pitched_runs()
15             .by_counts([3], cyclic=True)
16             [1],
17         ),
18     dynamic_expressions=consort.DynamicExpression(
19         division_period=2,
20         dynamic_tokens='pp mf p p mf mf pp',
21         start_dynamic_tokens='fp o',
22         stop_dynamic_tokens='o f',
23     ),
24     stem_tremolo_spinner=consort.AttachmentExpression(
25         attachments=spannertools.StemTremoloSpanner(),
26         selector=selectortools.select_pitched_runs(),
27     ),
28     ),
29     color='red',
30     pitch_handler=consort.AbsolutePitchHandler(
31         logical_tie_expressions=[
32             consortium.ChordExpression(
33                 chord_expr=_,
34                 ) for _ in abbreviations.guitar_chords
35             ],
36     pitchSpecifier=consort.PitchSpecifier(
37         pitch_segments=(
38             'D3',
39             'F3',
40             'G2',
41             ),
42             ratio=(1, 1, 1),
43             ),
44     ),
45     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
46         tieSpecifier=rhythmmakertools.TieSpecifier(
47             tie_across_divisions=True,
48             ),
49         ),
50     )

```

#### D.2.8 ERSILIA.MATERIALS.GUITAR\_UNDULATION\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 guitar_undulation_tremolo_music_specifier = consortium.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         accents=consort.AttachmentExpression(

```

```

13     attachments=indicatortools.Articulation('accent'),
14     selector=selectortools.Selector()
15         .by_logical_tie(pitched=True)
16         .by_duration('==', (1, 8), preprolated=True)
17         [0]
18         .with_next_leaf()
19     ),
20     dynamic_expressions=consort.DynamicExpression(
21         dynamic_tokens='p mp pp',
22         start_dynamic_tokens='o',
23         stop_dynamic_tokens='o',
24         division_period=2,
25     ),
26     stem_tremolo_spinner=consort.AttachmentExpression(
27         attachments=spannertools.StemTremoloSpinner(),
28         selector=selectortools.select_pitched_runs(),
29     ),
30     ),
31     color='red',
32     labels=[],
33     pitch_handler=consort.AbsolutePitchHandler(
34         forbid_repetitions=True,
35         logical_tie_expressions=(
36             consort.ChordExpression(chord_expr=[0, 7, 14, 15]),
37         ),
38         pitchSpecifier=abbreviations.agitato_pitchSpecifier,
39         pitchOperationSpecifier=abbreviations.pitch_operationSpecifier,
40     ),
41     rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
42         denominators=[8],
43         extra_counts_per_division=[0, 1],
44         output_masks=[
45             rhythmmakertools.SustainMask(
46                 indices=[2],
47                 period=3,
48             ),
49             rhythmmakertools.SustainMask(
50                 indices=[0, -1],
51             ),
52         ],
53         tieSpecifier=rhythmmakertools.TieSpecifier(
54             tie_across_divisions=True,
55         ),
56     ),
57 )

```

## D.2.9 ERSILIA.MATERIALS.PERCUSION\_BAMBOO\_WINDCHIMES\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 import ersilia
4 from abjad.tools import indicatortools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import selectortools

```

```

7 from abjad.tools import spannertools
8 from ersilia.materials import abbreviations
9
10
11 percussion_bamboo_windchimes_musicSpecifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         accents=consort.AttachmentExpression(
14             attachments=[
15                 [
16                     indicatortools.Articulation('accent'),
17                     indicatortools.Dynamic('f'),
18                 ]
19             ],
20             selector=selectortools.Selector()
21             .by_logical_tie(pitched=True)
22             .by_duration('==', (1, 16), preprolated=True)
23             [0],
24         ),
25         piano=consort.AttachmentExpression(
26             attachments=indicatortools.Dynamic('pp'),
27             selector=selectortools.Selector()
28             .by_logical_tie(pitched=True)
29             .by_duration('>', (1, 16), preprolated=True)
30             [0]
31         ),
32         textSpanner=consort.AttachmentExpression(
33             attachments=abbreviations.make_text_spanner('windchimes'),
34             selector=selectortools.select_pitched_runs(),
35         ),
36         tremolo=consort.AttachmentExpression(
37             attachments=spannertools.StemTremoloSpanner(),
38             selector=selectortools.Selector()
39             .by_logical_tie(pitched=True)
40             .by_duration('>', (1, 16), preprolated=True)
41         ),
42     ),
43     color='yellow',
44     labels=['bamboo windchimes'],
45     pitch_handler=consort.AbsolutePitchHandler(
46         pitchSpecifier=ersilia.Percussion.BAMBOO_WIND_CHIMES,
47         pitches_are_nonsemantic=True,
48     ),
49     rhythm_maker=consort.CompositeRhythmMaker(
50         default=rhythmmakertools.NoteRhythmMaker(
51             tieSpecifier=rhythmmakertools.TieSpecifier(
52                 tie_across_divisions=True,
53             ),
54         ),
55         first=rhythmmakertools.IncisedRhythmMaker(
56             inciseSpecifier=rhythmmakertools.InciseSpecifier(
57                 fill_with_notes=False,
58                 prefix_counts=[1],
59                 prefix_talea=[1],
60                 talea_denominator=16,

```

```
61         ),
62     ),
63   ),
64 )
```

#### D.2.10 ERSILIA.MATERIALS.PERCUSION\_CROTALES\_FLASH\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import selectortools
5 from ersilia.materials import abbreviations
6
7
8 percussion_crotale_flash_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10        clef_spanner=consort.ClefSpanner('treble^15'),
11        dynamic_expressions=consort.DynamicExpression(
12            only_first=True,
13            start_dynamic_tokens='f p mp',
14        ),
15        text_spanner=consort.AttachmentExpression(
16            attachments=abbreviations.make_text_spanner('crotale'),
17            selector=selectortools.select_pitched_runs(),
18        ),
19    ),
20    color=None,
21    labels=[],
22    pitch_handler=consort.PitchClassPitchHandler(
23        forbid_repetitions=True,
24        pitch_specifier=abbreviations.agitato_pitch_specifier,
25        register_specifier=consort.RegisterSpecifier(
26            base_pitch="c'"),
27            segment_inflections=consort.RegisterInflection
28                .zigzag(6)
29                .align()
30        ),
31    ),
32    rhythm_maker=rhythmmakertools.IncisedRhythmMaker(
33        inciseSpecifier=rhythmmakertools.InciseSpecifier(
34            fill_with_notes=False,
35            prefix_counts=[4, 3],
36            prefix_talea=[1],
37            talea_denominator=16,
38        ),
39    ),
40 )
```

#### D.2.11 ERSILIA.MATERIALS.PERCUSION\_CROTALES INTERRUPTION\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
```

```

5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 percussion_crotale INTERRUPTION_musicspecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         accents=consort.AttachmentExpression(
13             attachments=[
14                 [
15                     indicatortools.Articulation('accent'),
16                     indicatortools.Dynamic('fff'),
17                 ]
18             ],
19             selector=selectortools.Selector()
20             .by_logical_tie(pitched=True)
21             .by_duration('==', (1, 16), preprolated=True)
22             [0],
23         ),
24         clef_spanner=consort.ClefSpanner('treble^15'),
25         shimmer=consort.AttachmentExpression(
26             attachments=[
27                 [
28                     indicatortools.Articulation('accent'),
29                     indicatortools.Dynamic('fp'),
30                 ],
31             ],
32             selector=selectortools.Selector()
33             .by_logical_tie(pitched=True)
34             .by_duration('>', (1, 16), preprolated=True)
35             .by_leaves()
36             .by_length('==', 1)
37             [0]
38         ),
39         swell=consort.AttachmentExpression(
40             attachments=spannertools.Hairpin('niente < f'),
41             selector=selectortools.Selector()
42             .by_logical_tie(pitched=True)
43             .by_duration('>', (1, 16), preprolated=True)
44             .by_leaves()
45             .by_length('>', 1)
46         ),
47         text_spanner=consort.AttachmentExpression(
48             attachments=abbreviations.make_text_spanner('crotale'),
49             selector=selectortools.select_pitched_runs(),
50         ),
51         tremolo=consort.AttachmentExpression(
52             attachments=spannertools.StemTremoloSpanner(),
53             selector=selectortools.Selector()
54             .by_logical_tie(pitched=True)
55             .by_duration('>', (1, 16), preprolated=True)
56         ),
57     ),
58     color='yellow',

```

```

59     labels=[],
60     pitch_handler=consort.PitchClassPitchHandler(
61         forbid_repetitions=True,
62         pitchSpecifier=abbreviations.agitato_pitchSpecifier,
63         registerSpecifier=consort.RegisterSpecifier(
64             base_pitch="c''",
65             segment_inflections=consort.RegisterInflection
66             .zigzag(6)
67             .align()
68         ),
69     ),
70     rhythm_maker=consort.CompositeRhythmMaker(
71         default=rhythmmakertools.NoteRhythmMaker(
72             tieSpecifier=rhythmmakertools.TieSpecifier(
73                 tie_across_divisions=True,
74             ),
75         ),
76         first=rhythmmakertools.IncisedRhythmMaker(
77             inciseSpecifier=rhythmmakertools.InciseSpecifier(
78                 fill_with_notes=False,
79                 prefix_counts=[1],
80                 prefix_talea=[1],
81                 talea_denominator=16,
82             ),
83         ),
84     ),
85 )

```

### D.2.12 ERSILIA.MATERIALS.PERCUSION\_LOW\_PEDAL\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 import ersilia
4 from abjad.tools import indicatortools
5 from abjad.tools import pitchtools
6 from abjad.tools import rhythmmakertools
7 from abjad.tools import selectortools
8 from abjad.tools import spannertools
9 from ersilia.materials import abbreviations
10
11
12 percussion_low_pedal_musicSpecifier = consort.MusicSpecifier(
13     attachmentHandler=consort.AttachmentHandler(
14         accents=consort.AttachmentExpression(
15             attachments=indicatortools.Articulation('accent'),
16             selector=selectortools.Selector()
17             .by_logical_tie()
18             .get_slice(start=1, apply_to_each=False)
19             [0]
20         ),
21         bass_drum_indication=consort.AttachmentExpression(
22             attachments=consort.AttachmentExpression(
23                 attachments=abbreviations.make_text_spanner('bass drum'),
24                 selector=selectortools.select_pitched_runs(),

```

```

25         ),
26         selector=selectortools.Selector()
27             .by_logical_tie()
28             .by_pitch(pitches=ersilia.Percussion.BASS_DRUM)
29             .by_contiguity()
30             .by_leaves()
31         ),
32     tam_tam_indication=consort.AttachmentExpression(
33         attachments=consort.AttachmentExpression(
34             attachments=abbreviations.make_text_spanner('tam'),
35             selector=selectortools.select_pitched_runs(),
36         ),
37         selector=selectortools.Selector()
38             .by_logical_tie()
39             .by_pitch(pitches=ersilia.Percussion.TAM_TAM)
40             .by_contiguity()
41             .by_leaves()
42         ),
43     dynamic_expressions=consort.DynamicExpression(
44         division_period=2,
45         dynamic_tokens='p ppp p ppp mf',
46         start_dynamic_tokens='o',
47         stop_dynamic_tokens='o',
48     ),
49     stem_tremolo_spinner=consort.AttachmentExpression(
50         attachments=spannertools.StemTremoloSpanner(),
51         selector=selectortools.select_pitched_runs(),
52     ),
53     ),
54     color='red',
55     labels=[],
56     minimum_phrase_duration=(3, 2),
57     pitch_handler=consort.AbsolutePitchHandler(
58         pitch_application_rate='phrase',
59         pitchSpecifier=pitchtools.PitchSegment([
60             ersilia.Percussion.BASS_DRUM,
61             ersilia.Percussion.TAM_TAM,
62         ]),
63         pitches_are_nonsemantic=True,
64     ),
65     rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
66         denominators=[8],
67         output_masks=[
68             rhythmmakertools.SustainMask(
69                 indices=[0, 1],
70                 period=3,
71             ),
72             rhythmmakertools.SustainMask(
73                 indices=[0, -1],
74             ),
75         ],
76         tieSpecifier=rhythmmakertools.TieSpecifier(
77             tie_across_divisions=True,
78         ),

```

```

79     ),
80 )

```

**D.2.13 ERSILIA.MATERIALS.PERCUSION\_MARIMBA\_AGITATO\_MUSIC\_SPECIFIER**

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import selectortools
5 from abjad.tools import spannertools
6 from ersilia.materials import abbreviations
7 import consort
8
9
10 percussion_marimba_agitato_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         accents=consort.AttachmentExpression(
13             attachments=indicatortools.Articulation('accent'),
14             selector=selectortools.select_pitched_runs()[0],
15         ),
16         chords=consort.AttachmentExpression(
17             attachments=[
18                 consort.ChordExpression(chord_expr=[0, 3]),
19                 consort.ChordExpression(chord_expr=[0, 5]),
20             ],
21             is_destructive=True,
22             selector=selectortools.Selector()
23             .by_logical_tie(pitched=True)
24             .by_duration('==', (1, 16), preprolated=True)
25             .by_pattern(
26                 rhythmmakertools.BooleanPattern(
27                     indices=[0, 3],
28                     period=7,
29                 ),
30             ),
31         ),
32         clef_spanner=consort.ClefSpannerExpression(),
33         dynamic_expressions=consort.DynamicExpression(
34             division_period=2,
35             dynamic_tokens='mf mp fff',
36             start_dynamic_tokens='f',
37             stop_dynamic_tokens='mf',
38         ),
39         staccato=consort.AttachmentExpression(
40             attachments=indicatortools.Articulation('staccato'),
41             selector=selectortools.Selector()
42             .by_logical_tie(pitched=True)
43             .by_duration('==', (1, 16), preprolated=True)
44             [0]
45         ),
46         staff_lines_spanner=spannertools.StaffLinesSpanner([-4, -2, 0, 2, 4]),
47         text_spanner=consort.AttachmentExpression(
48             attachments=abbreviations.make_text_spanner('marimba'),
49             selector=selectortools.select_pitched_runs(),

```

```

50 ),
51 tremolo_chords=consort.AttachmentExpression(
52     attachments=[
53         [
54             spannertools.StemTremoloSpanner(),
55             consort.ChordExpression(chord_expr=[0, 3]),
56         ],
57     ],
58     is_destructive=True,
59     selector=selectortools.Selector()
60         .by_logical_tie(pitched=True)
61         .by_duration('>', (1, 16), preprolated=True)
62     ),
63 ),
64 color='magenta',
65 labels=[],
66 pitch_handler=consort.PitchClassPitchHandler(
67     forbid_repetitions=True,
68     leap_constraint=9,
69     pitch_specifier=abbreviations.agitato_pitch_specifier,
70     registerSpecifier=consort.RegisterSpecifier(
71         base_pitch='F2',
72         phrase_inflections=consort.RegisterInflection.zigzag(6)
73             .reverse()
74             .align(),
75         segment_inflections=consort.RegisterInflection
76             .descending(width=12)
77             .align()
78     ),
79     register_spread=6,
80 ),
81 rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
82     extra_counts_per_division=[0, 0, 1, 2, 0, 1],
83     output_masks=[
84         rhythmmakertools.SustainMask(
85             indices=[1],
86             period=3,
87         ),
88     ],
89     talea=rhythmmakertools.Talea(
90         counts=[
91             1, -1,
92             1, 1, -1,
93             1, 1, 1, -1,
94             1, 1, -1,
95             1, 1, 1, -2,
96         ],
97         denominator=16,
98     ),
99 )
100 )

```

#### D.2.14 ERSILIA.MATERIALS.PERCUSION\_MARIMBA\_OSTINATO\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 percussion_marimba_ostinato_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         clef_spanner=consort.ClefSpannerExpression(),
13         deadstroke=consort.AttachmentExpression(
14             attachments=indicatortools.Articulation('stopped'),
15             selector=selectortools.Selector()
16                 .by_logical_tie(pitched=True)
17                 .by_duration('==', (1, 16), preprolated=True)
18                 .by_contiguity()
19                 .by_length('==', 1)
20                 .by_leaves()
21             [0]
22         ),
23         dynamic_expressions=consort.DynamicExpression(
24             dynamic_tokens='p',
25         ),
26         slur=consort.AttachmentExpression(
27             attachments=spannertools.Slur(),
28             selector=selectortools.Selector()
29                 .by_logical_tie(pitched=True)
30                 .by_duration('==', (1, 16), preprolated=True)
31                 .by_contiguity()
32                 .by_length('>', 1)
33                 .by_leaves()
34         ),
35         staccati=consort.AttachmentExpression(
36             attachments=indicatortools.Articulation('staccato'),
37             selector=selectortools.Selector()
38                 .by_logical_tie(pitched=True)
39                 .by_duration('==', (1, 16), preprolated=True)
40                 .by_contiguity()
41                 .by_length('>', 1)
42                 .by_leaves()
43             [-1]
44         ),
45         staff_lines_spacer=spannertools.StaffLinesSpacer([-4, -2, 0, 2, 4]),
46         text_spacer=consort.AttachmentExpression(
47             attachments=abbreviations.make_text_spacer('marimba'),
48             selector=selectortools.select_pitched_runs(),
49         ),
50     ),
51     color='darkyellow',
52     pitch_handler=consort.AbsolutePitchHandler(
```

```

53     forbid_repetitions=True,
54     logical_tie_expressions=[
55         consort.ChordExpression(chord_expr=[0, 5]),
56         ],
57     pitchSpecifier="d' f",
58     ),
59     rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
60         extra_counts_per_division=[0, 0, 1, 2, 0, 1],
61         talea=rhythmmakertools.Talea(
62             counts=[1, 1, -3],
63             denominator=16,
64             )
65         ),
66     )

```

## D.2.15 ERSILIA.MATERIALS.PERCUSION\_MARIMBA\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import pitchtools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 percussion_marimba_tremolo_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         clef_spanner=consort.ClefSpannerExpression(),
13         dynamic_expressions=consort.DynamicExpression(
14             division_period=2,
15             dynamic_tokens='p ppp',
16             start_dynamic_tokens='o',
17             stop_dynamic_tokens='o',
18             ),
19         staff_lines_spanner=spannertools.StaffLinesSpanner([-4, -2, 0, 2, 4]),
20         stem_tremolo_spanner=consort.AttachmentExpression(
21             attachments=spannertools.StemTremoloSpanner(),
22             selector=selectortools.select_pitched_runs(),
23             ),
24         text_spanner=consort.AttachmentExpression(
25             attachments=abbreviations.make_text_spanner('marimba'),
26             selector=selectortools.select_pitched_runs(),
27             ),
28         ),
29         color='red',
30         labels=[],
31         pitch_handler=consort.PitchClassPitchHandler(
32             leap_constraint=9,
33             logical_tie_expressions=(
34                 consort.ChordExpression(
35                     chord_expr=pitchtools.IntervalSegment([0, 3]),
36                     ),
37                 consort.ChordExpression(

```

```

38         chord_expr=pitchtools.IntervalSegment([0, 5]),
39         ),
40     consort.ChordExpression(
41         chord_expr=pitchtools.IntervalSegment([0, 3]),
42         ),
43     consort.ChordExpression(
44         chord_expr=pitchtools.IntervalSegment([0, 2]),
45         ),
46     ),
47     pitchSpecifier=consort.PitchSpecifier(
48         pitchSegments=[
49             "d d f d d f g",
50             "ef gf gf df b df",
51             ],
52         ),
53     registerSpecifier=consort.RegisterSpecifier(
54         base_pitch='F2',
55         segment_inflections=consort.RegisterInflection
56             .zigzag(12)
57             .reverse()
58             .align(),
59         ),
60     registerSpread=3,
61     ),
62     rhythmMaker=rhythmmakertools.EvenDivisionRhythmMaker(
63         denominators=[16],
64         outputMasks=[
65             rhythmmakertools.SustainMask(
66                 indices=[2],
67                 invert=True,
68                 period=3,
69                 ),
70             ],
71             tieSpecifier=rhythmmakertools.TieSpecifier(
72                 tie_across_divisions=True,
73                 ),
74             ),
75         )

```

#### D.2.16 ERSILIA.MATERIALS.PERCUSION\_SNARE INTERRUPTION\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 import ersilia
4 from abjad.tools import indicatortools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8 from ersilia.materials import abbreviations
9
10
11 percussion_snare_interruption_music_specifier = consort.MusicSpecifier(
12     attachmentHandler=consort.AttachmentHandler(
13         accents=consort.AttachmentExpression(

```

```

14     attachments=[
15         [
16             indicatortools.Articulation('accent'),
17             indicatortools.Dynamic('fff'),
18         ]
19     ],
20     selector=selectortools.Selector()
21         .by_logical_tie(pitched=True)
22         .by_duration('==', (1, 16), preprolated=True)
23         [0],
24     ),
25     shimmer=consort.AttachmentExpression(
26         attachments=[
27             [
28                 indicatortools.Articulation('accent'),
29                 indicatortools.Dynamic('fp'),
30             ],
31         ],
32         selector=selectortools.Selector()
33             .by_logical_tie(pitched=True)
34             .by_duration('>', (1, 16), preprolated=True)
35             .by_leaves()
36             .by_length('==', 1)
37         [0]
38     ),
39     swell=consort.AttachmentExpression(
40         attachments=spannertools.Hairpin('niente < f'),
41         selector=selectortools.Selector()
42             .by_logical_tie(pitched=True)
43             .by_duration('>', (1, 16), preprolated=True)
44             .by_leaves()
45             .by_length('>', 1)
46     ),
47     text_spinner=consort.AttachmentExpression(
48         attachments=abbreviations.make_text_spinner('snare'),
49         selector=selectortools.select_pitched_runs(),
50     ),
51     tremolo=consort.AttachmentExpression(
52         attachments=spannertools.StemTremoloSpanner(),
53         selector=selectortools.Selector()
54             .by_logical_tie(pitched=True)
55             .by_duration('>', (1, 16), preprolated=True)
56     ),
57     ),
58     color='yellow',
59     labels=[],
60     pitch_handler=consort.AbsolutePitchHandler(
61         pitchSpecifier=ersilia.Percussion.SNARE_DRUM,
62         pitches_are_nonsemantic=True,
63     ),
64     rhythm_maker=consort.CompositeRhythmMaker(
65         default=rhythmmakertools.NoteRhythmMaker(
66             tieSpecifier=rhythmmakertools.TieSpecifier(
67                 tie_across_divisions=True,

```

```

68         ),
69     ),
70     first=rhythmmakertools.IncisedRhythmMaker(
71         inciseSpecifier=rhythmmakertools.InciseSpecifier(
72             fillWithNotes=False,
73             prefixCounts=[1, 2],
74             prefixTalea=[1, -1],
75             taleaDenominator=16,
76         ),
77     ),
78 ),
79 )

```

### D.2.17 ERSILIA.MATERIALS.PERCUSION\_TEMPLE\_BLOCK\_FANFARE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 import ersilia
4 from abjad.tools import pitchtools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import selectortools
7 from abjad.tools import indicatortools
8 from abjad.tools import spannertools
9 from ersilia.materials import abbreviations
10
11
12 percussion_temple_block_fanfare_music_specifier = consort.MusicSpecifier(
13     attachmentHandler=consort.AttachmentHandler(
14         accent=consort.AttachmentExpression(
15             attachments=indicatortools.Articulation('accent'),
16             selector=selectortools.Selector()
17                 .by_logical_tie(pitched=True)
18                 .by_duration('>', (1, 16), preprolated=True)
19                 [0]
20         ),
21         chords=consort.AttachmentExpression(
22             attachments=(
23                 consort.ChordExpression(
24                     chordExpr=pitchtools.PitchSegment([
25                         ersilia.Percussion.WOOD_BLOCK_5,
26                         ersilia.Percussion.WOOD_BLOCK_4,
27                     ]),
28                 ),
29                 None,
30                 consort.ChordExpression(
31                     chordExpr=pitchtools.PitchSegment([
32                         ersilia.Percussion.WOOD_BLOCK_4,
33                         ersilia.Percussion.WOOD_BLOCK_3,
34                     ]),
35                 ),
36                 consort.ChordExpression(
37                     chordExpr=pitchtools.PitchSegment([
38                         ersilia.Percussion.WOOD_BLOCK_3,
39                         ersilia.Percussion.WOOD_BLOCK_2,

```

```

40                 ],
41             ),
42             None,
43             consort.ChordExpression(
44                 chord_expr=pitchtools.PitchSegment([
45                     ersilia.Percussion.WOOD_BLOCK_2,
46                     ersilia.Percussion.WOOD_BLOCK_1,
47                 ]),
48             ),
49         ),
50         is_destructive=True,
51         selector=selectortools.Selector()
52             .by_logical_tie(pitched=True)
53             .by_duration('>', (1, 16), preprolated=True)
54         ),
55     dynamic_expression=consort.DynamicExpression(
56         division_period=2,
57         start_dynamic_tokens='p fp',
58         stop_dynamic_tokens='f',
59         unsustained=True,
60     ),
61     staccato=consort.AttachmentExpression(
62         attachments=indicatortools.Articulation('staccato'),
63         selector=selectortools.Selector()
64             .by_logical_tie(pitched=True)
65             .by_duration('<', (1, 16), preprolated=True)
66             [0],
67     ),
68     staff_lines_spanner=spannertools.StaffLinesSpanner([-4, -2, 0, 2, 4]),
69     text_spanner=consort.AttachmentExpression(
70         attachments=abbreviations.make_text_spanner('blocks'),
71         selector=selectortools.select_pitched_runs(),
72     ),
73     tremolo=consort.AttachmentExpression(
74         attachments=spannertools.StemTremoloSpanner(),
75         selector=selectortools.Selector()
76             .by_logical_tie(pitched=True)
77             .by_duration('>', (1, 16), preprolated=True)
78     ),
79     ),
80     color='magenta',
81     pitch_handler=consort.AbsolutePitchHandler(
82         #logical_tie_expressions=[
83         #    ],
84         pitchSpecifier=pitchtools.PitchSegment([
85             ersilia.Percussion.WOOD_BLOCK_5,
86             ersilia.Percussion.WOOD_BLOCK_4,
87             ersilia.Percussion.WOOD_BLOCK_3,
88             ersilia.Percussion.WOOD_BLOCK_2,
89             ersilia.Percussion.WOOD_BLOCK_1,
90             ersilia.Percussion.WOOD_BLOCK_4,
91             ersilia.Percussion.WOOD_BLOCK_3,
92             ersilia.Percussion.WOOD_BLOCK_2,
93             ersilia.Percussion.WOOD_BLOCK_3,

```

```

94         ersilia.Percussion.WOOD_BLOCK_2,
95         ersilia.Percussion.WOOD_BLOCK_2,
96     ]),
97     pitch_operationSpecifier=consort.PitchOperationSpecifier(
98         pitch_operations=[
99             None,
100            pitchtools.Retrogression(),
101            None,
102        ],
103    ),
104    pitches_are_nonsemantic=True,
105  ),
106 rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
107     extra_counts_per_division=[0, 1, 2],
108     talea=rhythmmakertools.Talea(
109         counts=[
110             1, 1, -1,
111             1, 1, -1,
112             1, 1, -2,
113             1, 1, 1, 1, 1, 1, -1,
114             1, 1, -2,
115             1, 1, 1, -2,
116             1, 1, -1,
117             1, 1, 1, 1, 1, 1, -1,
118             1, 1, 1, -1,
119             1, 1, 1, 1, -2,
120         ],
121         denominator=16,
122     ),
123     output_masks=[
124         rhythmmakertools.SustainMask(
125             indices=[1],
126             period=2,
127         ),
128     ],
129   )
130 )

```

## D.2.18 ERSILIA.MATERIALS.PERCUSION\_TOM\_FANFARE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 import ersilia
4 from abjad.tools import indicatortools
5 from abjad.tools import pitchtools
6 from abjad.tools import rhythmmakertools
7 from abjad.tools import selectortools
8 from abjad.tools import spannertools
9 from ersilia.materials import abbreviations
10
11
12 percussion_tom_fanfare_music_specifier = consort.MusicSpecifier(
13     attachment_handler=consort.AttachmentHandler(
14         accent=consort.AttachmentExpression(

```

```

15     attachments=indicatortools.Articulation('accent'),
16     selector=selectortools.Selector()
17         .by_logical_tie(pitched=True)
18         .by_duration('>', (1, 16), preprolated=True)
19         [0]
20     ),
21     dynamic_expression=consort.DynamicExpression(
22         start_dynamic_tokens='p fp',
23         stop_dynamic_tokens='f',
24         unsustained=True,
25     ),
26     staccato=consort.AttachmentExpression(
27         attachments=indicatortools.Articulation('staccato'),
28         selector=selectortools.Selector()
29             .by_logical_tie(pitched=True)
30             .by_duration('<=', (1, 16), preprolated=True)
31             [0],
32     ),
33     staff_lines_spinner=spannertools.StaffLinesSpinner([-4, -2, 0, 2, 4]),
34     text_spinner=consort.AttachmentExpression(
35         attachments=abbreviations.make_text_spinner('toms'),
36         selector=selectortools.select_pitched_runs(),
37     ),
38     tremolo=consort.AttachmentExpression(
39         attachments=spannertools.StemTremoloSpinner(),
40         selector=selectortools.Selector()
41             .by_logical_tie(pitched=True)
42             .by_duration('>', (1, 16), preprolated=True)
43     ),
44     ),
45     color='magenta',
46     pitch_handler=consort.AbsolutePitchHandler(
47         #logical_tie_expressions=[# ],
48         pitchSpecifier=pitchtools.PitchSegment([
49             ersilia.Percussion.TOM_4,
50             ersilia.Percussion.TOM_3,
51             ersilia.Percussion.TOM_2,
52             ersilia.Percussion.TOM_1,
53             ersilia.Percussion.TOM_4,
54             ersilia.Percussion.TOM_3,
55             ersilia.Percussion.TOM_2,
56             ersilia.Percussion.TOM_3,
57             ersilia.Percussion.TOM_2,
58             ersilia.Percussion.TOM_2,
59             ]),
60         pitchOperationSpecifier=consort.PitchOperationSpecifier(
61             pitchOperations=[
62                 None,
63                 pitchtools.Retrogression(),
64                 None,
65                 ],
66             ),
67         ),
68     pitches_are_nonsemantic=True,

```

```

69     ),
70     rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
71         extra_counts_per_division=[0, 1, 2],
72         talea=rhythmmakertools.Talea(
73             counts=[
74                 1, 1, -1,
75                 1, 1, -1,
76                 1, 1, -2,
77                 1, 1, -2,
78                 1, 1, 1, 1, 1, 1, 1, 1, 1,
79                 1, 1, 1, -2,
80                 1, 1, -1,
81                 1, 1, 1, -1,
82                 1, 1, 1, 1, -2,
83             ],
84             denominator=16,
85         ),
86         output_masks=[
87             rhythmmakertools.SustainMask(
88                 indices=[1],
89                 period=2,
90             ),
91         ],
92     )
93 )

```

#### D.2.19 ERSILIA.MATERIALS.PERMITTED\_TIME\_SIGNATURES

```

1 # -*- encoding: utf-8 -*-
2 from abjad import *
3
4
5 permitted_time_signatures = indicatortools.TimeSignatureInventory([
6     #(2, 4),
7     (4, 8),
8     (5, 8),
9     (6, 8),
10    #(3, 4),
11    (7, 8),
12    (4, 4),
13    #(9, 8),
14    #(5, 4),
15    #(6, 4),
16 ])

```

#### D.2.20 ERSILIA.MATERIALS.PIANO\_AGITATO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 from abjad.tools import indicatortools
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import scoretools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations

```

```

8 import consort
9
10
11 piano_agitato_musicSpecifier = consort.MusicSpecifier(
12     attachmentHandler=consort.AttachmentHandler(
13         accents=consort.AttachmentExpression(
14             attachments=indicatortools.Articulation('accent'),
15             selector=selectortools.Selector()
16             .by_logical_tie(pitched=True)
17             .by_duration('==', (1, 8), preprolated=True)
18             [0],
19         ),
20         dynamic_expressions=consort.DynamicExpression(
21             divisionPeriod=2,
22             dynamicTokens='mf mp fff',
23             startDynamicTokens='f',
24             stopDynamicTokens='mf',
25         ),
26         mordent=consort.AttachmentExpression(
27             attachments=indicatortools.Articulation('mordent'),
28             selector=selectortools.Selector()
29             .by_logical_tie(pitched=True)
30             .by_duration('>=', (1, 8), preprolated=True)
31             .by_class(scoretools.Note)
32             [0],
33         ),
34         staccati=consort.AttachmentExpression(
35             attachments=indicatortools.Articulation('staccato'),
36             selector=selectortools.Selector()
37             .by_logical_tie(pitched=True)
38             .by_duration('==', (1, 16), preprolated=True)
39             [0],
40         ),
41         tremoli=consort.AttachmentExpression(
42             attachments=spannertools.StemTremoloSpanner(),
43             selector=selectortools.Selector()
44             .by_logical_tie(pitched=True)
45             .by_duration('>=', (1, 8), preprolated=True)
46             .by_class(scoretools.Chord)
47             [0],
48         ),
49     ),
50     color='magenta',
51     labels=[],
52     pitchHandler=consort.PitchClassPitchHandler(
53         forbidRepetitions=True,
54         leapConstraint=12,
55         logicalTieExpressions=[
56             consortium.ChordExpression([-6, -3, 3, 8]),
57             None,
58             consortium.ChordExpression([0, 3]),
59             None,
60             None,
61             consortium.ChordExpression([0, 3]),

```

```

62     consort.ChordExpression([-1, 2]),
63     None,
64     consort.KeyClusterExpression(
65         include_black_keys=False,
66     ),
67     consort.ChordExpression([0, 3]),
68     None,
69     None,
70 ],
71 pitchSpecifier=abbreviations.agitato_pitchSpecifier,
72 registerSpecifier=consort.RegisterSpecifier(
73     basePitch='G3',
74     phraseInflections=consort.RegisterInflection.zigzag(6)
75         .reverse().align(),
76     segmentInflections=consort.RegisterInflection.descending(
77         width=12).align()
78 ),
79 registerSpread=6,
80 ),
81 rhythmMaker=rhythmmakertools.TaleaRhythmMaker(
82     burnishSpecifier=rhythmmakertools.BurnishSpecifier(
83         leftClasses=[scoretools.Rest],
84         leftCounts=[1, 1, 0],
85         rightClasses=[scoretools.Rest],
86         rightCounts=[1],
87     ),
88     extraCountsPerDivision=[0, 0, 1],
89     outputMasks=[
90         rhythmmakertools.SustainMask(
91             indices=[0],
92             period=3,
93         ),
94     ],
95     talea=rhythmmakertools.Talea(
96         counts=[1, 1, 1, 1, 2],
97         denominator=16,
98     ),
99 ),
100 )

```

### D.2.21 ERSILIA.MATERIALS.PIANO\_ARM\_CLUSTER\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6
7
8 piano_arm_cluster_music_specifier = consort.MusicSpecifier(
9     attachmentHandler=consort.AttachmentHandler(
10         accents=consort.AttachmentExpression(
11             attachments=indicatortools.Articulation('accent'),
12             selector=selectortools.Selector())

```

```

13     .by_logical_tie(pitched=True)[0],
14 ),
15 dynamic_expressions=consort.AttachmentExpression(
16     attachments=indicatortools.Dynamic('fff'),
17     selector=selectortools.Selector()
18     .by_logical_tie(pitched=True)
19     [0]
20 ),
21 ),
22 color='yellow',
23 labels=['piano arm cluster'],
24 pitch_handler=consort.AbsolutePitchHandler(
25     logical_tie_expressions=[
26         consortium.KeyClusterExpression(
27             staff_space_width=19,
28         ),
29     ]
30 ),
31 rhythm_maker=rhythmmakertools.IncisedRhythmMaker(
32     inciseSpecifier=rhythmmakertools.InciseSpecifier(
33         fill_with_notes=False,
34         prefix_counts=[1],
35         prefix_talea=[1],
36         talea_denominator=8,
37     ),
38 ),
39 )

```

## D.2.22 ERSILIA.MATERIALS.PIANO\_GLISSANDO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import spannertools
5 from ersilia.materials import abbreviations
6
7
8 piano_glissando_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10         dynamic_expressions=consort.DynamicExpression(
11             dynamic_tokens='p',
12             only_first=True,
13         ),
14         glissando=spannertools.Glissando(),
15         keys_spanner=(
16             abbreviations.chromatic_keys_spanner,
17             abbreviations.white_keys_spanner,
18             abbreviations.white_keys_spanner,
19         ),
20     ),
21     color=None,
22     labels=[],
23     pitch_handler=consort.AbsolutePitchHandler(
24         forbid_repetitions=True,

```

```

25     pitchSpecifier="c' c'' f' f'' c'' c' f' c'' f'' c''' c'' f'' f''",
26   ),
27   rhythm_maker=consort.CompositeRhythmMaker(
28     last=rhythmmakertools.IncisedRhythmMaker(
29       inciseSpecifier=rhythmmakertools.InciseSpecifier(
30         prefixCounts=[0],
31         suffixTalea=[1],
32         suffixCounts=[1],
33         taleaDenominator=16,
34       ),
35       durationSpellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
36         forbiddenWrittenDuration=(1, 4),
37         forbidMeterRewriting=True,
38         spellMetrically='unassignable',
39       ),
40       tieSpecifier=rhythmmakertools.TieSpecifier(
41         stripTies=True,
42       ),
43     ),
44     default=rhythmmakertools.EvenDivisionRhythmMaker(
45       denominators=(4,),
46       durationSpellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
47         forbiddenWrittenDuration=(1, 4),
48         forbidMeterRewriting=True,
49         spellMetrically='unassignable',
50       ),
51     ),
52   ),
53 )

```

### D.2.23 ERSILIA.MATERIALS.PIANO\_PALM\_CLUSTER\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import scoretools
4 from abjad.tools import rhythmmakertools
5
6
7 piano_palm_cluster_music_specifier = consortium.MusicSpecifier(
8   attachmentHandler=consort.AttachmentHandler(
9     dynamicExpressions=consort.DynamicExpression(
10       dynamicTokens='p mf mp pp f',
11       divisionPeriod=2,
12     ),
13     octavation=consort.OctavationExpression(),
14   ),
15   labels='pedaled',
16   pitchHandler=consort.PitchClassPitchHandler(
17     forbidRepetitions=True,
18     logicalTieExpressions=(
19       consortium.KeyClusterExpression(
20         includeBlackKeys=False,
21       ),
22       consortium.KeyClusterExpression(

```

```

23         include_black_keys=False,
24         ),
25     consort.KeyClusterExpression(
26         staff_space_width=9,
27         ),
28     ),
29     pitchSpecifier="c e g",
30     registerSpecifier=consort.RegisterSpecifier(
31         division_inflections=consort.RegisterInflection.descending(),
32         phrase_inflections=consort.RegisterInflection.zigzag(),
33         segment_inflections=consort.RegisterInflection.descending(
34             width=24).align()
35         ),
36     ),
37 rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
38     burnishSpecifier=rhythmmakertools.BurnishSpecifier(
39         left_classes=[scoretools.Rest],
40         left_counts=[1],
41         right_classes=[scoretools.Rest],
42         right_counts=[1, 0, 0],
43         ),
44     denominators=[16, 16, 8, 16, 16, 8],
45     extra_counts_per_division=(0, 0, 1),
46     ),
47 )

```

#### D.2.24 ERSILIA.MATERIALS.PIANO\_PEDALS\_MUSIC\_SETTING

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4
5
6 piano_pedals_music_setting = consort.MusicSetting(
7     timespan_maker=consort.DependentTimespanMaker(
8         hysteresis=(1, 4),
9         include_inner_starts=True,
10        include_inner_stops=False,
11        inspect_music=True,
12        labels=(
13            'pedaled',
14            ),
15        voice_names=(
16            'Piano Upper Voice',
17            'Piano Lower Voice',
18            ),
19        ),
20    piano_pedals=consort.MusicSpecifier(
21        attachment_handler=consort.AttachmentHandler(
22            piano_pedal_spanner=consort.ComplexPianoPedalSpanner(
23                include_inner_leaves=True,
24                ),
25                ),
26    rhythm_maker=rhythmmakertools.SkipRhythmMaker(

```

```

27     duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
28         forbid_meter_rewriting=True,
29         ),
30     ),
31 )
32
33 )

```

## D.2.25 ERSILIA.MATERIALS.PIANO\_POINTILLIST\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import scoretools
6 from abjad.tools import selectortools
7 from ersilia.materials import abbreviations
8
9
10 piano_pointillist_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expressions=consort.DynamicExpression(
13             start_dynamic_tokens='ppp',
14             only_first=True,
15             ),
16         mordent=consort.AttachmentExpression(
17             attachments=indicatortools.Articulation('mordent'),
18             selector=selectortools.Selector()
19                 .by_class(scoretools.Note)
20                 .by_logical_tie()
21                 [0]
22             ),
23         tenuti=consort.AttachmentExpression(
24             attachments=indicatortools.Articulation('tenuto'),
25             selector=selectortools.Selector()
26                 .by_logical_tie(pitched=True)
27                 [0]
28             ),
29         ),
30         color='darkyellow',
31         labels=['pedaled'],
32         pitch_handler=consort.PitchClassPitchHandler(
33             logical_tie_expressions=(
34                 None,
35                 consortium.ChordExpression([-2, 3]),
36                 consortium.ChordExpression([0, 3]),
37                 None,
38                 consortium.ChordExpression([-4, 5]),
39             ),
40         pitchSpecifier=abbreviations.agitato_pitchSpecifier,
41         registerSpecifier=consort.RegisterSpecifier(
42             base_pitch='G3',
43             phrase_inflections=consort.RegisterInflection
44                 .zigzag(12)

```

```

45         .reverse()
46         .align(),
47         segment_inflections=consort.RegisterInflection
48             .descending(width=12)
49             .align()
50         ),
51     register_spread=6,
52     ),
53 rhythm_maker=consort.CompositeRhythmMaker(
54     default=rhythmmakertools.TaleaRhythmMaker(
55         extra_counts_per_division=[0, 0, 1],
56         talea=rhythmmakertools.Talea(
57             counts=[1, -2, 1, -3, 1, -4],
58             denominator=16,
59             ),
60         ),
61     last=rhythmmakertools.IncisedRhythmMaker(
62         inciseSpecifier=rhythmmakertools.InciseSpecifier(
63             fill_with_notes=False,
64             prefix_counts=[2],
65             prefix_talea=[-1, 1, 1],
66             talea_denominator=16,
67             ),
68         ),
69     ),
70 )

```

## D.2.26 ERSILIA.MATERIALS.PIANO\_STRING\_GLISSANDO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4 from abjad.tools import spannertools
5 from ersilia.materials import abbreviations
6
7
8 piano_string_glissando_music_specifier = consort.MusicSpecifier(
9     attachment_handler=consort.AttachmentHandler(
10        clef_spanner=consort.ClefSpanner('percussion'),
11        dynamic_expressions=consort.DynamicExpression(
12            dynamic_tokens='p',
13            only_first=True,
14            ),
15        glissando=spannertools.Glissando(),
16        staff_lines_spinner=spannertools.StaffLinesSpinner(
17            lines=[-4, 4],
18            overrides={
19                'note_head__style': 'cross',
20                }
21            ),
22        text_spinner=abbreviations.make_text_spinner('inside'),
23        ),
24    color=None,
25    labels=['pedaled'],

```

```

26     pitch_handler=consort.AbsolutePitchHandler(
27         forbid_repetitions=True,
28         pitchSpecifier="f c' g' c' f g' c' f c' g' f",
29     ),
30     rhythm_maker=consort.CompositeRhythmMaker(
31         last=rhythmmakertools.IncisedRhythmMaker(
32             inciseSpecifier=rhythmmakertools.InciseSpecifier(
33                 prefix_counts=[0],
34                 suffix_talea=[1],
35                 suffix_counts=[1],
36                 talea_denominator=16,
37             ),
38             duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
39                 forbidden_written_duration=(1, 4),
40                 forbid_meter_rewriting=True,
41                 spell_metrically='unassignable',
42             ),
43             tieSpecifier=rhythmmakertools.TieSpecifier(
44                 strip_ties=True,
45             ),
46         ),
47         default=rhythmmakertools.EvenDivisionRhythmMaker(
48             denominators=(4,),
49             duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
50                 forbidden_written_duration=(1, 4),
51                 forbid_meter_rewriting=True,
52                 spell_metrically='unassignable',
53             ),
54         ),
55     ),
56 )

```

#### D.2.27 ERSILIA.MATERIALS.PIANO\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import pitchtools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7
8
9 piano_tremolo_musicSpecifier = consortium.MusicSpecifier(
10     attachmentHandler=consort.AttachmentHandler(
11         dynamicExpressions=consort.DynamicExpression(
12             startDynamicTokens='fp',
13             dynamicTokens="p mf p p mf pp",
14             divisionPeriod=2,
15         ),
16         octavation=consort.OctavationExpression(),
17         stemTremoloSpanner=consort.AttachmentExpression(
18             attachments=spannertools.StemTremoloSpanner(),
19             selector=selectortools.select_pitched_runs(),
20         ),

```

```

21     ),
22     color='red',
23     labels=[
24         'pedaled',
25         'piano tremolo',
26     ],
27     pitch_handler=consort.AbsolutePitchHandler(
28         deviations=[0, -2, 0, 0, 2, 0, 3],
29         logical_tie_expressions=(
30             consortium.ChordExpression(
31                 chord_expr=pitchtools.IntervalSegment([-7, -3, 0, 5, 6, 12]),
32             ),
33             consortium.ChordExpression(
34                 chord_expr=pitchtools.IntervalSegment([-7, -3, 0, 1, 5, 12]),
35             ),
36         ),
37         pitchSpecifier=consort.PitchSpecifier(
38             pitch_segments=["d'", "f'", "c'"],
39         ),
40     ),
41     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
42         tieSpecifier=rhythmmakertools.TieSpecifier(
43             tie_across_divisions=True,
44         ),
45     ),
46 )

```

## D.2.28 ERSILIA.MATERIALS.PITCH\_PIPE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consortium
3 from abjad.tools import indicatortools
4 from abjad.tools import markuptools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import selectortools
7 from ersilia.materials import abbreviations
8
9
10 pitch_pipe_music_specifier = consortium.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         accents=consort.AttachmentExpression(
13             attachments=indicatortools.Articulation('accent'),
14             selector=selectortools.Selector()
15             .by_logical_tie()
16             .get_slice(start=1, apply_to_each=False)
17             [0]
18         ),
19         dynamic_expressions=consort.DynamicExpression(
20             division_period=2,
21             dynamic_tokens='p ppp mf',
22             start_dynamic_tokens='o fp',
23         ),
24         inhale_exhale=consort.AttachmentExpression(
25             attachments=[


```

```

26         markuptools.Markup('exhale', Up)
27             .italic().smaller().pad_around(0.5).box(),
28     markuptools.Markup('inhale', Up)
29             .italic().smaller().pad_around(0.5).box(),
30     markuptools.Markup('inhale', Up)
31             .italic().smaller().pad_around(0.5).box(),
32         ],
33     selector=selectortools.select_pitched_runs()[0],
34     ),
35     percussion_staff=abbreviations.percussion_staff
36     ),
37     color='blue',
38     labels=['pitch pipes'],
39     pitch_handler=consort.AbsolutePitchHandler(
40         pitches_are_nonsensematic=True,
41     ),
42     rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
43         denominators=[8],
44         output_masks=[
45             rhythmmakertools.SustainMask(
46                 indices=[0, 2],
47                 period=3,
48             ),
49             rhythmmakertools.SustainMask(
50                 indices=[0, -1],
51             ),
52         ],
53         tieSpecifier=rhythmmakertools.TieSpecifier(
54             tie_across_divisions=True,
55         ),
56     ),
57 )

```

## D.2.29 ERSILIA.MATERIALS.SAXOPHONE\_AGITATO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import scoretools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8 from ersilia.materials import abbreviations
9
10
11 saxophone_agitato_music_specifier = consortium.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         dynamic_expressions=consort.DynamicExpression(
14             division_period=2,
15             dynamic_tokens='mf mp fff',
16             start_dynamic_tokens='f',
17             stop_dynamic_tokens='mf',
18         ),
19         slur=consort.AttachmentExpression(

```

```

20     attachments=spanertools.Slur(),
21     selector=selectortools.Selector()
22         .by_logical_tie(pitched=True)
23         .by_duration('==', (1, 16), preprolated=True)
24         .by_contiguity()
25         .by_length('>', 1)
26         .by_pattern(
27             pattern=rhythmmakertools.BooleanPattern(
28                 indices=[0], period=2,
29             ),
30         )
31         .by_leaves()
32     ),
33 staccati=consort.AttachmentExpression(
34     attachments=indicatortools.Articulation('staccato'),
35     selector=selectortools.Selector()
36         .by_logical_tie(pitched=True)
37         .by_duration('==', (1, 16), preprolated=True)
38         .by_contiguity()
39         .by_length('>', 1)
40         .by_pattern(
41             pattern=rhythmmakertools.BooleanPattern(
42                 indices=[1], period=2,
43             ),
44         )
45         .by_leaves()
46     [1:]
47     ),
48 stopped=consort.AttachmentExpression(
49     attachments=indicatortools.Articulation('stopped'),
50     selector=selectortools.Selector()
51         .by_leaves()
52         .by_run(scoretools.Note)
53     [0]
54     ),
55 trill_spinner=consort.AttachmentExpression(
56     attachments=spanertools.ComplexTrillSpinner(
57         interval='+m3',
58     ),
59     selector=selectortools.Selector()
60         .by_logical_tie(pitched=True)
61         .by_duration('>', (1, 16), preprolated=True)
62         .by_contiguity()
63         .by_leaves()
64     ),
65     ),
66 color='magenta',
67 pitch_handler=consort.PitchClassPitchHandler(
68     forbid_repetitions=True,
69     leap_constraint=12,
70     pitchSpecifier=abbreviations.agitato_pitchSpecifier,
71     registerSpecifier=consort.RegisterSpecifier(
72         base_pitch='C2',
73         phrase_inflections=consort.RegisterInflection.zigzag(6)

```

```

74     .reverse().align(),
75     segment_inflections=consort.RegisterInflection.descending(
76         width=12).align()
77     ),
78     register_spread=6,
79     ),
80 rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
81     extra_counts_per_division=[0, 0, 1, 2, 0, 1],
82     output_masks=[
83         rhythmmakertools.SustainMask(
84             indices=[1],
85             period=3,
86             ),
87         ],
88     talea=rhythmmakertools.Talea(
89         counts=[
90             1, -1,
91             1, 1, -1,
92             1, 1, 1, -1,
93             1, 1, 1, 1, 1, 1, -1,
94             1,
95             denominator=16,
96             ),
97         )
98     )

```

### D.2.30 ERSILIA.MATERIALS.SHAKER\_DECELERANDO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import selectortools
7 from ersilia.materials import abbreviations
8
9
10 shaker_decelerando_musicSpecifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         staccati=indicatortools.Articulation('staccato'),
13         dynamic_expression=consort.DynamicExpression(
14             start_dynamic_tokens='f mf mp',
15             stop_dynamic_tokens='p',
16             ),
17         percussion_staff=abbreviations.percussion_staff,
18         text_spanner=consort.AttachmentExpression(
19             attachments=abbreviations.make_text_spanner('shaker'),
20             selector=selectortools.Selector().by_leaves(),
21             ),
22         ),
23         color='blue',
24         labels=['shakers'],
25         pitch_handler=consort.AbsolutePitchHandler(
26             pitches_are_nonsemantic=True,

```

```

27     ),
28     rhythm_maker=rhythmmakertools.AccelerandoRhythmMaker(
29         beamSpecifier=rhythmmakertools.BeamSpecifier(
30             use_feather_beams=True,
31         ),
32         duration_spellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
33             forbid_meter_rewriting=True,
34         ),
35         interpolationSpecifiers=rhythmmakertools.InterpolationSpecifier(
36             start_duration=durationtools.Duration(1, 32),
37             stop_duration=durationtools.Duration(1, 8),
38             written_duration=durationtools.Duration(1, 32),
39         ),
40         tuplet_spellingSpecifier=rhythmmakertools.TupletSpellingSpecifier(
41             use_note_duration_bracket=True,
42         ),
43     ),
44 )

```

### D.2.31 ERSILIA.MATERIALS.SHAKER\_SPORADIC\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 shaker_sporadic_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expression=consort.DynamicExpression(
13             dynamic_tokens='ppp',
14             transitions=['constante'],
15         ),
16         percussion_staff=abbreviations.percussion_staff,
17         staccati=consort.AttachmentExpression(
18             attachments=indicatortools.Articulation('staccato'),
19             selector=selectortools.Selector()
20                 .by_logical_tie(pitched=True)
21                 .by_duration('<=', (1, 16), preprolated=True)
22                 [0],
23         ),
24         text_spanner=consort.AttachmentExpression(
25             attachments=abbreviations.make_text_spanner('shaker'),
26             selector=selectortools.Selector().by_leaves(),
27         ),
28         tremolo=consort.AttachmentExpression(
29             attachments=spannertools.StemTremoloSpanner(),
30             selector=selectortools.Selector()
31                 .by_logical_tie(pitched=True)
32                 .by_duration('>', (1, 16), preprolated=True)
33         ),

```

```

34     ),
35     color='blue',
36     labels=['shakers'],
37     pitch_handler=consort.AbsolutePitchHandler(
38         pitches_are_nonsemantic=True,
39     ),
40     rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
41         extra_counts_per_division=[0, 0, 1],
42         output_masks=[
43             rhythmmakertools.SustainMask(
44                 indices=[2],
45                 period=3,
46             ),
47         ],
48         talea=rhythmmakertools.Talea(
49             counts=[1, 1, -2, 1, -2, 1, 1, 1, -3, 1, 1, -2, 1, 1, -1],
50             denominator=16,
51         ),
52     ),
53 )

```

### D.2.32 ERSILIA.MATERIALS.SHAKER\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import scoretools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8 from ersilia.materials import abbreviations
9
10
11 shaker_tremolo_music_specifier = consort.MusicSpecifier(
12     attachment_handler=consort.AttachmentHandler(
13         accents=consort.AttachmentExpression(
14             attachments=indicatortools.Articulation('accent'),
15             selector=selectortools.Selector()
16             .by_leaves()
17             .by_run(scoretools.Note)
18             [0],
19         ),
20         dynamic_expression=consort.DynamicExpression(
21             dynamic_tokens='fp',
22         ),
23         percussion_staff=abbreviations.percussion_staff,
24         text_spinner=consort.AttachmentExpression(
25             attachments=abbreviations.make_text_spinner('shaker'),
26             selector=selectortools.Selector().by_leaves(),
27         ),
28         tremolo_spinner=spannertools.StemTremoloSpanner(),
29     ),
30     color='blue',
31     labels=['shakers'],

```

```

32     pitch_handler=consort.AbsolutePitchHandler(
33         pitches_are_nonsemantic=True,
34         ),
35     rhythm_maker=rhythmmakertools.NoteRhythmMaker(
36         tieSpecifier=rhythmmakertools.TieSpecifier(
37             tie_across_divisions=True,
38             ),
39         ),
40     )

```

### D.2.33 ERSILIA.MATERIALS.SPARSE\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5
6
7 sparse_timespan_maker = consort.TaleaTimespanMaker(
8     initial_silence_talea=rhythmmakertools.Talea(
9         counts=[1, 0, 3, 2, 4, 5, 1, 2],
10        denominator=8,
11        ),
12     playing_talea=rhythmmakertools.Talea(
13         counts=[2, 3, 2, 2, 3],
14         denominator=8,
15         ),
16     playing_groupings=[1],
17     repeat=True,
18     silence_talea=rhythmmakertools.Talea(
19         counts=[4, 8, 6],
20         denominator=8,
21         ),
22     step_anchor=Right,
23     synchronize_groupings=False,
24     synchronize_step=False,
25     timespanSpecifier=consort.TimespanSpecifier(
26         minimum_duration=durationtools.Duration(1, 8),
27         ),
28     )

```

### D.2.34 ERSILIA.MATERIALS.STRING\_AGITATO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 string_agitato_musicSpecifier = consort.MusicSpecifier(
11     attachmentHandler=consort.AttachmentHandler(

```

```

12     dynamic_expressions=consort.DynamicExpression(
13         division_period=2,
14         dynamic_tokens='mp fff',
15         start_dynamic_tokens='f p',
16         stop_dynamic_tokens='p f',
17         ),
18     harmonics=consort.AttachmentExpression(
19         attachments=consort.HarmonicExpression('P4'),
20         is_destructive=True,
21         selector=selectortools.Selector()
22             .by_logical_tie(pitched=True)
23             .by_duration('==', (1, 16), preprolated=True)
24             .by_pattern(
25                 pattern=rhythmmakertools.BooleanPattern(
26                     indices=[2, 4],
27                     period=5,
28                     ),
29             )
30         ),
31     slur=consort.AttachmentExpression(
32         attachments=spannertools.Slur(),
33         selector=selectortools.Selector()
34             .by_logical_tie(pitched=True)
35             .by_duration('==', (1, 16), preprolated=True)
36             .by_contiguity()
37             .by_length('>', 1)
38             .by_leaves()
39         ),
40     staccati=consort.AttachmentExpression(
41         attachments=indicatortools.Articulation('staccato'),
42         selector=selectortools.Selector()
43             .by_logical_tie(pitched=True)
44             .by_duration('==', (1, 16), preprolated=True)
45             .by_contiguity()
46             .by_leaves()
47         ),
48     accents=consort.AttachmentExpression(
49         attachments=indicatortools.Articulation('accent'),
50         selector=selectortools.Selector()
51             .by_logical_tie(pitched=True)
52             .by_duration('>', (1, 16), preprolated=True)
53             .by_contiguity()
54             .by_leaves()
55             [0]
56         ),
57     trill_spinner=consort.AttachmentExpression(
58         attachments=[
59             spannertools.ComplexTrillSpinner('+m3'),
60             spannertools.ComplexTrillSpinner('+P4'),
61             ],
62         selector=selectortools.Selector()
63             .by_logical_tie(pitched=True)
64             .by_duration('>', (1, 16), preprolated=True)
65             .by_contiguity()

```

```

66         .by_leaves()
67     ),
68 ),
69 color='magenta',
70 labels=[],
71 pitch_handler=consort.PitchClassPitchHandler(
72     forbid_repetitions=True,
73     pitch_application_rate='division',
74     pitch_specifier=abbreviations.agitato_pitchSpecifier,
75     registerSpecifier=consort.RegisterSpecifier(
76         base_pitch='G3',
77         phrase_inflections=consort.RegisterInflection.zigzag(6)
78             .reverse()
79             .align(),
80         segment_inflections=consort.RegisterInflection
81             .descending(width=6)
82             .align()
83     ),
84     register_spread=3,
85 ),
86 rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
87     extra_counts_per_division=[0, 0, 1, 2, 0, 1],
88     output_masks=[
89         rhythmmakertools.SustainMask(
90             indices=[1],
91             period=3,
92         ),
93     ],
94     talea=rhythmmakertools.Talea(
95         counts=[
96             1, -1,
97             1, 1, -1,
98             1, 1, 1, -1,
99             1, 1, 1, 1, -1,
100            1, 1, 1, 1, 1, -1,
101            1, 1, -2,
102        ],
103        denominator=16,
104    ),
105 ),
106 )

```

### D.2.35 ERSILIA.MATERIALS.STRING\_LEGATO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import indicatortools
5 from abjad.tools import rhythmmakertools
6 from abjad.tools import selectortools
7 from abjad.tools import spannertools
8
9
10 string_legato_musicSpecifier = consort.MusicSpecifier(

```

```

11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expressions=consort.DynamicExpression(
13             dynamic_tokens='p mf',
14             start_dynamic_tokens='o fp',
15             stop_dynamic_tokens='o ff'
16         ),
17         glissando=spannertools.Glissando(),
18         tenuti=consort.AttachmentExpression(
19             attachments=indicatortools.Articulation('tenuto'),
20             selector=selectortools.Selector()
21                 .by_leaves()
22                 .by_logical_tie(pitched=True)
23                 .by_pattern(
24                     rhythmmakertools.BooleanPattern(
25                         indices=[3],
26                         period=4,
27                     ),
28                 )
29             [0]
30         ),
31         tremolo_trill=consort.AttachmentExpression(
32             attachments=(
33                 spannertools.ComplexTrillSpanner(interval='+m3'),
34                 spannertools.StemTremoloSpanner(),
35                 spannertools.ComplexTrillSpanner(interval='+m3'),
36                 spannertools.ComplexTrillSpanner(interval='+M2'),
37                 spannertools.StemTremoloSpanner(),
38             ),
39             selector=selectortools.Selector()
40                 .by_leaves()
41                 .by_logical_tie(pitched=True)
42                 .by_pattern(
43                     rhythmmakertools.BooleanPattern(
44                         indices=[0, 1, 2],
45                         period=4,
46                     ),
47                 ),
48             ),
49         ),
50         color='green',
51         labels=[],
52         pitch_handler=consort.AbsolutePitchHandler(
53             pitchSpecifier="d' f' d' fqs' ef' d' ef' f' fqs' d' g' d' d' as",
54             pitchApplicationRate='division',
55         ),
56         rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
57             denominators=[8, 4, 8, 1],
58             durationSpellingSpecifier=rhythmmakertools.DurationSpellingSpecifier(
59                 forbiddenWrittenDuration=durationtools.Duration(1, 2),
60             ),
61             extraCountsPerDivision=[0, 1, 0, 2, 1],
62         )
63     )

```

### D.2.36 ERSILIA.MATERIALS.STRING\_LOW\_PEDAL\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7
8
9 string_low_pedal_musicSpecifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         accents=consort.AttachmentExpression(
12             attachments=indicatortools.Articulation('accent'),
13             selector=selectortools.Selector()
14             .by_logical_tie()
15             .get_slice(start=1, apply_to_each=False)
16             [0]
17         ),
18         dynamic_expressions=consort.DynamicExpression(
19             division_period=2,
20             dynamic_tokens='p ppp',
21             start_dynamic_tokens='o',
22             stop_dynamic_tokens='o',
23             ),
24         glissando=spannertools.Glissando(),
25         ),
26         color=None,
27         labels=[],
28         minimum_phrase_duration=(3, 2),
29         pitch_handler=consort.PitchClassPitchHandler(
30             pitch_application_rate='phrase',
31             deviations=[0, 0, 0, 0.5],
32             pitchSpecifier='d f d g f d f',
33             registerSpecifier=consort.RegisterSpecifier(
34                 base_pitch='C4',
35                 ),
36             register_spread=0,
37             ),
38         rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
39             denominators=[8],
40             output_masks=[
41                 rhythmmakertools.SustainMask(
42                     indices=[0, 1],
43                     period=3,
44                     ),
45                 rhythmmakertools.SustainMask(
46                     indices=[0, -1],
47                     ),
48                 ],
49             tieSpecifier=rhythmmakertools.TieSpecifier(
50                 tie_across_divisions=True,
51                 ),
52             ),
```

53 )

### D.2.37 ERSILIA.MATERIALS.STRING\_OSTINATO\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 string_ostinato_music_specifier = consort.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         dynamic_expressions=consort.DynamicExpression(
13             dynamic_tokens='p',
14         ),
15         pizzicati=consort.AttachmentExpression(
16             attachments=[
17                 [
18                     abbreviations.make_text_markup('pizz.'),
19                     indicatortools.Articulation('snappizzicato'),
20                 ],
21             ],
22             selector=selectortools.Selector()
23             .by_logical_tie(pitched=True)
24             .by_contiguity()
25             .by_length('==', 1)
26             .by_duration('==', (1, 16), preprolated=True)
27             .by_leaves()
28             [0]
29         ),
30         tenuti=consort.AttachmentExpression(
31             attachments=indicatortools.Articulation('tenuto'),
32             selector=selectortools.Selector()
33             .by_logical_tie(pitched=True)
34             .by_contiguity()
35             .by_length('==', 1)
36             .by_duration('>', (1, 16), preprolated=True)
37             .by_leaves()
38             [0]
39         ),
40         slur=consort.AttachmentExpression(
41             attachments=spannertools.Slur(),
42             selector=selectortools.Selector()
43             .by_logical_tie(pitched=True)
44             .by_contiguity()
45             .by_length('>', 1)
46             .by_leaves()
47         ),
48     ),
49     color='darkyellow',
50     pitch_handler=consort.PitchClassPitchHandler(
```

```

51     deviations=[0, 0, 0, 0.5, 0, -0.5],
52     forbid_repetitions=True,
53     leap_constraint=6,
54     pitchSpecifier='d f d f d f c f bf d f df',
55     registerSpecifier=consort.RegisterSpecifier(
56         base_pitch='C4',
57         segment_inflections=consort.RegisterInflection
58             .zigzag(6)
59             .reverse(),
60         ),
61     register_spread=3,
62     ),
63     rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
64         extra_counts_per_division=[0, 0, 1, 2, 0, 1],
65         talea=rhythmmakertools.Talea(
66             counts=[1, 1, -3, 2, 1, -2, 3, 1, -3],
67             denominator=16,
68             )
69         ),
70     )

```

### D.2.38 ERSILIA.MATERIALS.STRING\_OVERPRESSURE\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from ersilia.materials import abbreviations
7
8
9 string_overpressure_musicSpecifier = consortium.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         accents=consort.AttachmentExpression(
12             attachments=indicatortools.Articulation('accent'),
13             selector=selectortools.Selector()
14                 .by_logical_tie(pitched=True)
15                 [0]
16             ),
17         dynamic_expressions=consort.DynamicExpression(
18             start_dynamic_tokens='fff',
19             only_first=True,
20             ),
21         text_spanner=consort.AttachmentExpression(
22             attachments=abbreviations.make_text_spanner('overpressure'),
23             selector=selectortools.Selector().by_leaves(),
24             ),
25         ),
26     color=None,
27     labels=[],
28     pitch_handler=consort.AbsolutePitchHandler(
29         deviations=[0, 1, 0, 0.5],
30         logical_tie_expressions=[
31             consortium.ChordExpression(chord_expr=[0, 7]),

```

```

32         ],
33     pitch_application_rate='phrase',
34     pitchSpecifier='A3 B3 Bb3 C4',
35   ),
36 rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
37   denominators=[8],
38   extra_counts_per_division=[0, 1],
39   output_masks=[
40     rhythmmakertools.SustainMask(
41       indices=[2],
42       period=3,
43     ),
44     rhythmmakertools.SustainMask(
45       indices=[0, -1],
46     ),
47   ],
48   tieSpecifier=rhythmmakertools.TieSpecifier(
49     tie_across_divisions=True,
50   ),
51 ),
52 )

```

## D.2.39 ERSILIA.MATERIALS.STRING\_POINTILLIST\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from ersilia.materials import abbreviations
7
8
9 string_pointillist_musicSpecifier = consort.MusicSpecifier(
10   attachment_handler=consort.AttachmentHandler(
11     mordents=consort.AttachmentExpression(
12       attachments=indicatortools.Articulation('mordent'),
13       selector=selectortools.Selector()
14         .by_logical_tie(pitched=True)
15       [0]
16     ),
17     dynamic_expressions=consort.DynamicExpression(
18       start_dynamic_tokens='ppp',
19       only_first=True,
20     ),
21     text_spanner=consort.AttachmentExpression(
22       attachments=consort.AttachmentExpression(
23         attachments=abbreviations.make_text_spanner('pizz.'),
24         selector=selectortools.Selector().by_leaves(),
25       ),
26       selector=selectortools.Selector().by_leaves(),
27     ),
28   ),
29   color='darkyellow',
30   labels=[]

```

```

31     pitch_handler=consort.PitchClassPitchHandler(
32         forbid_repetitions=True,
33         pitchSpecifier=abbreviations.agitato_pitchSpecifier,
34         registerSpecifier=consort.RegisterSpecifier(
35             base_pitch='G3',
36             phraseInflections=consort.RegisterInflection
37                 .zigzag(3)
38                 .reverse()
39                 .align(),
40             segmentInflections=consort.RegisterInflection
41                 .descending(width=3)
42                 .align()
43         ),
44     ),
45     rhythmMaker=consort.CompositeRhythmMaker(
46         default=rhythmmakertools.TaleaRhythmMaker(
47             extra_counts_per_division=[0, 0, 1],
48             talea=rhythmmakertools.Talea(
49                 counts=[1, -1, 1, -2, 1, -3],
50                 denominator=16,
51             ),
52         ),
53         last=rhythmmakertools.IncisedRhythmMaker(
54             inciseSpecifier=rhythmmakertools.InciseSpecifier(
55                 fill_with_notes=False,
56                 prefix_counts=[1],
57                 prefix_talea=[1],
58                 talea_denominator=16,
59             ),
60         ),
61     ),
62 )

```

#### D.2.40 ERSILIA.MATERIALS.STRING\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 string_tremolo_music_specifier = consortium.MusicSpecifier(
11     attachmentHandler=consort.AttachmentHandler(
12         accents=consort.AttachmentExpression(
13             attachments=indicatortools.Articulation('accent'),
14             selector=selectortools.Selector()
15                 .by_logical_tie()
16                 .by_duration('==', (1, 16), preprolated=True)
17                 .with_next_leaf()
18         ),
19         dynamicExpressions=consort.DynamicExpression(

```

```

20     division_period=2,
21     dynamic_tokens='p ppp mp',
22     start_dynamic_tokens='o fp',
23   ),
24   stem_tremolo=spannertools.StemTremoloSpanner(),
25   ),
26   color='red',
27   labels=[],
28   pitch_handler=consort.PitchClassPitchHandler(
29     deviations=[0, 0.5, 0, -0.5],
30     logical_tie_expressions=[
31       consort.ChordExpression([0, 8]),
32     ],
33     pitch_application_rate='phrase',
34     pitchSpecifier=abbreviations.agitato_pitchSpecifier,
35     pitchOperationSpecifier=abbreviations.pitch_operationSpecifier,
36     registerSpecifier=consort.RegisterSpecifier(
37       base_pitch='G3',
38     ),
39   ),
40   rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
41     denominators=[16],
42     extra_counts_per_division=[0, 1],
43     output_masks=[
44       rhythmmakertools.SustainMask(
45         indices=[2],
46         period=3,
47       ),
48       rhythmmakertools.SustainMask(
49         indices=[0, -1],
50       ),
51     ],
52     tieSpecifier=rhythmmakertools.TieSpecifier(
53       tie_across_divisions=True,
54     ),
55   ),
56 )

```

#### D.2.41 ERSILIA.MATERIALS.SUSTAINED\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5
6
7 sustained_timespan_maker = consort.TaleaTimespanMaker(
8   initial_silence_talea=rhythmmakertools.Talea(
9     counts=(0, 2, 1),
10    denominator=8,
11   ),
12   playing_talea=rhythmmakertools.Talea(
13     counts=(4, 5, 4, 3, 7, 6),
14     denominator=8,

```

```

15     ),
16     playing_groupings=(3, 4, 2, 2, 3, 5),
17     repeat=True,
18     silence_talea=rhythmmakertools.Talea(
19         counts=(2, 1, 2, 1, 3, 7, 1, 2, 9),
20         denominator=8,
21         ),
22     step_anchor=Right,
23     synchronize_groupings=False,
24     synchronize_step=False,
25     timespanSpecifier=consort.TimespanSpecifier(
26         minimum_duration=durationtools.Duration(1, 8),
27         ),
28 )

```

#### D.2.42 ERSILIA.MATERIALS.TUTTI\_TIMESPAN MAKER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import durationtools
4 from abjad.tools import rhythmmakertools
5
6
7 tutti_timespan_maker = consort.TaleaTimespanMaker(
8     fuse_groups=True,
9     playing_talea=rhythmmakertools.Talea(
10        counts=(4, 5, 4, 3, 7, 6),
11        denominator=8,
12        ),
13     playing_groupings=(3, 4, 2, 2, 3, 5),
14     repeat=True,
15     silence_talea=rhythmmakertools.Talea(
16        counts=(3, 4, 2, 5, 6, 9),
17        denominator=4,
18        ),
19     step_anchor=Right,
20     synchronize_groupings=True,
21     synchronize_step=True,
22     timespanSpecifier=consort.TimespanSpecifier(
23         minimum_duration=durationtools.Duration(1, 4),
24         ),
25 )

```

#### D.2.43 ERSILIA.MATERIALS.WIND\_AGITATO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9

```

```

10 wind_agitato_musicSpecifier = consort.MusicSpecifier(
11     attachmentHandler=consort.AttachmentHandler(
12         accents_short=consort.AttachmentExpression(
13             attachments=[
14                 [
15                     indicatorTools.Articulation('accent'),
16                     indicatorTools.Articulation('staccatissimo'),
17                 ],
18             ],
19             selector=selectortools.Selector()
20                 .by_logical_tie(pitched=True)
21                 .by_duration('==', (1, 16), preprolated=True)
22                 .by_contiguity()
23                 .by_length('==', 1)
24                 .by_leaves()
25             [0]
26         ),
27         accents_long=consort.AttachmentExpression(
28             attachments=indicatorTools.Articulation('accent'),
29             selector=selectortools.Selector()
30                 .by_logical_tie(pitched=True)
31                 .by_duration('>', (1, 16), preprolated=True)
32                 .by_contiguity()
33                 .by_length('==', 1)
34                 .by_leaves()
35             [0]
36         ),
37         dynamic_expressions=consort.DynamicExpression(
38             division_period=2,
39             dynamic_tokens='mf mp fff',
40             start_dynamic_tokens='f',
41             stop_dynamic_tokens='p mp mf',
42         ),
43         flutter_tongue=consort.AttachmentExpression(
44             attachments=abbreviations.make_text_spanner('Flz.'),
45             selector=selectortools.Selector()
46                 .by_logical_tie(pitched=True)
47                 .by_duration('>', (1, 16), preprolated=True)
48                 .by_contiguity()
49                 .by_leaves()
50         ),
51         slur=consort.AttachmentExpression(
52             attachments=spannertools.Slur(),
53             selector=selectortools.Selector()
54                 .by_logical_tie(pitched=True)
55                 .by_duration('==', (1, 16), preprolated=True)
56                 .by_contiguity()
57                 .by_length('>', 1)
58                 .by_pattern(
59                     pattern=rhythmmakertools.BooleanPattern(
60                         indices=[0],
61                         period=2,
62                     ),
63                 )

```

```

64         .by_leaves()
65     ),
66     staccati=consort.AttachmentExpression(
67         attachments=indicatortools.Articulation('staccato'),
68         selector=selectortools.Selector()
69             .by_logical_tie(pitched=True)
70             .by_duration('==', (1, 16), preprolated=True)
71             .by_contiguity()
72             .by_length('>', 1)
73             .by_pattern(
74                 pattern=rhythmmakertools.BooleanPattern(
75                     indices=[0, 2],
76                     period=3,
77                 ),
78             )
79             .by_leaves()
80         ),
81     stem_tremolo_spanner=consort.AttachmentExpression(
82         attachments=spannertools.StemTremoloSpanner(),
83         selector=selectortools.Selector()
84             .by_logical_tie(pitched=True)
85             .by_duration('>', (1, 16), preprolated=True)
86             .by_contiguity()
87             .by_leaves()
88         ),
89     ),
90     color='magenta',
91     labels=[],
92     pitch_handler=consort.PitchClassPitchHandler(
93         forbid_repetitions=True,
94         pitch_specifier=abbreviations.agitato_pitch_specifier,
95         register_specifier=consort.RegisterSpecifier(
96             base_pitch='C4',
97             phrase_inflections=consort.RegisterInflection.zigzag(6)
98                 .reverse()
99                 .align(),
100             segment_inflections=consort.RegisterInflection
101                 .descending(width=12)
102                 .align()
103         ),
104         register_spread=6,
105     ),
106     rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
107         extra_counts_per_division=[0, 0, 1, 2, 0, 1],
108         output_masks=[
109             rhythmmakertools.SustainMask(
110                 indices=[1],
111                 period=3,
112             ),
113         ],
114         talea=rhythmmakertools.Talea(
115             counts=[
116                 1, -1,
117                 1, 1, -1,

```

```

118         1, 1, 1, -1,
119         1, 1, -2,
120         1, 1, 1, 1, -1,
121         ],
122         denominator=16,
123     ),
124 )
125 )

```

#### D.2.44 ERSILIA.MATERIALS.WIND\_CONTINUO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import scoretools
6 from abjad.tools import selectortools
7
8
9 wind_continuo_musicSpecifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         dynamic_expressions=consort.DynamicExpression(
12             division_period=2,
13             dynamic_tokens='p mp mf',
14             start_dynamic_tokens='o',
15             stop_dynamic_tokens='o',
16             ),
17             staccati=consort.AttachmentExpression(
18                 attachments=indicatortools.Articulation('staccato'),
19                 selector=selectortools.select_pitched_runs(),
20                 ),
21             ),
22             color=None,
23             labels=[],
24             pitch_handler=consort.PitchClassPitchHandler(
25                 deviations=[0, 2, 0, 3, 0, 3, 0, 2, 0, 5, 0, 3, 0, 5, 0, 8, 7],
26                 pitchSpecifier="d' f' df",
27                 pitch_application_rate='division',
28                 registerSpecifier=consort.RegisterSpecifier(
29                     base_pitch='C4',
30                     ),
31                 ),
32             rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
33                 burnishSpecifier=rhythmmakertools.BurnishSpecifier(
34                     left_classes=[scoretools.Rest],
35                     left_counts=[1, 1, 0, 0, 0, 1, 0],
36                     right_classes=[scoretools.Rest],
37                     right_counts=[1, 0],
38                     ),
39                     denominators=[16],
40                     extra_counts_per_division=(0, 0, 1, 2, 0, 1),
41                     )
42             )

```

#### D.2.45 ERSILIA.MATERIALS.WIND\_LOW\_PEDAL\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import rhythmmakertools
4
5
6 wind_low_pedal_musicSpecifier = consort.MusicSpecifier(
7     attachment_handler=consort.AttachmentHandler(
8         dynamic_expressions=consort.DynamicExpression(
9             division_period=2,
10            dynamic_tokens='p ppp',
11            start_dynamic_tokens='o',
12            stop_dynamic_tokens='o',
13        )
14    ),
15    color=None,
16    labels=[],
17    minimum_phrase_duration=(3, 2),
18    pitch_handler=consort.AbsolutePitchHandler(
19        pitchSpecifier='D2 F2 D2 G2 F2 D2 F2',
20    ),
21    rhythm_maker=rhythmmakertools.NoteRhythmMaker(
22        tieSpecifier=rhythmmakertools.TieSpecifier(
23            tie_across_divisions=True,
24        ),
25    ),
26 )
```

#### D.2.46 ERSILIA.MATERIALS.WIND\_OSTINATO\_MUSIC\_SPECIFIER

```
1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7
8
9 wind_ostinato_musicSpecifier = consort.MusicSpecifier(
10    attachment_handler=consort.AttachmentHandler(
11        dynamic_expressions=consort.DynamicExpression(
12            dynamic_tokens='p',
13        ),
14        slur=consort.AttachmentExpression(
15            attachments=spannertools.Slur(),
16            selector=selectortools.Selector(
17                .by_logical_tie(pitched=True)
18                .by_duration('==', (1, 16), preprolated=True)
19                .by_contiguity()
20                .by_length('>', 1)
21                .by_leaves()
22            ),
23        staccatti=consort.AttachmentExpression(
```

```

24     attachments=indicatortools.Articulation('staccato'),
25     selector=selectortools.Selector()
26         .by_logical_tie(pitched=True)
27         .by_duration('==', (1, 16), preprolated=True)
28         .by_contiguity()
29         .by_leaves()
30     [-1]
31     ),
32     ),
33 color='darkyellow',
34 pitch_handler=consort.AbsolutePitchHandler(
35     forbid_repetitions=True,
36     pitch_specifier="d' f",
37     ),
38 rhythm_maker=rhythmmakertools.TaleaRhythmMaker(
39     extra_counts_per_division=[0, 0, 1, 2, 0, 1],
40     talea=rhythmmakertools.Talea(
41         counts=[1, 1, -3],
42         denominator=16,
43         )
44     ),
45 )

```

#### D.2.47 ERSILIA.MATERIALS.WIND\_POINTILLIST\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from ersilia.materials import abbreviations
7
8
9 wind_pointillist_musicSpecifier = consort.MusicSpecifier(
10     attachment_handler=consort.AttachmentHandler(
11         dynamic_expressions=consort.DynamicExpression(
12             start_dynamic_tokens='ppp',
13             only_first=True,
14             ),
15         mordent=consort.AttachmentExpression(
16             attachments=indicatortools.Articulation('mordent'),
17             selector=selectortools.Selector()
18             .by_logical_tie(pitched=True)
19             [0]
20             ),
21         ),
22 color='darkyellow',
23 labels=[],
24 pitch_handler=consort.PitchClassPitchHandler(
25     forbid_repetitions=True,
26     pitch_specifier=abbreviations.agitato_pitchSpecifier,
27     registerSpecifier=consort.RegisterSpecifier(
28         base_pitch='C4',
29         phrase_inflections=consort.RegisterInflection

```

```

30         .zigzag(6)
31         .reverse()
32         .align(),
33     segment_inflections=consort.RegisterInflection
34         .descending(width=6)
35         .align()
36     ),
37 ),
38 rhythm_maker=consort.CompositeRhythmMaker(
39     default=rhythmmakertools.TaleaRhythmMaker(
40         extra_counts_per_division=[0, 1],
41         talea=rhythmmakertools.Talea(
42             counts=[1, 1, -1, 1, -2, 1, 1, -3],
43             denominator=16,
44         ),
45     ),
46     last=rhythmmakertools.IncisedRhythmMaker(
47         inciseSpecifier=rhythmmakertools.InciseSpecifier(
48             fill_with_notes=False,
49             prefix_counts=[2],
50             prefix_talea=[1, 0],
51             talea_denominator=16,
52         ),
53     ),
54 ),
55 )

```

#### D.2.48 ERSILIA.MATERIALS.WIND\_TREMOLO\_MUSIC\_SPECIFIER

```

1 # -*- encoding: utf-8 -*-
2 import consort
3 from abjad.tools import indicatortools
4 from abjad.tools import rhythmmakertools
5 from abjad.tools import selectortools
6 from abjad.tools import spannertools
7 from ersilia.materials import abbreviations
8
9
10 wind_tremolo_music_specifier = consortium.MusicSpecifier(
11     attachment_handler=consort.AttachmentHandler(
12         outer_accents=consort.AttachmentExpression(
13             attachments=indicatortools.Articulation('accent'),
14             selector=selectortools.select_pitched_runs()[0],
15         ),
16         inner_accents=consort.AttachmentExpression(
17             attachments=indicatortools.Articulation('accent'),
18             selector=selectortools.Selector(
19                 .by_logical_tie()
20                 .by_duration('==', (1, 8), preprolated=True)
21                 .with_next_leaf()
22             ),
23             dynamic_expressions=consort.DynamicExpression(
24                 division_period=2,
25                 dynamic_tokens='p ppp mp',

```

```

26         start_dynamic_tokens='fp',
27         stop_dynamic_tokens='mf ppp',
28     ),
29     trill_spanner=consort.AttachmentExpression(
30         attachments=spannertools.ComplexTrillSpanner('M2'),
31         selector=selectortools.select_pitched_runs(),
32     ),
33 ),
34 color='red',
35 labels=[],
36 pitch_handler=consort.PitchClassPitchHandler(
37     deviations=[0, 1],
38     pitch_application_rate='phrase',
39     pitch_specifier=abbreviations.agitato_pitch_specifier,
40     pitch_operation_specifier=abbreviations.pitch_operation_specifier,
41     register_specifier=consort.RegisterSpecifier(
42         base_pitch='C4',
43     ),
44 ),
45 rhythm_maker=rhythmmakertools.EvenDivisionRhythmMaker(
46     denominators=[8],
47     extra_counts_per_division=[0, 1, 2],
48     output_masks=[
49         rhythmmakertools.SustainMask(
50             indices=[2],
51             period=3,
52         ),
53         rhythmmakertools.SustainMask(
54             indices=[0, -1],
55         ),
56     ],
57     tie_specifier=rhythmmakertools.TieSpecifier(
58         tie_across_divisions=True,
59     ),
60 ),
61 )

```

### D.3 ERSILIA

#### SEGMENTS

SOURCE

##### D.3.1 ERSILIA.SEGMENTS.CHEMISH

```

1 # -*- encoding: utf-8 -*-
2 import abjad
3 import consort
4 import ersilia
5 from abjad import new
6 from abjad.tools import rhythmmakertools
7 from abjad.tools import timespantools
8 from ersilia.materials import abbreviations
9
10
11 ### SEGMENT ###
12
13 segment_maker = ersilia.ErsiliaSegmentMaker(

```

```

14     desired_duration_in_seconds=90,
15     name='Chemish',
16     permitted_time_signatures=ersilia.permitted_time_signatures,
17     tempo=abjad.Tempo((1, 4), 80),
18   )
19
20 ### PEDAL ###
21
22 segment_maker.add_setting(
23   timespan_identifier=timespantools.Timespan(start_offset=1),
24   timespan_maker=new(
25     ersilia.sustained_timespan_maker,
26     fuse_groups=True,
27   ),
28   percussion=ersilia.percussion_low_pedal_music_specifier,
29 )
30
31 segment_maker.add_setting(
32   timespan_identifier=[-1, 1],
33   timespan_maker=new(
34     ersilia.sustained_timespan_maker,
35     fuse_groups=True,
36   ),
37   bass=new(
38     ersilia.string_low_pedal_music_specifier,
39     pitch_handler__register_spread=0,
40     ).transpose('E1'),
41   )
42
43 ### TREMOLO ###
44
45 segment_maker.add_setting(
46   timespan_identifier=[
47     -1, 2,
48     -3, 10,
49     -4,
50   ],
51   timespan_maker=new(
52     ersilia.sustained_timespan_maker,
53     fuse_groups=True,
54   ),
55   flute=ersilia.wind_tremolo_music_specifier,
56   clarinet=ersilia.wind_tremolo_music_specifier,
57 )
58
59 segment_maker.add_setting(
60   timespan_maker=new(
61     ersilia.sparse_timespan_maker,
62     silence_talea_denominator=4,
63   ),
64   violin=ersilia.string_tremolo_music_specifier,
65   viola=new(
66     ersilia.string_tremolo_music_specifier,
67     pitch_handler__registerSpecifier__base_pitch='C3',

```

```

68     ),
69     cello=new(
70         ersilia.string_tremolo_music_specifier,
71         pitch_handler__register_specifier__base_pitch='C2',
72         ),
73     )
74
75 ### CONTINUO ###
76
77 segment_maker.add_setting(
78     timespan_identifier=[-1, 1, -2, 1, -3, 1, -2],
79     timespan_maker=ersilia.sparse_timespan_maker,
80     piano_lh=ersilia.piano_string_glissando_music_specifier,
81     )
82
83 ### OSTINATO ###
84
85 ### AGITATO ###
86
87 musicSpecifier = new(
88     ersilia.saxophone_agitato_music_specifier,
89     attachment_handler__dynamic_expressions=consort.DynamicExpression(
90         start_dynamic_tokens='fp o',
91         stop_dynamic_tokens='mp mf p f o',
92         dynamic_tokens='pp p',
93         ),
94         pitch_handler__register_specifier__base_pitch='C4',
95         pitch_handler__register_specifier__segment_inflections=consort.RegisterInflection
96             .ascending(width=12)
97             .align()
98         )
99
100 segment_maker.add_setting(
101     timespan_identifier=[
102         -1, 6,
103         -5, 1,
104         -1, 2,
105         -1, 1,
106         ],
107     timespan_maker=ersilia.dense_timespan_maker,
108     saxophone=musicSpecifier,
109     )
110
111 segment_maker.add_setting(
112     timespan_maker=ersilia.sparse_timespan_maker,
113     flute=musicSpecifier,
114     oboe=musicSpecifier,
115     clarinet=musicSpecifier.transpose(-12),
116     )
117
118 segment_maker.add_setting(
119     timespan_identifier=[-1, 5],
120     timespan_maker=ersilia.sparse_timespan_maker,
121     guitar=ersilia.guitar_strummed_music_specifier,

```

```

122     )
123
124 ### POINTILLIST ###
125
126 segment_maker.add_setting(
127     timespan_identifier=[-1, 1, -1, 1, -1],
128     timespan_maker=ersilia.sparse_timespan_maker,
129     piano_rh=new(
130         ersilia.piano_pointillist_music_specifier,
131         pitch_handler__leap_constraint=8,
132         ),
133     )
134
135 ### AUXILIARY ###
136
137 segment_maker.add_setting(
138     timespan_identifier>[
139         4, -1,
140         2, -1,
141         1, -1,
142         5,
143     ],
144     timespan_maker=new(
145         ersilia.sustained_timespan_maker,
146         fuse_groups=True,
147         initial_silence_talea=None,
148         padding=(1, 4),
149         playing_groupings=[2, 3, 2, 4, 6],
150         silence_talea__denominator=2,
151         timespanSpecifier=consort.TimespanSpecifier(
152             minimum_duration=0,
153             ),
154         ),
155         flute=ersilia.shaker_tremolo_music_specifier,
156         clarinet=ersilia.shaker_tremolo_music_specifier,
157         oboe=ersilia.shaker_tremolo_music_specifier,
158         violin=ersilia.shaker_tremolo_music_specifier,
159         viola=ersilia.shaker_tremolo_music_specifier,
160         cello=ersilia.shaker_tremolo_music_specifier,
161     )
162
163 #segment_maker.add_setting(
164 #    timespan_identifier=[-2, 1, -3, 1, -5, 1, -3],
165 #    timespan_maker=new(
166 #        ersilia.sparse_timespan_maker,
167 #        padding=(1, 4),
168 #        timespanSpecifier=consort.TimespanSpecifier(
169 #            minimum_duration=0,
170 #            ),
171 #            ),
172 #        flute=ersilia.shaker_decelerando_music_specifier,
173 #        clarinet=ersilia.shaker_decelerando_music_specifier,
174 #        oboe=ersilia.shaker_decelerando_music_specifier,
175 #        violin=ersilia.shaker_decelerando_music_specifier,

```

```

176 #     viola=ersilia.shaker_decelerando_music_specifier,
177 #     cello=ersilia.shaker_decelerando_music_specifier,
178 #   )
179
180 segment_maker.add_setting(
181   timespan_identifier=[
182     -4, 1,
183     -4, 1,
184     -4, 1,
185     -4, 1,
186     -2, 2,
187   ],
188   timespan_maker=new(
189     ersilia.dense_timespan_maker,
190     fuse_groups=True,
191     silence_talea__denominator=4,
192     timespanSpecifier=consort.TimespanSpecifier(
193       minimum_duration=0,
194     ),
195   ),
196   guitar_pp=ersilia.pitch_pipe_music_specifier,
197   piano_pp=ersilia.pitch_pipe_music_specifier,
198   percussion_pp=ersilia.pitch_pipe_music_specifier,
199   bass_pp=ersilia.pitch_pipe_music_specifier,
200 )
201
202 ### INTERRUPT ###
203
204 segment_maker.add_setting(
205   timespan_identifier=[-1, 1, -1, 1],
206   timespan_maker=consort.BoundaryTimespanMaker(
207     labels=['pitch pipes'],
208     stop_talea=rhythmmakertools.Talea(
209       counts=[2, 9, 3, 1],
210       denominator=4,
211     ),
212   ),
213   percussion=new(
214     ersilia.percussion_snare_interruption_music_specifier,
215     rhythm_maker__first__inciseSpecifier__prefix_talea=[1],
216     rhythm_maker__first__inciseSpecifier__prefix_counts=[1],
217   ),
218   silenced_contexts=segment_maker.score_template.all_voice_names,
219 )
220
221 segment_maker.add_setting(
222   timespan_identifier=timespantools.Timespan(0, (1, 4)),
223   piano_rh=new(
224     ersilia.piano_arm_cluster_music_specifier,
225     attachment_handler__laissez_vibrer=abbreviations.laissez_vibrer,
226     ).transpose(12),
227 )

```

### D.3.2 ERSILIA.SEGMENTS.CUT\_1

```
1 # -*- encoding: utf-8 -*-
2 import ersilia
3 from abjad.tools import timespantools
4
5
6 ### SEGMENT ###
7
8 segment_maker = ersilia.ErsiliaSegmentMaker(
9     desired_duration_in_seconds=6,
10    name='[i]',
11    permitted_time_signatures=ersilia.permitted_time_signatures,
12    repeat=True,
13 )
14
15 ### PEDAL ###
16
17 ### TREMOLO ###
18
19 ### CONTINUO ###
20
21 ### OSTINATO ###
22
23 ### AGITATO ###
24
25 segment_maker.add_setting(
26    timespan_identifier=[1, -3],
27    timespan_maker=ersilia.dense_timespan_maker,
28    flute=ersilia.wind_agitato_music_specifier
29        .rotate(4),
30    clarinet=ersilia.wind_agitato_music_specifier
31        .transpose('C2')
32        .rotate(5),
33    oboe=ersilia.wind_agitato_music_specifier,
34    piano_rh=ersilia.piano_agitato_music_specifier.rotate(-1),
35    piano_lh=ersilia.piano_agitato_music_specifier
36        .transpose(-24),
37    violin=ersilia.string_agitato_music_specifier
38        .rotate(4),
39    viola=ersilia.string_agitato_music_specifier
40        .transpose('C3')
41        .rotate(5),
42    cello=ersilia.string_agitato_music_specifier
43        .transpose('C2')
44        .rotate(6),
45    bass=ersilia.string_agitato_music_specifier
46        .transpose('E1')
47        .rotate(7),
48    )
49
50 segment_maker.add_setting(
51    saxophone=ersilia.saxophone_agitato_music_specifier
52        .transpose('C2')
```

```

53     .rotate(6),
54     guitar=ersilia.guitar_agitato_music_specifier
55     .rotate(1),
56     percussion=ersilia.percussion_tom_fanfare_music_specifier,
57   )
58
59 ### POINTILLIST ###
60
61 ### INTERRUPT ###
62
63 ### AUXILIARY ###
64
65 segment_maker.add_setting(
66   timespan_identifier=timespantools.Timespan(0, (1, 4)),
67   piano_lh=ersilia.piano_arm_cluster_music_specifier
68   .transpose(-12),
69 )

```

### D.3.3 ERSILIA.SEGMENTS.CUT\_2

```

1 # -*- encoding: utf-8 -*-
2 import ersilia
3 from abjad.tools import timespantools
4
5
6 ### SEGMENT ###
7
8 segment_maker = ersilia.ErsiliaSegmentMaker(
9   desired_duration_in_seconds=6,
10  name='[ii]',
11  permitted_time_signatures=ersilia.permitted_time_signatures,
12  repeat=True,
13 )
14
15 ### PEDAL ###
16
17 ### TREMOLO ###
18
19 ### CONTINUO ###
20
21 ### OSTINATO ###
22
23 ### AGITATO ###
24
25 segment_maker.add_setting(
26   percussion=ersilia.percussion_tom_fanfare_music_specifier,
27 )
28
29 segment_maker.add_setting(
30   timespan_identifier=[1, -1],
31   timespan_maker=ersilia.dense_timespan_maker,
32   flute=[
33     ersilia.wind_pointillist_music_specifier.rotate(7),
34     ersilia.wind_agitato_music_specifier.rotate(7),

```

```

35     ],
36     clarinet=[
37         ersilia.wind_agitato_music_specifier.transpose('C2').rotate(8),
38         ersilia.wind_pointillist_music_specifier.transpose('C2').rotate(8),
39     ],
40     oboe:[
41         ersilia.wind_pointillist_music_specifier.rotate(9),
42         ersilia.wind_agitato_music_specifier.rotate(9),
43     ],
44     violin=ersilia.string_agitato_music_specifier
45         .rotate(8),
46     viola=ersilia.string_agitato_music_specifier
47         .transpose('C3')
48         .rotate(9),
49     cello=ersilia.string_agitato_music_specifier
50         .transpose('C2')
51         .rotate(10),
52     bass=ersilia.string_pointillist_music_specifier
53         .transpose('E1')
54         .rotate(11),
55 )
56
57 segment_maker.add_setting(
58     timespan_maker=ersilia.dense_timespan_maker,
59     saxophone=ersilia.saxophone_agitato_music_specifier
60         .transpose('C2'),
61     guitar=ersilia.guitar_agitato_music_specifier
62         .rotate(2),
63     piano_rh=ersilia.piano_palm_cluster_music_specifier,
64     piano_lh=ersilia.piano_agitato_music_specifier
65         .transpose(-24),
66 )
67
68 ### POINTILLIST ###
69
70 ### INTERRUPT ###
71
72 ### AUXILIARY ###

```

#### D.3.4 ERSILIA.SEGMENTS.KOMOKOME

```

1 # -*- encoding: utf-8 -*-
2 import abjad
3 import consort
4 import ersilia
5 from abjad import new
6 from abjad.tools import rhythmmakertools
7
8
9 segment_maker = ersilia.ErsiliaSegmentMaker(
10     desired_duration_in_seconds=120,
11     name='Komokome',
12     permitted_time_signatures=ersilia.permitted_time_signatures,
13     tempo=abjad.Tempo((1, 4), 96),

```

```

14     )
15
16 ### PEDAL ###
17
18 segment_maker.add_setting(
19     timespan_identifier>[
20         1,
21         -1, 1,
22         -2, 1,
23     ],
24     clarinet=ersilia.wind_low_pedal_music_specifier,
25     saxophone=ersilia.wind_low_pedal_music_specifier,
26     percussion=ersilia.percussion_low_pedal_music_specifier,
27     bass=ersilia.string_low_pedal_music_specifier
28     .transpose('E1'),
29 )
30
31 ### TREMOLO ###
32
33 segment_maker.add_setting(
34     timespan_identifier>[
35         -1, 2,
36         -3, 10,
37         -4,
38     ],
39     timespan_maker=new(
40         ersilia.sustained_timespan_maker,
41         fuse_groups=True,
42     ),
43     flute=ersilia.wind_tremolo_music_specifier,
44 )
45
46 segment_maker.add_setting(
47     timespan_maker=new(
48         ersilia.sustained_timespan_maker,
49         fuse_groups=True,
50     ),
51     guitar=ersilia.guitar_tremolo_music_specifier,
52 )
53
54 segment_maker.add_setting(
55     timespan_identifier>[
56         -2, 1,
57         -1, 1,
58         -3, 1,
59         -1, 1,
60         -1, 1,
61         -3, 1,
62         -1, 1,
63     ],
64     timespan_maker=new(
65         ersilia.sparse_timespan_maker,
66         fuse_groups=True,
67     ),

```

```

68     saxophone=ersilia.wind_tremolo_music_specifier
69         .transpose(12),
70     piano_rh=ersilia.piano_tremolo_music_specifier
71         .transpose(24),
72     )
73
74 ### AGITATO ###
75
76 segment_maker.add_setting(
77     timespan_identifier>[
78         -2, 1,
79         -2, 1,
80         -2, 2,
81         -3, 1,
82         -2, 3,
83         -3, 1,
84         -2,
85     ],
86     timespan_maker=new(
87         ersilia.dense_timespan_maker,
88         padding=(1, 4),
89     ),
90     percussion=consort.MusicSpecifierSequence(
91         application_rate='division',
92         music_specifiers:[
93             ersilia.percussion_tom_fanfare_music_specifier,
94             ersilia.percussion_temple_block_fanfare_music_specifier,
95             ersilia.percussion_tom_fanfare_music_specifier,
96             ersilia.percussion_tom_fanfare_music_specifier,
97             ersilia.percussion_temple_block_fanfare_music_specifier,
98             ersilia.percussion_temple_block_fanfare_music_specifier,
99         ]
100    ),
101 )
102
103 segment_maker.add_setting(
104     timespan_identifier>[
105         -1, 1,
106         -2, 2,
107         -3, 1,
108         -2, 2,
109         -3, 4,
110         -4, 1,
111     ],
112     timespan_maker=ersilia.dense_timespan_maker,
113     piano_rh=ersilia.piano_agitato_music_specifier
114         .rotate(1),
115     piano_lh=ersilia.piano_agitato_music_specifier
116         .rotate(2)
117         .transpose(-24),
118     saxophone=ersilia.saxophone_agitato_music_specifier
119         .transpose('C2'),
120     )
121

```

```

122 ### CONTINUO ###
123
124 segment_maker.add_setting(
125     timespan_identifier>[
126         -4, 1,
127     ],
128     timespan_maker=ersilia.sparse_timespan_maker
129         .rotate(1),
130     flute=ersilia.wind_continuo_music_specifier,
131     oboe=ersilia.wind_continuo_music_specifier,
132     clarinet=ersilia.wind_continuo_music_specifier,
133 )
134
135
136 ### POINTILLIST ###
137
138 segment_maker.add_setting(
139     timespan_identifier>[
140         -1, 4,
141         -2, 1,
142         -2, 2,
143         -1,
144     ],
145     timespan_maker=ersilia.sparse_timespan_maker
146         .rotate(1),
147     piano_rh=ersilia.piano_pointillist_music_specifier,
148     piano_lh=ersilia.piano_pointillist_music_specifier
149         .transpose(-12),
150 )
151
152 segment_maker.add_setting(
153     timespan_identifier>[
154         -1, 1,
155         -3, 1,
156         -2, 2,
157         -1, 3,
158         -2, 1,
159         -1, 1,
160         -2,
161     ],
162     timespan_maker=ersilia.dense_timespan_maker,
163     flute=ersilia.wind_pointillist_music_specifier,
164     oboe=ersilia.wind_pointillist_music_specifier
165         .rotate(1),
166     clarinet=ersilia.wind_pointillist_music_specifier
167         .transpose(-12)
168         .rotate(2),
169     piano_rh=ersilia.piano_palm_cluster_music_specifier,
170 )
171
172 segment_maker.add_setting(
173     timespan_identifier>[
174         -3, 1,
175         -2, 1,

```

```

176      -1, 4,
177      -2, 1,
178      -4, 1,
179      -1, 1,
180      -2, 1,
181      ],
182  timespan_maker=new(
183      ersilia.dense_timespan_maker,
184      timespanSpecifier__minimum_duration=(1, 8),
185      ),
186  violin=ersilia.string_pointillist_music_specifier,
187  viola=ersilia.string_pointillist_music_specifier
188      .transpose('C3'),
189  cello=ersilia.string_pointillist_music_specifier
190      .transpose('C2'),
191  bass=ersilia.string_pointillist_music_specifier
192      .transpose('E1'),
193  )
194
195 ### AUXILIARY ###
196
197 music_specifier = consort.MusicSpecifierSequence(
198     application_rate='division',
199     music_specifiers=[
200         ersilia.shaker_spodic_music_specifier,
201         ersilia.shaker_tremolo_music_specifier,
202         ersilia.shaker_spodic_music_specifier,
203         ersilia.shaker_decelerando_music_specifier,
204         ],
205     )
206 timespan_maker = new(
207     ersilia.sparse_timespan_maker,
208     padding=(3, 4),
209     )
210 segment_maker.add_setting(
211     timespan_identifier:[
212         -1, 1,
213         -2, 2,
214         -3, 1,
215         -1, 1,
216         -2, 1,
217         -3, 1,
218         -2,
219         ],
220     timespan_maker=timespan_maker,
221     flute=music_specifier,
222     clarinet=music_specifier,
223     oboe=music_specifier,
224     )
225 segment_maker.add_setting(
226     timespan_identifier:[
227         -2, 1,
228         -3, 1,
229         -1, 1,

```

```

230      -2, 1,
231      -3, 2,
232      -1, 1,
233      -2, 1,
234      -1,
235      ],
236      timespan_maker=timespan_maker,
237      violin=music_specifier,
238      viola=music_specifier,
239      cello=music_specifier,
240      )
241
242 ### CUT THROUGH ###
243
244 segment_maker.add_setting(
245     timespan_identifier>[
246         -8, 1,
247         -13, 1
248     ],
249     timespan_maker=new(
250         ersilia.dense_timespan_maker,
251         ),
252     flute=ersilia.wind_agitato_music_specifier
253         .rotate(1),
254     clarinet=ersilia.wind_agitato_music_specifier.transpose('C2')
255         .rotate(2),
256     oboe=ersilia.wind_agitato_music_specifier
257         .rotate(3),
258     saxophone=ersilia.saxophone_agitato_music_specifier.transpose('C2'),
259     guitar=ersilia.guitar_agitato_music_specifier,
260     piano_rh=ersilia.piano_agitato_music_specifier
261         .rotate(1),
262     piano_lh=ersilia.piano_agitato_music_specifier
263         .transpose(-24)
264         .rotate(2),
265     violin=ersilia.string_agitato_music_specifier
266         .rotate(1),
267     viola=ersilia.string_agitato_music_specifier
268         .transpose('C3')
269         .rotate(2),
270     cello=ersilia.string_agitato_music_specifier
271         .transpose('C2')
272         .rotate(3),
273     bass=ersilia.string_agitato_music_specifier
274         .transpose('E1')
275         .rotate(4),
276     )
277
278 segment_maker.add_setting(
279     timespan_identifier>[
280         -4, 1,
281         -5, 1,
282         -6, 1,
283         -2, 1,

```

```

284     ],
285     timespan_maker=new(
286         ersilia.sparse_timespan_maker,
287         padding=(1, 2),
288         repeat=False,
289     ),
290     percussion=ersilia.percussion_crotales_flash_music_specifier,
291 )
292
293 ### INTERRUPT ###
294
295 segment_maker.add_setting(
296     timespan_identifier>[
297         3, -1,
298         2, -1,
299         1,
300     ],
301     timespan_maker=ersilia.tutti_timespan_maker,
302     piano_lh=ersilia.piano_arm_cluster_music_specifier
303         .transpose(-12),
304     percussion=ersilia.percussion_snare_interruption_music_specifier,
305 )
306
307 segment_maker.add_setting(
308     timespan_maker=consort.BoundaryTimespanMaker(
309         labels='piano arm cluster',
310         output_masks=[
311             rhythmmakertools.SilenceMask(
312                 indices=[0, 1, 3],
313                 period=5,
314             ),
315         ],
316         start_talea=(3, 8),
317     ),
318     guitar=new(
319         ersilia.guitar_strummed_music_specifier,
320         attachment_handler__dynamic_expressions=consort.DynamicExpression(
321             dynamic_tokens='f',
322             only_first=True,
323         ),
324         rhythm_maker__inciseSpecifier__prefix_counts=[3, 2],
325         rhythm_maker__inciseSpecifier__prefix_talea=[1],
326     ),
327 )
328
329 segment_maker.add_setting(
330     timespan_identifier=[-1, 1],
331     timespan_maker=consort.BoundaryTimespanMaker(
332         labels='piano arm cluster',
333         output_masks=[
334             rhythmmakertools.SilenceMask(
335                 indices=[0, 1, 3],
336                 period=5,
337             ),

```

```

338         ],
339         start_talea=(3, 8),
340     ),
341     violin=ersilia.string_overpressure_music_specifier,
342     viola=ersilia.string_overpressure_music_specifier
343         .transpose(-5)
344         .rotate(1),
345     cello=ersilia.string_overpressure_music_specifier
346         .transpose(-17)
347         .rotate(1),
348     )

```

### D.3.5 ERSILIA SEGMENTS.SORT

```

1 # -*- encoding: utf-8 -*-
2 import abjad
3 import consort
4 import ersilia
5 from abjad import new
6 from abjad.tools import rhythmmakertools
7 from abjad.tools import timespantools
8
9
10 segment_maker = ersilia.ErsiliaSegmentMaker(
11     desired_duration_in_seconds=150,
12     name='Sort',
13     permitted_time_signatures=ersilia.permitted_time_signatures,
14     tempo=abjad.Tempo((1, 4), 64),
15 )
16
17 ### PEDAL ###
18
19 segment_maker.add_setting(
20     timespan_identifier=[
21         1,
22         -1, 1,
23         -2, 1,
24     ],
25     timespan_maker=new(
26         ersilia.sustained_timespan_maker,
27         fuse_groups=True,
28         initial_silence_talea=None,
29     ),
30     clarinet=ersilia.wind_low_pedal_music_specifier
31         .transpose(12),
32     saxophone=ersilia.wind_low_pedal_music_specifier
33         .transpose(12),
34     percussion=ersilia.percussion_low_pedal_music_specifier,
35     bass=ersilia.string_low_pedal_music_specifier
36         .transpose('E1'),
37     )
38
39 ### TREMOLO ###
40

```

```

41 segment_maker.add_setting(
42     timespan_maker=new(
43         ersilia.sustained_timespan_maker,
44         fuse_groups=True,
45     ),
46     guitar=ersilia.guitar_undulation_tremolo_music_specifier,
47     piano_rh=ersilia.piano_tremolo_music_specifier,
48 )
49
50 segment_maker.add_setting(
51     timespan_identifier=[-14, 3],
52     timespan_maker=new(
53         ersilia.sustained_timespan_maker,
54         fuse_groups=True,
55     ),
56     percussion=ersilia.percussion_marimba_tremolo_music_specifier,
57 )
58
59 segment_maker.add_setting(
60     timespan_identifier=[
61         -1, 1,
62         -1, 1,
63         -3, 1,
64         -1, 1,
65         -2, 1,
66         -1, 1,
67         -3, 1,
68     ],
69     timespan_maker=new(
70         ersilia.sparse_timespan_maker,
71         fuse_groups=True,
72     ),
73     saxophone=ersilia.wind_tremolo_music_specifier
74         .transpose(12),
75     piano_rh=ersilia.piano_tremolo_music_specifier
76         .transpose(24),
77 )
78
79 segment_maker.add_setting(
80     timespan_maker=new(
81         ersilia.sparse_timespan_maker,
82         silence_talea_denominator=4,
83     ),
84     violin=ersilia.string_tremolo_music_specifier,
85     viola=new(
86         ersilia.string_tremolo_music_specifier,
87         pitch_handler_register_specifier_base_pitch='C3',
88     ),
89     cello=new(
90         ersilia.string_tremolo_music_specifier,
91         pitch_handler_register_specifier_base_pitch='C2',
92     ),
93 )
94

```

```

95  ### AGITATO ###
96
97 segment_maker.add_setting(
98     timespan_identifier=[
99         -2, 1,
100        -1, 1,
101        -2, 3,
102        -2, 2,
103        -1, 3,
104        -2, 1,
105        -2,
106    ],
107    timespan_maker=ersilia.dense_timespan_maker,
108    percussion=ersilia.percussion_marimba_agitato_music_specifier,
109 )
110
111 segment_maker.add_setting(
112     timespan_identifier=[
113         -1, 1,
114         -2, 2,
115         -3, 1,
116         -2, 2,
117         -3, 4,
118         -4, 1,
119    ],
120    timespan_maker=ersilia.dense_timespan_maker,
121    percussion=ersilia.percussion_temple_block_fanfare_music_specifier,
122    piano_rh=ersilia.piano_agitato_music_specifier
123        .rotate(1),
124    piano_lh=ersilia.piano_agitato_music_specifier
125        .rotate(2)
126        .transpose(-24),
127    saxophone=ersilia.saxophone_agitato_music_specifier
128        .transpose('C2'),
129 )
130
131 ### CONTINUO ###
132
133 segment_maker.add_setting(
134     timespan_identifier=[
135         -2, 3,
136         -4, 2,
137         -3, 1,
138    ],
139    timespan_maker=new(
140        ersilia.sparse_timespan_maker,
141        padding=(1, 2),
142    ),
143    percussion=new(
144        ersilia.percussion_crotales_flash_music_specifier,
145        rhythm_maker__inciseSpecifier__prefixCounts=[1],
146    ),
147 )
148

```

```

149 musicSpecifier = consort.MusicSpecifierSequence(
150     application_rate='division',
151     musicSpecifiers=[
152         ersilia.wind_continuo_music_specifier,
153         ersilia.wind_continuo_music_specifier,
154         ersilia.saxophone_agitato_music_specifier.transpose(24),
155         ersilia.wind_continuo_music_specifier,
156         ersilia.saxophone_agitato_music_specifier.transpose(24),
157         ersilia.saxophone_agitato_music_specifier.transpose(24),
158     ],
159 )
160 segment_maker.add_setting(
161     timespanIdentifier=[
162         -9, 1,
163         -4, 1,
164         -2, 3,
165         -1, 1,
166         -4, 2,
167         -2, 3,
168         -8, 1,
169         -2, 3,
170         -4, 2,
171         -3, 1,
172     ],
173     timespanMaker=ersilia.dense_timespan_maker
174         .rotate(1),
175     flute=musicSpecifier,
176     oboe=musicSpecifier.transpose(12),
177     clarinet=musicSpecifier, # .transpose(-12),
178 )
179
180 ### POINTILLIST ####
181
182 segment_maker.add_setting(
183     timespanIdentifier=[
184         -1, 1,
185         -3, 1,
186         -2, 2,
187         -1, 3,
188         -2, 1,
189         -1, 1,
190         -2,
191     ],
192     timespanMaker=ersilia.sparse_timespan_maker,
193     flute=ersilia.wind_pointillist_music_specifier,
194     oboe=new(
195         ersilia.wind_pointillist_music_specifier,
196         pitch_handler__register_specifier__base_pitch='G4',
197     ),
198     clarinet=ersilia.wind_pointillist_music_specifier
199         .transpose(-12)
200         .rotate(2),
201     saxophone=ersilia.wind_pointillist_music_specifier
202         .transpose(-12)

```

```

203     .rotate(2),
204     piano_rh=ersilia.piano_palm_cluster_music_specifier,
205   )
206
207 segment_maker.add_setting(
208   timespan_identifier>[
209     -3, 1,
210     -2, 1,
211     -1, 4,
212     -2, 1,
213     -4, 1,
214     -1, 1,
215     -2, 1,
216   ],
217   timespan_maker=new(
218     ersilia.sparse_timespan_maker,
219     timespanSpecifier__minimum_duration=(1, 8),
220   ),
221   violin=ersilia.string_legato_music_specifier,
222   viola=ersilia.string_legato_music_specifier
223     .transpose('C3'),
224   cello=ersilia.string_legato_music_specifier
225     .transpose('C2'),
226   bass=ersilia.string_legato_music_specifier
227     .transpose('E1'),
228   )
229
230 ### AUXILIARY ###
231
232 music_specifier = new(
233   ersilia.pitch_pipe_music_specifier,
234   rhythm_maker__output_masks=[rhythmmakertools.SustainMask(indices=[0, -1])],
235   attachment_handler__dynamic_expressions=consort.DynamicExpression(
236     start_dynamic_tokens='fp',
237     stop_dynamic_tokens='o',
238   ),
239 )
240 segment_maker.add_setting(
241   timespan_identifier=timespantools.Timespan(0, (3, 2)),
242   timespan_maker=new(
243     ersilia.tutti_timespan_maker,
244     fuse_groups=True,
245     timespan_specifier=consort.TimespanSpecifier(
246       minimum_duration=0,
247     ),
248   ),
249   guitar_pp=music_specifier,
250   piano_pp=music_specifier,
251   percussion_pp=music_specifier,
252   bass_pp=music_specifier,
253 )
254
255 segment_maker.add_setting(
256   timespan_identifier=[

```

```

257     -4, 1,
258     -4, 1,
259     -4, 1,
260     -4, 1,
261     -4,
262     ],
263     timespan_maker=new(
264         ersilia.dense_timespan_maker,
265         fuse_groups=True,
266         silence_talea_denominator=4,
267         timespanSpecifier=consort.TimespanSpecifier(
268             minimum_duration=0,
269             ),
270             ),
271     guitar_pp=music_specifier,
272     piano_pp=music_specifier,
273     percussion_pp=music_specifier,
274     bass_pp=music_specifier,
275     )
276
277 ### INTERRUPT ###
278
279 segment_maker.add_setting(
280     timespan_identifier=timespantools.Timespan(0, (1, 4)),
281     piano_lh=ersilia.piano_arm_cluster_music_specifier
282         .transpose(-12),
283     )
284
285 segment_maker.add_setting(
286     timespan_identifier=[
287         3, -1,
288         2, -1,
289         1,
290         ],
291     timespan_maker=ersilia.sparse_timespan_maker,
292     percussion=ersilia.percussion_bamboo_windchimes_music_specifier,
293     )
294
295 segment_maker.add_setting(
296     timespan_identifier=[
297         -1, 3,
298         -1, 3,
299         ],
300     timespan_maker=ersilia.sparse_timespan_maker,
301     guitar=ersilia.guitar_strummed_music_specifier,
302     )
303
304 segment_maker.add_setting(
305     timespan_identifier=[-1, 1],
306     timespan_maker=consort.BoundaryTimespanMaker(
307         labels='bamboo windchimes',
308         output_masks=[
309             rhythmmakertools.SilenceMask(
310                 indices=[0, 1, 3],

```

```

311         period=5,
312     ),
313     ],
314     start_talea=rhythmmakertools.Talea(
315         counts=[2, 3, 4],
316         denominator=8,
317     ),
318     start_groupings=[3, 4, 3, 2],
319 ),
320 violin=ersilia.string_overpressure_musicSpecifier,
321 viola=ersilia.string_overpressure_musicSpecifier
322 .transpose(7)
323 .rotate(1),
324 cello=ersilia.string_overpressure_musicSpecifier
325 .transpose(-7)
326 .rotate(1),
327 )
328
329 segment_maker.add_setting(
330     timespan_identifier>[
331         -5, 1,
332         -4, 1,
333         -13, 1,
334         -5,
335     ],
336     timespan_maker=new(
337         ersilia.dense_timespan_maker,
338         fuse_groups=True,
339         repeat=False,
340     ),
341     percussion=ersilia.percussion_crotales_interruption_musicSpecifier,
342     silenced_contexts=segment_maker.score_template.all_voice_names,
343 )
344
345 segment_maker.add_setting(
346     timespan_identifier=timespantools.Timespan((321, 8), (325, 8)),
347     percussion=ersilia.percussion_crotales_flash_musicSpecifier,
348     silenced_contexts=segment_maker.score_template.all_voice_names,
349 )

```

## D.4 ERSILIA

### STYLESHEET

SOURCE

#### D.4.1 STYLESHEET.ILY

```

1 \include "scheme.ily"
2
3 #(set-default-paper-size "11x17" 'portrait)
4 #(set-global-staff-size 12)
5
6 \header {
7     composer = \markup {
8         \column {
9             \override #'(font-name . "Didot")
10            \fontsize #3 "Josiah Wolf Oberholtzer (1984)"}

```

```

11      " "
12      }
13  }
14  tagline = \markup { "}
15  title = \markup {
16    \column {
17      \center-align {
18        \override #'(font-name . "Didot Italic")
19        \fontsize #4 {
20          \line { Invisible Cities (iii): }
21        }
22        \vspace #0.5
23        \override #'(font-name . "Didot")
24        \fontsize #16 {
25          \line { ERSILIA }
26        }
27        \vspace #0.5
28        \override #'(font-name . "Didot Italic")
29        \fontsize #2 {
30          \line { ( a botanical survey of the uninhabited northeastern isles ) }
31        }
32        \vspace #1
33        \override #'(font-name . "Didot Italic")
34        \fontsize #4 {
35          \line { for Ensemble Dal Niente }
36          \null
37          \null
38        }
39      }
40    }
41  }
42 }
43
44 \paper {
45
46   indent = 20\mm
47   short-indent = 15\mm
48
49   bottom-margin = 10\mm
50   left-margin = 10\mm
51   right-margin = 10\mm
52   top-margin = 10\mm
53
54   oddHeaderMarkup = \markup {}
55   evenHeaderMarkup = \markup {}
56   oddFooterMarkup = \markup
57     \fill-line {
58       \override #'(font-name . "Didot")
59       \bold \fontsize #3 "Invisible Cities (iii): Ersilia"
60       \override #'(font-name . "Didot")
61       \bold \fontsize #3 \date
62       \concat {
63         \override #'(font-name . "Didot")
64         \bold \fontsize #3

```

```

65          \on-the-fly #print-page-number-check-first
66          \fromproperty #'page:page-number-string
67      }
68  }
69 evenFooterMarkup = \markup
70   \fill-line {
71     \concat {
72       \override #'(font-name . "Didot")
73         \bold \fontsize #3
74         \on-the-fly #print-page-number-check-first
75         \fromproperty #'page:page-number-string
76     }
77     \override #'(font-name . "Didot")
78       \bold \fontsize #3 \date
79     \override #'(font-name . "Didot")
80       \bold \fontsize #3 "Invisible Cities (iii): Ersilia"
81   }
82 print-first-page-number = ##t
83 print-page-number = ##t
84 max-systems-per-page = 1
85 page-breaking = #ly:optimal-breaking
86 ragged-bottom = ##f
87 ragged-last-bottom = ##t
88 markup-system-spacing = #'(
89   (basic-distance . 0)
90   (minimum-distance . 12)
91   (padding . 0)
92   (stretchability . 0)
93 )
94 system-system-spacing = #'(
95   (basic-distance . 12)
96   (minimum-distance . 18)
97   (padding . 12)
98   (stretchability . 20)
99 )
100 top-markup-spacing = #'(
101   (basic-distance . 0)
102   (minimum-distance . 0)
103   (padding . 8)
104   (stretchability . 0)
105 )
106 top-system-spacing = #'(
107   (basic-distance . 0)
108   (minimum-distance . 10)
109   (padding . 0)
110   (stretchability . 0)
111 )
112 }
113
114 \layout {
115   \accidentalStyle modern-cautionary
116   ragged-bottom = ##f
117   ragged-last = ##f
118   ragged-right = ##t

```

```

119
120    %%% ANNOTATIONS %%%
121
122    \context {
123        \Voice
124            \name AnnotatedDivisionsVoice
125            \type Engraver_group
126            \alias Voice
127            \override Accidental.stencil = ##f
128            \override Dots.stencil = ##f
129            \override Flag.stencil = ##f
130            \override NoteCollision.merge-differently-dotted = ##t
131            \override NoteCollision.merge-differently-headed = ##t
132            \override NoteColumn.ignore-collision = ##t
133            \override NoteHead.no-ledgers = ##t
134            \override NoteHead.transparent = ##t
135            \override Stem.stencil = ##f
136            \override TupletBracket.direction = #down
137            \override TupletBracket.outside-staff-padding = 1
138            \override TupletBracket.outside-staff-priority = 999
139            \override TupletBracket.thickness = 2
140            \override TupletNumber.stencil = ##f
141    }
142
143    \context {
144        \Voice
145            \name AnnotatedPhrasesVoice
146            \type Engraver_group
147            \alias Voice
148            \override Accidental.stencil = ##f
149            \override Dots.stencil = ##f
150            \override Flag.stencil = ##f
151            \override NoteCollision.merge-differently-dotted = ##t
152            \override NoteCollision.merge-differently-headed = ##t
153            \override NoteColumn.ignore-collision = ##t
154            \override NoteHead.no-ledgers = ##t
155            \override NoteHead.transparent = ##t
156            \override Stem.stencil = ##f
157            \override TupletBracket.direction = #down
158            \override TupletBracket.outside-staff-padding = 1
159            \override TupletBracket.outside-staff-priority = 1000
160            \override TupletBracket.thickness = 2
161            \override TupletNumber.stencil = ##f
162    }
163
164    %%% DEFAULTS %%%
165
166    \context {
167        \Voice
168        \remove Forbid_line_break_engraver
169    }
170
171    \context {
172        \Staff

```

```

173     \remove Time_signature_engraver
174     \accepts AnnotatedDivisionsVoice
175     \accepts AnnotatedPhrasesVoice
176 }
177
178 \context {
179     \Dynamics
180     \remove Bar_engraver
181     \override DynamicLineSpanner.staff-padding = 11.5
182     \override DynamicText.self-alignment-X = -1
183 }
184
185 %%% TIME SIGNATURE CONTEXT %%%
186
187 \context {
188     \name TimeSignatureContext
189     \type Engraver_group
190     \consists Axis_group_engraver
191     \consists Bar_number_engraver
192     \consists Mark_engraver
193     \consists Metronome_mark_engraver
194     \consists Script_engraver
195     \consists Text_engraver
196     \consists Text_spanner_engraver
197     \consists Time_signature_engraver
198     \override BarNumber.extra-offset = #'(-6 . -4)
199     \override BarNumber.font-name = "Didot Italic"
200     \override BarNumber.font-size = 1
201     \override BarNumber.padding = 4
202
203     \override MetronomeMark.X-extent = #'(0 . 0)
204     \override MetronomeMark.Y-extent = #'(0 . 0)
205     \override MetronomeMark.break-align-symbols = #'(left-edge)
206     \override MetronomeMark.extra-offset = #'(0 . 2)
207     \override MetronomeMark.font-size = 3
208     \override MetronomeMark.use-skylines = ##f
209     \override MetronomeMark.padding = 0
210     \override MetronomeMark.staff-padding = 0
211     \override MetronomeMark.outside-staff-padding = 0
212     \override MetronomeMark.outside-staff-horizontal-padding = 0
213     \override MetronomeMark.minimum-space = 0
214
215     \override RehearsalMark.X-extent = #'(0 . 0)
216     \override RehearsalMark.X-offset = 6
217     \override RehearsalMark.Y-offset = -2.25
218     \override RehearsalMark.break-align-symbols = #'(time-signature)
219     \override RehearsalMark.break-visibility = #end-of-line-invisible
220     \override RehearsalMark.font-name = "Didot"
221     \override RehearsalMark.font-size = 10
222     \override RehearsalMark.outside-staff-priority = 500
223     \override RehearsalMark.self-alignment-X = #center
224     \override Script.extra-offset = #'(4 . -9)
225     \override Script.font-size = 6
226     \override TextScript.font-size = 3

```

```

227     \override TextScript.outside-staff-priority = 600
228     \override TextScript.padding = 6
229
230     \override TextScript.parent-alignment-X = #center
231     \override TextScript.self-alignment-X = #center
232
233     \override TextSpanner.bound-details.right.attach-dir = #LEFT
234     \override TextSpanner.padding = 6.75
235     \override TimeSignature.X-extent = #'(0 . 0)
236     \override TimeSignature.break-align-symbol = #'left-edge
237     \override TimeSignature.break-visibility = #end-of-line-invisible
238     \override TimeSignature.font-size = 3
239     \override TimeSignature.space-alist.clef = #'(extra-space . 0.5)
240     \override TimeSignature.style = #'numbered
241     \override VerticalAxisGroup.default-staff-staff-spacing = #'(
242         (basic-distance . 0)
243         (minimum-distance . 15)
244         (padding . 8)
245         (stretchability . 0)
246     )
247     \override VerticalAxisGroup.minimum-Y-extent = #'(-20 . 20)
248 }
249
250 %%% PERFORMERS %%%
251
252 \context {
253     \Staff
254     \name PitchPipes
255     \type Engraver_group
256     \alias Staff
257     \RemoveEmptyStaves
258     \override StaffSymbol.line-count = 1
259 }
260
261 \context {
262     \Staff
263     \name FluteStaff
264     \type Engraver_group
265     \alias Staff
266 }
267
268 \context {
269     \Staff
270     \name ClarinetStaff
271     \type Engraver_group
272     \alias Staff
273 }
274
275 \context {
276     \Staff
277     \name OboeStaff
278     \type Engraver_group
279     \alias Staff
280 }

```

```

281
282     \context {
283         \Staff
284         \name SaxophoneStaff
285         \type Engraver_group
286         \alias Staff
287     }
288
289     \context {
290         \StaffGroup
291         \name WindSectionStaffGroup
292         \type Engraver_group
293         \alias StaffGroup
294         \accepts FluteStaff
295         \accepts ClarinetStaff
296         \accepts OboeStaff
297         \accepts SaxophoneStaff
298         \override StaffGrouper.staffgroup-staff-spacing = #'(
299             (basic-distance . 0)
300             (minimum-distance . 20)
301             (padding . 15)
302             (stretchability . 10)
303         )
304     }
305
306     \context {
307         \Staff
308         \name GuitarStaff
309         \type Engraver_group
310         \alias Staff
311     }
312
313     \context {
314         \StaffGroup
315         \name GuitarStaffGroup
316         \type Engraver_group
317         \alias Staff
318         \accepts GuitarStaff
319         \accepts PitchPipes
320         systemStartDelimiter = #'SystemStartSquare
321     }
322
323     \context {
324         \Staff
325         \name PianoUpperStaff
326         \type Engraver_group
327         \alias Staff
328     }
329
330     \context {
331         \Staff
332         \name PianoLowerStaff
333         \type Engraver_group
334         \alias Staff

```

```

335     }
336
337     \context{
338         \PianoStaff
339         \remove "Keep_alive_together_engraver"
340         \accepts PianoLowerStaff
341         \accepts PianoUpperStaff
342         \accepts PitchPipes
343     }
344
345     \context {
346         \StaffGroup
347         \name PianoStaffGroup
348         \type Engraver_group
349         \alias StaffGroup
350         \accepts PianoStaff
351         \accepts PitchPipes
352         systemStartDelimiter = #'SystemStartSquare
353     }
354
355     \context {
356         \Staff
357         \name PercussionStaff
358         \type Engraver_group
359         \alias Staff
360     }
361
362     \context {
363         \StaffGroup
364         \name PercussionStaffGroup
365         \type Engraver_group
366         \alias StaffGroup
367         \accepts PercussionStaff
368         \accepts PitchPipes
369         \override StaffGrouper.staffgroup-staff-spacing = #'(
370             (basic-distance . 0)
371             (minimum-distance . 20)
372             (padding . 15)
373             (stretchability . 10)
374         )
375         systemStartDelimiter = #'SystemStartSquare
376     }
377
378     \context {
379         \StaffGroup
380         \name PercussionSectionStaffGroup
381         \type Engraver_group
382         \alias StaffGroup
383         \accepts GuitarStaffGroup
384         \accepts PianoStaffGroup
385         \accepts PercussionStaffGroup
386     }
387
388     \context {

```

```

389     \Staff
390     \name ViolinStaff
391     \type Engraver_group
392     \alias Staff
393 }
394
395 \context {
396     \Staff
397     \name ViolaStaff
398     \type Engraver_group
399     \alias Staff
400 }
401
402 \context {
403     \Staff
404     \name CelloStaff
405     \type Engraver_group
406     \alias Staff
407 }
408
409 \context {
410     \Staff
411     \name ContrabassStaff
412     \type Engraver_group
413     \alias Staff
414 }
415
416 \context {
417     \StaffGroup
418     \name ContrabassStaffGroup
419     \type Engraver_group
420     \alias StaffGroup
421     \accepts ContrabassStaff
422     \accepts PitchPipes
423     systemStartDelimiter = #'SystemStartSquare
424 }
425
426 \context {
427     \StaffGroup
428     \name StringSectionStaffGroup
429     \type Engraver_group
430     \alias StaffGroup
431     \accepts ViolinStaff
432     \accepts ViolaStaff
433     \accepts CelloStaff
434     \accepts ContrabassStaffGroup
435 }
436
437 %%% SCORE %%%
438
439 \context {
440     \Score
441     \accepts PercussionSectionStaffGroup
442     \accepts StringSectionStaffGroup

```

```

443 \accepts TimeSignatureContext
444 \accepts WindSectionStaffGroup
445 \remove Bar_number_engraver
446 \remove Mark_engraver
447 \remove Metronome_mark_engraver
448 \override BarLine.bar-extent = #'(-2 . 2)
449 \override BarLine.hair-thickness = 0.5
450 \override BarLine.space-alist = #'(
451     (time-signature extra-space . 0.0)
452     (custos minimum-space . 0.0)
453     (clef minimum-space . 0.0)
454     (key-signature extra-space . 0.0)
455     (key-cancellation extra-space . 0.0)
456     (first-note fixed-space . 0.0)
457     (next-note semi-fixed-space . 0.0)
458     (right-edge extra-space . 0.0)
459 )
460 \override Beam.beam-thickness = 0.75
461 \%override Beam.direction = #down
462 \override Beam.breakable = ##t
463 \override Beam.damping = 5
464 \override Beam.length-fraction = 1.5
465 \override Glissando.breakable = ##t
466 \override Glissando.thickness = 3
467 \override Hairpin.bound-padding = 1.5
468
469 \override InstrumentName.self-alignment-X = #RIGHT
470 \override MultiMeasureRest.expand-limit = #1
471
472 \override NoteCollision.merge-differently-dotted = ##t
473 \override NoteColumn.ignore-collision = ##t
474 \override OttavaBracket.outside-staff-priority = 500
475 \override OttavaBracket.padding = 2
476 \shape #'((-1.5 . 0) (-1 . 0) (-0.5 . 0) (0 . 0)) RepeatTie
477 \override RepeatTie.X-extent = ##f
478 \override SpanBar.hair-thickness = 0.5
479 \override SpacingSpanner.base-shortest-duration = #(ly:make-moment 1 32)
480 \override SpacingSpanner.strict-grace-spacing = ##f
481 \override SpacingSpanner.strict-note-spacing = ##f
482 \override SpacingSpanner.uniform-stretching = ##t
483 \override StaffSymbol.color = #(x11-color 'grey50)
484 \override StaffSymbol.layer = -1
485 \override Stem.details.beamed-lengths = #'(6)
486 \override Stem.details.lengths = #'(6)
487 \override Stem.stemlet-length = 1.5
488 \override StemTremolo.beam-width = 1.5
489 \override StemTremolo.flag-count = 4
490 \override StemTremolo.slope = 0.5
491 \override SustainPedal.self-alignment-X = #CENTER
492 \override SustainPedalLineSpanner.padding = 2
493 \override SustainPedalLineSpanner.outside-staff-padding = 2
494 \override SustainPedalLineSpanner.to-barline = ##t
495 \override SystemStartSquare.thickness = 2
496

```

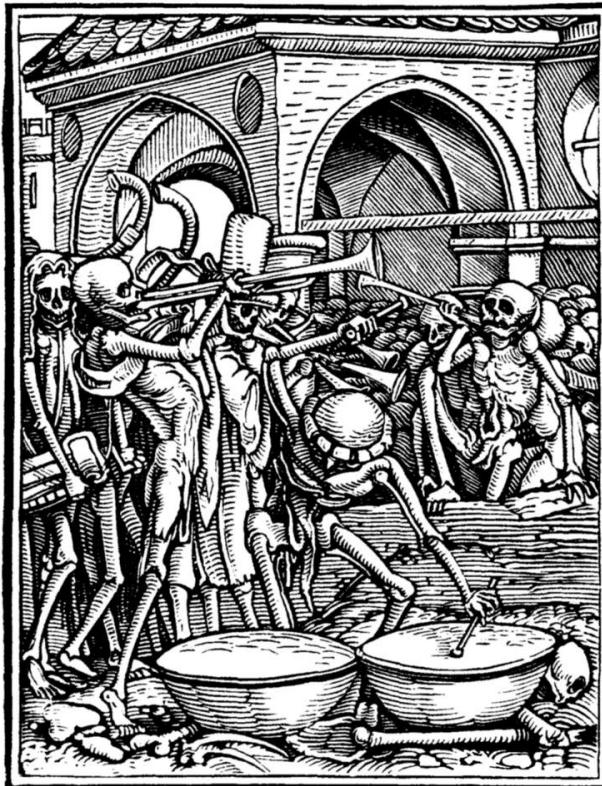
```

497     \override TextSpanner.padding = 1
498     \override TextSpanner.bound-details.right.padding = 2
499
500     \override TrillSpanner.bound-details.right.padding = 1
501
502     \override TupletBracket.breakable = ##t
503     \override TupletBracket.full-length-padding = 1.5
504     \override TupletBracket.full-length-to-extent = ##f
505     \override TupletBracket.padding = 1.5
506     \override TupletBracket.outside-staff-padding = 0.75
507     \override TupletNumber.font-size = 1
508     \override TupletNumber.text = #tuplet-number::calc-fraction-text
509     \override StaffGrouper.staffgroup-staff-spacing = #'(
510         (basic-distance . 10)
511         (minimum-distance . 10)
512         (padding . 5)
513         (stretchability . 0)
514         )
515     \override StaffGrouper.staff-staff-spacing = #'(
516         (basic-distance . 10)
517         (minimum-distance . 10)
518         (padding . 5)
519         (stretchability . 0)
520         )
521     autoBeaming = ##f
522     pedalSustainStyle = #'mixed
523     proportionalNotationDuration = #(ly:make-moment 1 32)
524     tupletFullLength = ##t
525     barNumberFormatter = #format-oval-barnumbers
526 }
527 }
```

# References

- [1] Agostini, A. & Ghisi, D. (2013). Real-time Computer-aided Composition with BACH. *Contemporary Music Review*, 32(1), 41–48.
- [2] Assayag, G., Rueda, C., Laurson, M., Agon, C., & Delerue, O. (1999). Computer-Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3), pp. 59–72.
- [3] Bača, T. (2012). Personal Communication.
- [4] Bača, T., Oberholtzer, J., & Adán, V. (2011). Abjad Reference Manual.
- [5] Bača, T., Oberholtzer, J., Treviño, J., & Adán, V. (2015). Abjad: An open-source software system for formalized score control. Forthcoming.
- [6] Beazley, D. & Jones, B. K. (2013). *Python Cookbook, Third edition*. O'Reilly Media.
- [7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. The MIT Press.
- [8] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.
- [9] Kuuskankare, M. & Laurson, M. (2004). Recent Developments in ENP-score-notation. *Sound and Music Computing*, 4.
- [10] Laurson, M., Kuuskankare, M., & Norilo, V. (2009). An Overview of PWGL, a Visual Programming Environment for Music. *Computer Music Journal*, 33(1), 19–31.
- [11] Loeliger, J. (2009). *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media.
- [12] McCartney, J. (2002). Rethinking the computer music language: SuperCollider. *Computer Music Journal*, 26(4), 61–68.
- [13] Nauert, P. C. (1997). *Timespan formation in nonmetric, posttonal music*. PhD thesis, Columbia University.
- [14] Nienhuys, H.-W. & Nieuwenhuizen, J. (2003). LilyPond, A System for Automated Music Engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*(pp. 167–172).: Citeseer.
- [15] Roads, C. (1996). *The computer music tutorial*. MIT press.
- [16] Roads, C. (2004). *Microsound*. MIT press.
- [17] Taube, H. (1991). Common Music: A Music Composition Language in Common Lisp and CLOS. *Computer Music Journal*, (pp. 21–32).

- [18] Trevino, J. R. (2013). *Compositional and analytic applications of automated music notation via object-oriented programming*. PhD thesis, UCSD.
- [19] Van Rossum, G. & Drake, F. L. (2003). *Python Language Reference Manual*. Network Theory Limited.
- [20] Wishart, T. & Emmerson, S. (1996). *On Sonic Art*. Contemporary music studies. Harwood Academic Publishers.
- [21] Xenakis, I. (1992). *Formalized music: thought and mathematics in composition*. Pendragon Press.



*The Ossuary*  
Hans Holbein  
*The Dance of Death*  
(c.1527)

**H**IS THESIS WAS TYPESET using L<sup>A</sup>T<sub>E</sub>X, originally developed by Leslie Lamport and based on Donald Knuth's T<sub>E</sub>X. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The monospaced text is set in 10 point Inconsolata. The interactive Python interpreter sessions and their graphic output were managed by *abjad-book*, a tool for executing and embedding Python code and music notation in L<sup>A</sup>T<sub>E</sub>X documents. Notation examples were rendered by LilyPond and graph-theoretic examples were rendered by Graphviz.