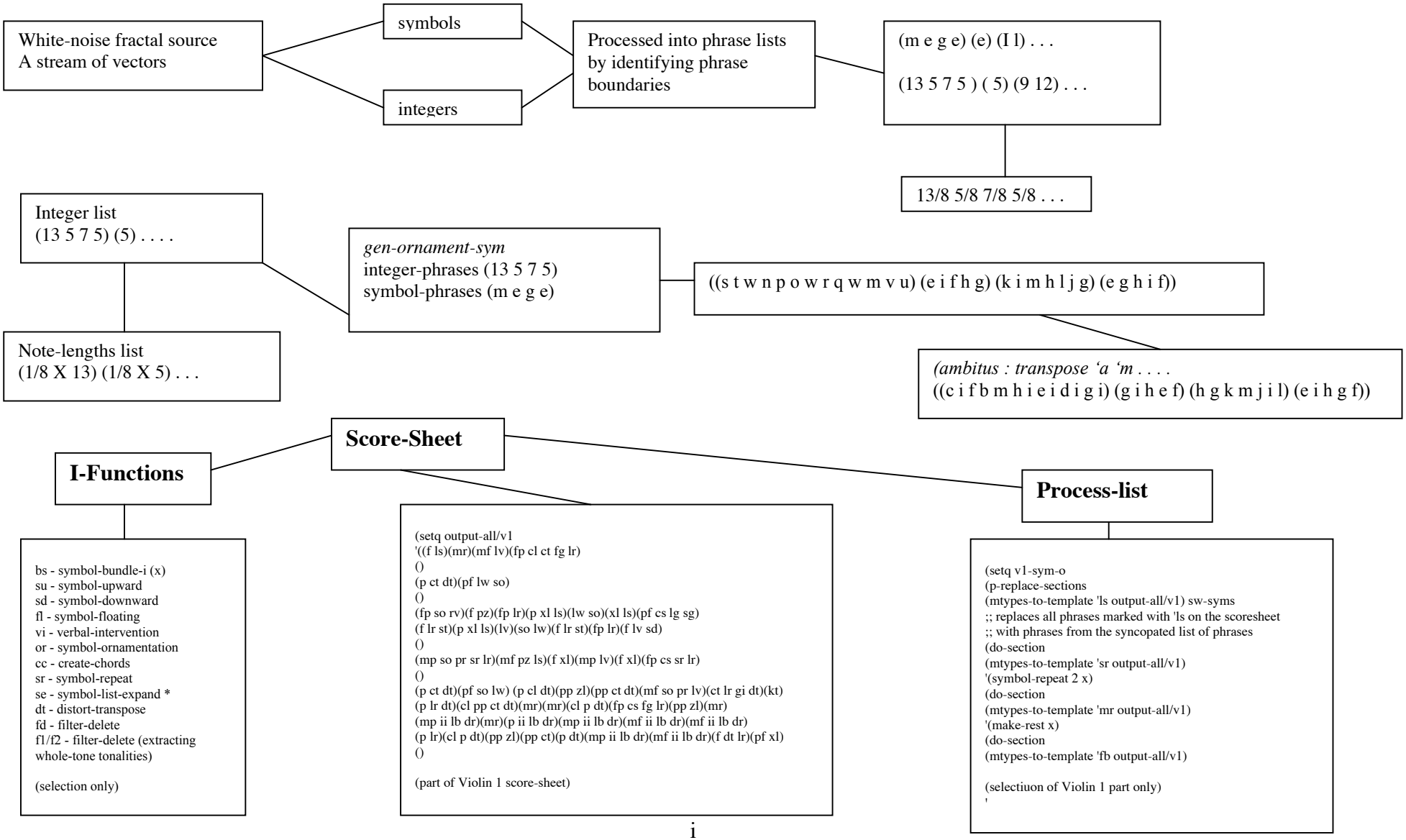
Objects of Curiosity (SuperCity)

For String Quartet and Digital Media

Symbolic Composer Score Files

Nigel Morgan

Objects of Curiosity (SuperCity)



Objects of Curiosity – Introduction to the code annotations

The pages of annotated Symbolic Composer code are prefaced by a sequence of diagrams and images used to illustrate some of the particular features surrounding this composition.

Diagram 1 is flowchart explaining how the various modules through which the composition of the piece passes connect together.

Diagram 2 is a screen shot of Symbolic Composer showing the way a number of files are kept open to enable the composer to edit and process parts of the score more easily. This is a very similar approach to that used in the composition of the *Six Concertos* (2003-6). The only 'window' missing from that used in the Concertos is the timesheet.

Diagram 3 is a scanned image of a printed draft score outline prepared in the early stages of the composition when the structure of the work was in place – but prior to the I-Function processing via the Score-sheet. It was on this 'score' that the composer planned the detailed composition of the music.

When studying the pages of annotation it is important to take into account that the presentation of the code here subsumes four individual files into one. Some of the layout used features columns, which do not occur in Lisp code! This is simply to save space and paper,

The coding of this composition represents one of the most successful attempts by the composer to date to achieve a 'Composing Continuum'. This is where the process of composition follows a seamless and uninterrupted route within a computer environment: from the creation of pre-compositional material and composing tools, through to a notated score and an accurate simulation of a performance. Unlike the *Six Concertos* this score for quartet includes within its code dynamics and instrumental effects (*pizz*, *tremolando* etc).

To find out more about the technical and aesthetic issues surrounding the idea of the 'Composing Continuum' there is a sequence of PowerPoint slides available on the composer's website at this address:

http://www.nigel-morgan.co.uk/files/futurelab.ppt

These slides were created to illustrate a research seminar delivered at the Interdisciplinary Centre for Computer music Research at the Future Music Lab of the University of Plymouth in 2006.

Nigel Morgan 2006

File Edit Lisp Env Def Symbol Solar Mrac Cresc Cadar Major Windows Help 12:23 pm SCOM 4.3

OofC Score 1 {ccl:En

```
;; score-template

(def-tonality
v1 (activate-tonality (chromatic f 5))
v2 (activate-tonality (chromatic f 5))
va (activate-tonality (chromatic f 4))
vc (activate-tonality (chromatic f 3))
)

(def-symbol
  v1 v1-sym-ox
  v2 v2-sym-ox
  va va-sym-ox
  vc vc-sym-ox

)

(def-length
v1 v1-len-o
v2 v2-len-o
va va-len-o
vc vc-len-o
)

(def-duration
v1 stacc-v1
v2 stacc-v2
va stacc-va
vc stacc-vc
)

(def-velocity
v1 v1-dyn-o
v2 v2-dyn-o
va va-dyn-o
vc vc-dyn-o
)

(def-zone
```

SC

I-Functions{HARD/d...

```
;; I-functions 29.10.03

;; to be evaluated only when /
;; 'not' generic functions for

;; current list

#|

calculate-rests
t-replace
t-replace-i

play-registers
symbol-bundle-i
symbol-bundle-x
chord-creator
symbol-upward
symbol-downward
symbol-floating
verbal-intervention
symbol-ornamentation
sequence-transpose
shuffle-pauses
phrase-forward
phrase-backward

symbol-scale
symbol-compress
symbol-fold
symbol-group
```

SC

OofC Scoresheet 1a {ccl:Environment:Run:Instrumentarium:Objects of Curiosity:}

```
;; Scoresheet

(init-rnd 0.5598)

(setq output-all/v1
'((f ls)(mr)(mf lv)(fp cl ct fg lr) ; A
()
(p ct dt)(pf lw so) ; B
()
(fp so rv)(f pz)(fp lr)(p xl ls)(lw so)(xl ls)(pf cs lg sg)
(f lr st)(p xl ls)(lv)(so lw)(f lr st)(fp lr)(f lv sd) ; C
()
(mp so pr sr lr)(mf pz ls)(f xl)(mp lv)(f xl)(fp cs sr lr) ; D
()
(p ct dt)(pf so lw) (p cl dt)(pp zl)(pp ct dt)(mf so pr lv)(ct lr gi dt)(kt)
(p lr dt)(cl pp ct dt)(mr)(mr)(cl p dt)(fp cs fg lr)(pp zl)(mr)
(mp ii lb dr)(mr)(p ii lb dr)(mp ii lb dr)(mf ii lb dr)(mf ii lb dr)
(p lr)(cl p dt)(pp zl)(pp ct)(p dt)(mp ii lb dr)(mf ii lb dr)(f dt lr)(pf xl) ; E
()
(rv pf cs)(lr sr fp so)(pz)(xl ls)(lg sg)(pz)(pf sr lr so)(rv pf cs)
(lr sr fp so)(ct xl)(pf sr st cl lr so)(mr xl)(cl lr so fp)(mr)(mr)(f lv sd) ; F
()
(p f1 lb dr)(pp f2)(p f1 lb dr)(pp so cl)(pp f2)(pp f1)(pp so dt zl) ; G
()
()
(p ls)(p pr); H
()
(fp so rv)(fp lr)(p ct dt)(pf so lv)(pp zl)(pf lv su so) ; I
()
(xl)(xl)(xl)(f)(cs sd sr lr)(pz)(mp ts lb dr )(mp ts lb dr )(p ts sr lr)
(p ts sr lr)(mf ts lb dr)(fp lr)(fp cl lr st)(pp ts sr lr)(pp ts sr lr)(pp ts sr lr)
(pp ts sr lr)(fp rv cl lr st)(mr)(f lv dt)(ff lv su) ; J
()
(so p ls)(cl p)(pf)(p cs f2)(p pz zl)(rv p dt)(mr)(ct pp))) ; K

(setq output-all/v2
'((f ls)(mr)(mf lv)(fp cl ct fg lr)
()
(p ct dt)(pf si so lw)
()
(fp so rv)(f pz)(fp lr)(p xl ls)(lw so)(xl l
(f lr st)(p xl ls)(lv)(lw so)(f lr st)(fp lr
()
```

SC

OofC Process-list 1a {ccl:Environment:Ru...

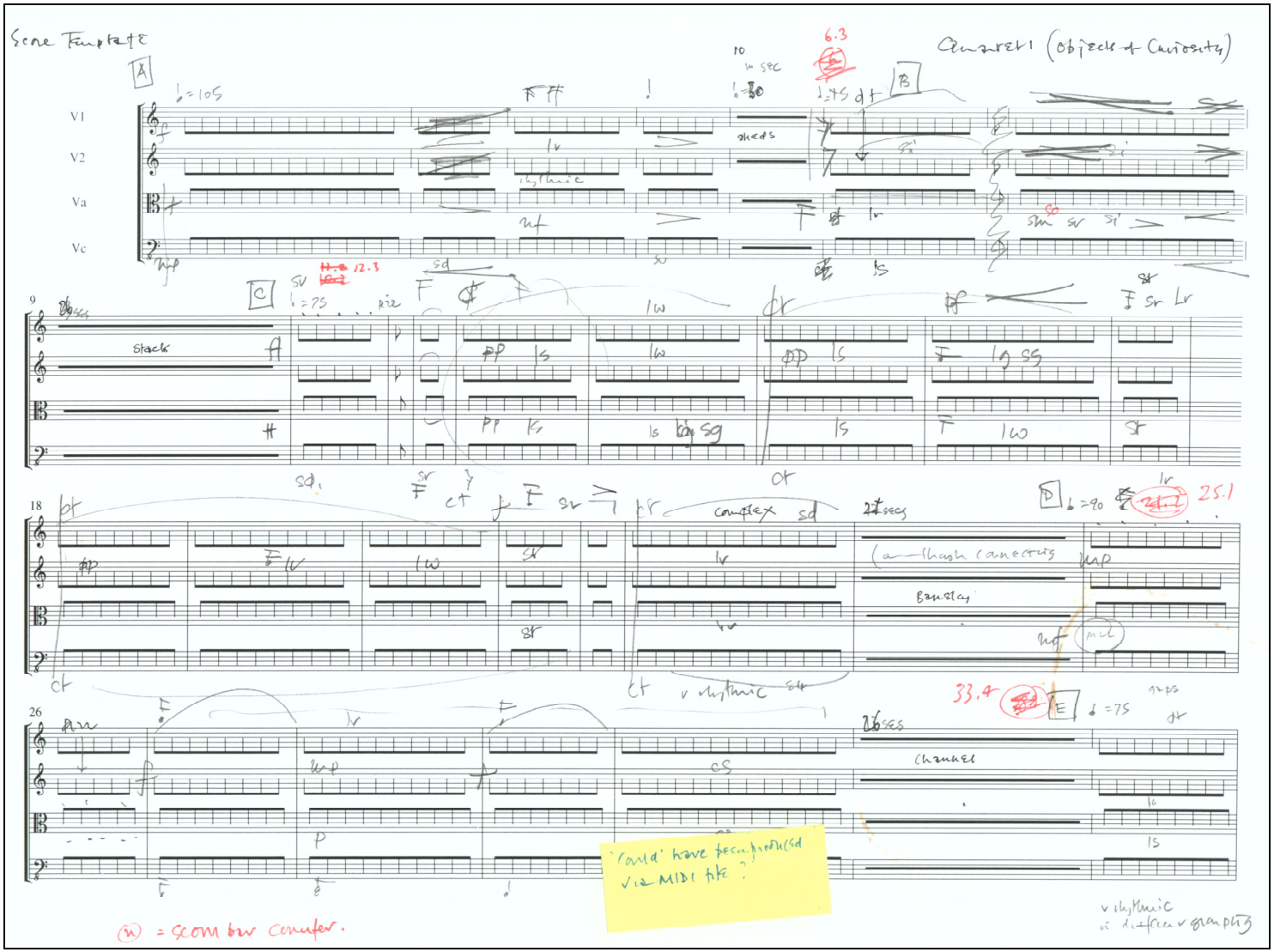
```
(setq v1-sym-o
(p-replace-sections
(mtypes-to-template 'ls output-all/v1) sw-syms
;; replaces all phrases marked with 'ls on the sc
;; with phrases from the syncopated list of phras
(do-section
(mtypes-to-template 'sr output-all/v1)
'(symbol-repeat 2 x)
(do-section
(mtypes-to-template 'mr output-all/v1)
'(make-rest x)
(do-section
(mtypes-to-template 'fb output-all/v1)
'(find-beat x)
(do-section
(mtypes-to-template 'ct output-all/v1)
'(symbol-cut-i 1 2 x nil :start)
(do-section
(mtypes-to-template 'sm output-all/v1)
'(symbol-mask-i x (pick1 nil '(0.1 0.2 0.21 0.22)
```

SC

Listener 1

```
1 >
```

SC


Score Template
Quartet (Objects + Curiosity)
V1
V2
Va
Vc
Stack
Complex
Bandstand
Channel
'Could' have been notated via MIDI file?
(w) = score bar counter.
rhythmic
is different graphing

```
;;; Objects of Curiosity (SuperCity) 29.01.05 - 7.02.05

;; version with object-space moments devised for quartet

;; functions

(defun add1x (pattern) ;; add1 can't be applied or mapcared
(let ((out) (element))
  (dotimes (i (length pattern))
    (setq element
          (add1 (car pattern)))
(setq pattern (cdr pattern))
    (push element out))
  (nreverse out)))

(defun isolate (lis) ; isolates '= symbols
  (if (is-rest (car lis))
    (list (list (car lis)) (cdr lis))
    lis))

(isolate '(= a v c d e))
; ((=) (a v c d e))

(defun phrase-articulation (lis-of-len)
(append (but-last lis-of-len)
(change-length :divide 2 (last lis-of-len) :ratio)))

(phrase-articulation '(1/8 1/8 1/8 1/8 1/8))
;(1/8 1/8 1/8 1/8 1/16)

(defun r-template-list (pattern)
"indentifies lists of pauses and creates a template list"
(prog (out)
loop
(cond ((null pattern) (return out)))
(setq out (append out
                  (list
                   (if  (is-pause-list (car pattern))
                     'x
                     '=))))
(setq pattern (cdr pattern))
(go loop)))

(r-template-list '((a s d e) (= = = =) (s r f t)))
; (= x =)
```

These are specially designed functions for use in this piece only

```
;(include-file "I-functions")

;; symbol/integer-stream

(setq source (gen-noise-white 128 1.0 0.2))

(setq sym  (vector-to-symbol a m source))

(setq int  (add1x (c-symbol-to-integer sym)))

(setq f-int (find-change int) ;; create integer version
      a-int (find-anacrusis int)
)

(setq f-sym (find-change sym)
      a-sym (find-anacrusis sym)
)

(setq f-sym-phrases (flat-them (mapcar 'isolate (create-lists f-sym)))
      ; ((m e g e) (=) (i l) (=) (e a b g i h i d g i g d b l) (=) . . .
      f-int-phrases (symbol-divide (mapcar 'length f-sym-phrases) nil nil int))
      ; ((13 5 7 5) (5) (9 12) (12) (5 1 2 7 9 8 9 4 7 9 7 4 2 12) (12) . . .

(setq f-zone-list (do-section :all '(change-length :times '1/8 x :ratio)  f-int-phrases))
; ((13/8 5/8 7/8 5/8) (5/8) (9/8 3/2) (3/2) . . .

(setq zone-lengths (mapcar 'length f-zone-list))

#| ;; example code

(setq int-x '(13 2 4 6 2)
      sym-x '(m b d f b))

(gen-process '(symbol-repeat x y) int-x '(1/8) :list)
; ((1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8) (1/8 1/8) (1/8 1/8 1/8 1/8) (1/8 1/8 1/8 1/8 1/8 1/8) (1/8 1/8))

(gen-process-list '(symbol-repeat x y) int-x (mapcar 'list sym-x))
; ((m m m m m m m m m m m m m) (b b) (d d d d) (f f f f f f) (b b))

(symbol-divide int-x nil nil (gen-ornament-sym int-x sym-x))
; ((f i g e l c m d c h c k j) (b a) (a c b d) (a d b c e f) (b a))
;     c                        b     d        f             b
|#
```

The I-Functions are a file of specially prepared processors called by a two-letter name. See the Appendix on Page 27 for a key to the full I-Function list. For further information on how the I-Functions work read the SCOM Guide to Composing.

The source material of the Quartet is a stream of vectors generated from white-noise.The output produces 128 samples, which are then converted into symbols and integers.

This function seeks for any repeated symbols and when they occur creates a unique list placing a pause symbol where the repetition takes place

Here is the structure and length of each bar. The single values are (at the moment) silent bars

This example shows how the note-length and symbol structure is devised from a short list of integers and symbols.

This is the key function that generates a unique symbol structure from the list of symbols and integers

```
(setq f-len-phrases (do-section :all '(gen-process '(symbol-repeat x y) x '(1/8) :list) f-int-phrases))
; ((1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8) (1/8 1/8 1/8 1/8 1/8) (1/8 1/8 1/8 1/8 1/8 1/8 1/8) (1/8 1/8 1/8 1/8 1/8))

(setq f-sym-phrases-orn (gen-process-list '(gen-ornament-sym x y) f-int-phrases f-sym-phrases)) ;; slight change to gen-ornament-sym
; ((s w p v n o q w t u r m w e f i h g m h g l j i k h f g e i) (= = = = =)

(setq f-sym-phrases-ornx (gen-process-list '(symbol-divide x nil nil y ) f-int-phrases  f-sym-phrases-orn))
; (((s t w n p o w r q w m v u) (e i f h g) (k i m h l j g) (e g h i f)) ((= = = = =))

(setq f-sym-ph-o (ambitus :transpose 'a 'm (flat-them f-sym-phrases-ornx)))
; ((c i f b m h i e i d i g i) (g i h e f)
(setq f-len-ph-o (flat-them f-len-phrases)
      f-dur-ph-o  (mapcar 'phrase-articulation f-len-ph-o))

#| ;; master-list (see also "OofC Phrase-list 1")
((c i f b m h i e i d i g i) (g i h e f) (h g k m j i l) (e i h g f) ; 4
(= = = = =)
(i c l m j e k d b) (h i e m g b i f i i d c i) ; 2
(= = = = = = = = = = = = =)
(h e g f i) (a) (c b) (j l h g m k i) (m k e c d b l i j) (k h i j l m b c) (m e f l k c d f b j) ; 7
(g d f e) (k m g i j l h) (b l c m k e d j i) (g m l j k i h) (f d e g) (c b) (f h b e m h d c h l h g) ; 7
(= = = = = = = = = = = =)
(b f g d e h c a) (b a c e d) (h d b f g a c e) (c j g d i b b h f e) (g d f b a e c) (e f j e e h i k l g m e e)
(= = = = = = = = = = = = =)
(f d e a b c g h) (e d c h f a i g b) (a c b) (a d c b) (a b e c g d f) (c a b d e) (e b c d a g f) (a g d f b e c h) ;8
(b a) (b a d c) (j h l i k d g f e d d d) (e b h a g d c i f) (b c a) (d k h d f l j d d e g i) (a) (c e f d a b) ;8
(k j f i m e e h e e l e g) (c d a b e) (c f i e c g d h k j c) (l m k e j e f h i e e e g) (d d j f i e g h d d k l) ; 5
(j e i k f c c g c h d)(a b)(b c a)(a)(b a c)(b f e h g d c a) (b e c j b d f g i h)(c f g k h j c d e c i) (a b)(b c h g e a f d); 10
(= = = = = = = =)
(c k c g c d h i e f j) (a b c) (a) (d f b h i g j b e c) (d l j e d f k i h d g d) (a) (b d c a) (b e d c h b f j i g) ; 8
(a c b) (a e b c d) (d c b a) (b e f d a c) (c a b) (c e d c g i k f h j c) (d k g e h d l i d f d j) (a i c g e d b h f);8
 (= = = = = = = = =)
(d f l g h d d e d i k j) (g e a d b c f) (b d e h f a c g) (a c b) (f e a c d h b g) (k e f c g d c c j h i) (b c a) ;7
(= = =) (= = =)
(i e h g f) (d h b h f h m h e c l g) ; 2
(= = = = = = = = = = = =)
(h i g f j k) (c b) (k g f i j h) (l h g m j i k) (d f e g) (c l k m g g d e b f g) ;6
(= = = = = = = = = = =)
(b c) (d e c) (h f g k i j) (h i f g e) (i h b i i i i f c g m d e) (a) (k c m h b l i j) (k f j i h g) (c g i d i h i i i e b m f) ;9
(k h m b c j l i) (d b f h l h e g c h h m) (b c) (d f e g) (e l c g f g b k d g m) (m k j e l d c b i) (b g g f c l g d m k e) ;7
(l d b m c h h e f h g h) (d g f e) (c m j l h i b k) (k l i g j h m) (k m j e i l b d c) ;5
(= = = = = = = = =)
(e c d b a g f h) (d b a c) (c e f a b d) (m e e k i e l e f g h e j) (a) (b g c f a d h e) (a b g d c e f) (c e f b d a));8
```

The ambitus function fits the output of ornamented symbols into a single octave retaining the same pitch class as the original output.

```
|#

;; Scoresheet

; (mapcar 'make-score-sheet  (mapcar 'length   f-int-phrases)) ; creates scoresheet template


(init-rnd 0.5598)

(setq output-all/v1
'((f ls)(mr)(mf lv)(fp cl ct fg lr) ;A
()
(p ct dt)(pf lw so) ;B
()
(fp so rv)(f pz)(fp lr)(p xl ls)(lw so)(xl ls)(pf cs lg sg)
(f lr st)(p xl ls)(lv)(so lw)(f lr st)(fp lr)(f lv sd) ;C
()
(mp so pr sr lr)(mf pz ls)(f xl)(mp lv)(f xl)(fp cs sr lr) ; D
()
(p ct dt)(pf so lw) (p cl dt)(pp zl)(pp ct dt)(mf so pr lv)(ct lr gi dt)(kt)
(p lr dt)(cl pp ct dt)(mr)(mr)(cl p dt)(fp cs fg lr)(pp zl)(mr)
(mp ii lb dr)(mr)(p ii lb dr)(mp ii lb dr)(mf ii lb dr)(mf ii lb dr)
(p lr)(cl p dt)(pp zl)(pp ct)(p dt)(mp ii lb dr)(mf ii lb dr)(f dt lr)(pf xl) ; E
()
(rv pf cs)(lr sr fp so)(pz)(xl ls)(lg sg)(pz)(pf sr lr so)(rv pf cs)
(lr sr fp so)(ct xl)(pf sr st cl lr so)(mr xl)(cl lr so fp)(mr)(mr)(f lv sd) ; F
()
(p f1 lb dr)(pp f2)(p f1 lb dr)(pp so cl)(pp f2)(pp f1)(pp so dt zl) ; G
()
()
(p ls)(p pr) ; H
()
(fp so rv)(fp lr)(p ct dt)(pf so lv)(pp zl)(pf lv su so) ; I
()
(xl)(xl)(xl)(f)(cs sd sr lr)(pz)(mp ts lb dr )(mp ts lb dr )(p ts sr lr)
(p ts sr lr)(mf ts lb dr)(fp lr)(fp cl lr st)(pp ts sr lr)(pp ts sr lr)(pp ts sr lr)
(pp ts sr lr)(fp rv cl lr st)(mr)(f lv dt)(ff lv su) ; J
()
(so p ls)(cl p)(pf)(p cs f2)(p pz zl)(rv p dt)(mr)(ct pp))) ; K
```

This scoresheet for the first violin shows the processing I-Functions applied to each bar of section A. Here is a key:
(f ls) - forte and length-swallow
(mr) - make-rest
(mf lv) - mezzo-forte and length-variate
(fp cl ct fg lr) - descrescendo from f to p, change-length, symbol-cut-i, figurate-symbols and length-repeat.

```
(setq output-all/v2
'((f ls)(mr)(mf lv)(fp cl ct fg lr)
()
(p ct dt)(pf si so lw)
()
(fp so rv)(f pz)(fp lr)(p xl ls)(lw so)(xl ls)(pf cs lg sg)
(f lr st)(p xl ls)(lv)(lw so)(f lr st)(fp lr)(f lv si)
()
(mp sr lr so si)(mf pz ls)(f xl)(mp lv)(f xl)(fp cs sr si lr)
()
(p dt)(pf so lw)(p cl)(pp cl st)(p ct dt)(mf so pr lv)(ct lr gi)(sm lb dr)
(p lr dt si)(cl pp ct su)(mp ii lb dr)(ii lb dr)(rv p st)(fp cs fg lr si)(rv sr pp cl dt)(pp)
(pp kt)(pp)(pp so kt)(pp so kt)(ct so pp)(pp)
(p lr)(rv p st)(rv sr pp cl dt)(pp ct)(p si fg so lr)(mp ij lb dr)(mf il lb dr)(f lr)(pf sd xl)
()
(pf cs)(rv lr sr fp so)(pz)(xl ls)(ts lb dr)(pz)(pf si sr lr so)(rv pf cs)
(rv lr sr fp so)(mr)(pf si st sr cl lr so)(rv xl dt)(cl lr st fp)(lv so)(lv so)(rv dt su fp so)
()
(pp f1)(p f2 lb dr)(pp f1)(pp so cl)(pp si f2)(pp si f1)(pp so si zl)
()
()
(p lr fg)(p so sm sr lr)
()
(fp so rv)(fp lr)(p dt)(pf so rv lv)(pp cl st)(pf lv si so)
()
(xl)(xl)(xl)( si dt)(cs si sr lr)(pz)(p ts sr lr )(p ts sr lr)(mf ts si lb dr)
(mf si ts lb dr)(mf si ts lb dr)(fp lr)(fp cl si lr st)(pp ts sr lr)(pp ts sr lr)(pp ts sr lr)
(pp ts sr lr)(fp rv si cl lr st)(mr)(f lv)(ff dt lv)
()
(so p ls si)(cl si p)(si pf)(p cs f2)(p pz zl)(rv p si dt)(mr)(ct pp)))
```

```
(setq output-all/va
'((f lv)(fp su)(mf lv)(fp cl pr su sr)
()
(mp lr fg)(mp so sm sr lr)
()
(fp so)(f pz)(fp lr)(p xl ls rv)(cs lg sg)(p xl ls rv)(pf lv)
(rv f lr st)(p xl ls rv)(cs lg sg)(cs lg sg)(rv f lr st)(fp lr)(f rv lv)
()
(mf pz si ls)(f sr lr so si)(ct f xl)(p ts si)(ct f xl)(fp si cs sr lr)
()
(mp lr fg)(mp sm sr lr)(p cl si)(rv pp cl st)(ct p so gc lr)(mf si so pr lv)(mr)(ct st lr)
(p ct lr)(cl pp ct pr)(p so ik)(p so il)(p st)(fp cs fg lr si)(pp sr cl dt lr)(pp)
(pp kt)(pp)(pp so kt)(pp so kt)(ct so pp)(pp)
(p lr)(p st)(pp sr cl dt lr)(mr)(mr)(ct p)(mp fg lr si)(f si lr)(xl dt si pf)
()
(rv pf cs)(rv si fp so)(pz)(xl si ls)(ts lb dr)(pz)(pf si so)(rv pf cs)
(rv si fp so)(mr)(cl ct pf gc so)(xl dt)(cl si lr st fp)(lw so)(lw si so) (f rv lv so)
()
(pp si f1)(pp si f2)(pp si f1)(si pp so cl)(p f2 lb dr)(pp f1)(pp so sd zl)
()
()
(p ct dt)(p si so lw)
()
(fp so)(fp lr)(mp lr fg)(mp sm sr lr)(rv pp cl st)(pf rv lv so)
()
(xl)(xl)(xl)(rv)(cs dt sd sr lr)(pz)(pp ts si sr lr)(pp ts si sr lr)(p ts sr lr)
(p ts sr lr)(mp ts su sr lr)(fp lr)(ct p lr sr cl)(mp ts lb dr)(mp ts lb dr)(pp ts si sr lr)
(pp ts si sr lr)(ct p lr sr cl)(mf lv)(f lv dt)(ff lv si)
()
(f lv so)(p dt cl)(dt pf )(p cs f1)(p pz zl cc)(mr)(rv p si dt)(ct pp)))
```

```
(setq output-all/vc
'((p)(pf sd)(mf lv)(fp cl sr sd pr)
()
(mp ls)(p pr)
()
(fp so)(f)(fp lr)(p xl ls rv)(cs lg sg)(p xl ls rv)(pf lv)
(rv f lr st)(p xl ls rv)(cs lg sg)(cs lg sg)(rv f lr st)(fp lr)(f pr su lv)
()
(mf pz ls)(f so sr lr)(ct f x)(p ts)(ct f xl)(fp cs sr lr)
()
(mp so ls)(p so pr)(p cl)(rv pp zl)(ct p so gc lr)(mf si so pr lv)(mr)(ct st lr)
(p ct lr)(cl pp ct pr)(mr)(p so ij)(p cl si)(fp cs fg lr)(pp zl)(mr)
(p ik so)(mr)(mr)(pp il so)(p il so)(mp il su so)
(p lr)(p cl si)(pp zl)(mr)(mr)(ct mp)(mf)(f lr)(xl pf)
()
(pf cs)(rv dt su fp so)(pz)(xl si ls)(ts lb dr)(pz)(pf dt su so)(rv pf cs)
(rv dt su fp so)(ct xl dt)(ct pf gc so)(xl mr)(rv dt su cl lr fp so)(mr)(mr)(f pr su lv)
()
(pp sd f1)(pp su f2)(pp sd f1)(sd pp dt so cl)(pp f2)(p f1 lb)(pp so su dt zl)
()
()
(pp ct dt)(p lw so)
()
(fp so)(fp lr)(mp so ls)(p so pr)(rv pp zl)(pf pr sd so lv)
()
(xl)(xl)(xl)()(cs su sr lr)(pz)(pp ts sr lr)(pp ts sr lr)(pp ts sr lr)
(p ts si sr lr)(mp sd ts sr lr)(fp lr)(ct p sr lr cl)(pp ts sr lr)(pp ts sr lr)(mp si ts lb dr)
(mp si ts lb dr)(ct p lr sr cl)(mf lv)(f lv dt)(ff lv sd)
()
(so p su ls)(p cl)(pf)(p cs f1)(p pz zl cc)(mr)(rv p dt)(ct pp)))
```

The preliminary Process-lists begin here but the main part of the list collection begins on page 10. The Process-list section of the score-file is usually contained in a separate file and kept open on SCOM's desktop.

```
;; Process-list

 (init-rnd 0.5598)

(setq sw-syms
(gen-process-list '(length-swallow x y)
 f-len-ph-o f-sym-ph-o))

;; produces a complete syncopation (in symbols and lengths)
;; of symbol list


(def-neuron p-to-v ;; sets dynamic scheme for 'object-space moments'
(in 1 '= ) '25
(otherwise (in 1 0)))

(defun pause-to-vel (&rest lis)
(apply 'run-neuron  'p-to-v lis))

(setq set-dyn
(mapcar 'pause-to-vel
(do-section :all '(fill-template x '64)
 f-sym-ph-o)))
```

Section A consists of 4 bars generared from the symbols (m e g e). This material will form the basis of any music happening in the pause bars created by the function create-lists. These symbols and their corresponding note-lengths have to be inserted into what was originally pause bars. The first pause bar become Objects of Curiosity A . . . and so on

```
;; inserting the object-of-curiosity material

(setq p-list (delete 'nil (do-section :all '(filter-delete '(=) x) f-sym-phrases)))
; ((m e g e) (i m) (e a b g i h j d g i g d b l) (h e h j g m)
(setq s-list '(5 13 12 13 8 9 3 3 12 11 9)) ; list of space lengths (1/4 @ mm 30)

(init-rnd 0.5598)

(setq v1-rhy (mapcar (function (lambda (x) (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 x))))) s-list)
      v2-rhy (mapcar (function (lambda (x) (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 x))))) s-list)
      va-rhy (mapcar (function (lambda (x) (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 x))))) s-list)
      vc-rhy (mapcar (function (lambda (x) (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 x))))) s-list))

; v1-rhy '((-1/8 1/8 -1/4 1/4 1/8 1/8 -1/4) (-1/8 1/8 1/8 1/8 -1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/4 -1/4) . . .

(init-rnd 0.5598)
```

Split-to-parts takes a list of symbols and splits it across the four instruments of the quartet. The output examples here show Objects of Curiosity A and B

```
(setq parts-list
      (gen-process-list
       '(split-to-parts nil x
         (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 y)))
         (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 y)))
         (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 y)))
         (l-rest-range '(0 1) (length-variate nil 2 2 (build-list '1/4 y))))
       p-list s-list))

; (((m e e e) (g m g e m) (e e e e) (m g m g)) ((m m i m m m i m m m m i i)

(init-rnd 0.5598)

(setq v1-part (mapcar 'first parts-list) ; ((m e e e) (m m i m m m i m m m m i i)
      v2-part (mapcar 'second parts-list) ; ((g m g e m) (i i m i i m i i m i i m)
      va-part (mapcar 'third parts-list)  ; ((e e e e) (m m m i m i i m m m i m)
      vc-part (mapcar 'fourth parts-list)) ;((m g m g) (i i i i m i i m i i m i)

(setq quartet-space (r-template-list f-sym-ph-o))
; (= = = = x = = x = = = = = . . .

(setq f-sym-ph-ov1 (p-replace nil (e-position 'x quartet-space) v1-part f-sym-ph-o)
      f-sym-ph-ov2 (p-replace nil (e-position 'x quartet-space) v2-part f-sym-ph-o)
      f-sym-ph-ova (p-replace nil (e-position 'x quartet-space) va-part f-sym-ph-o)
      f-sym-ph-ovc (p-replace nil (e-position 'x quartet-space) vc-part f-sym-ph-o))

; ((c k f b m h k e k d i g j) (g i h e f) (h g k m j i l) (e i h g f) (m e e e) . . .

(setq f-len-ph-ov1 (p-replace nil (e-position 'x quartet-space) v1-rhy f-len-ph-o)
      f-len-ph-ov2 (p-replace nil (e-position 'x quartet-space) v2-rhy f-len-ph-o)
      f-len-ph-ova (p-replace nil (e-position 'x quartet-space) va-rhy f-len-ph-o)
      f-len-ph-ovc (p-replace nil (e-position 'x quartet-space) vc-rhy f-len-ph-o))

; ((1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8) (1/8 1/8 1/8 1/8 1/8) (1/8 1/8 1/8 1/8 1/8 1/8 1/8)
; (1/8 1/8 1/8 1/8 1/8) (-1/8 1/8 -1/4 1/4 1/8 1/8 -1/4)
```

r-template-list was created specially for this score-file to produce a template showing where the pause symbols are situated in the phrase list. The template is then use to position the new material in the pause lists.

Here begins the Process-Lists for each instrument. The lists include all the I-Functions and their two letter triggers.

```
(setq v1-sym-o
(p-replace-sections
(mtypes-to-template 'ls output-all/v1) sw-syms
;; replaces all phrases marked with 'ls on the scoresheet
;; with phrases from the syncopated list of phrases
(do-section
(mtypes-to-template 'sr output-all/v1)
'(symbol-repeat 2 x)
(do-section
(mtypes-to-template 'mr output-all/v1)
'(make-rest x)
(do-section
(mtypes-to-template 'fb output-all/v1)
'(find-beat x)
(do-section
(mtypes-to-template 'ct output-all/v1)
'(symbol-cut-i 1 2 x nil :start)
(do-section
(mtypes-to-template 'sm output-all/v1)
'(symbol-mask-i x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'ts output-all/v1)
'(symbol-thin 2 5 x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'cs output-all/v1)
'(symbol-contract-i 4 8 x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'st output-all/v1)
'(sequence-transpose x)
(do-section
(mtypes-to-template 'se output-all/v1)
'(symbol-list-expand x)
(do-section
(mtypes-to-template 'dt output-all/v1)
'(distort-transpose 1 x)
(do-section
(mtypes-to-template 'sh output-all/v1)
'(symbol-harmonize nil 'mix -6 6  x)
(do-section
(mtypes-to-template 'si output-all/v1)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/v1)
'(chord-creator (get-random 3 7) '2  x)
```

The lists are laid out in two columns to enable the reader to see more material at a glance. A Lisp scorefile does not allow this layout!

```
(do-section
 (mtypes-to-template 'fg output-all/v1)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/v1)
  '(play-registers (pick1 nil '(2 3 4)) '(-7 12 -6 6 -4 4) x)
  (do-section
 (mtypes-to-template 'bs output-all/v1)
 '(symbol-bundle-x 2 x nil)
  (do-section
 (mtypes-to-template 'su output-all/v1)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/v1)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/v1)
 '(symbol-floating x)
(do-section
(mtypes-to-template 'gp output-all/v1)
'(gen-palindrome x)
(do-section
(mtypes-to-template 'gc output-all/v1)
'(g-coda (length x) x)
(do-section
(mtypes-to-template 'gi output-all/v1)
'(g-intro (length x) x)
(do-section
(mtypes-to-template 'rv output-all/v1)
'(symbol-retrograde x)
(do-section
(mtypes-to-template 'ii output-all/v1)
'(interval-edit-i x)
(do-section
(mtypes-to-template 'ij output-all/v1)
'(interval-edit-j x)
(do-section
(mtypes-to-template 'ik output-all/v1)
'(interval-edit-k x)
(do-section
(mtypes-to-template 'il output-all/v1)
'(interval-edit-l x)
(do-section
(mtypes-to-template 'f1 output-all/v1)
'(filter-delete wt1 x)
```

Length and Dynamic Process-Lists

```
(do-section
(mtypes-to-template 'f2 output-all/v1)
'(filter-delete wt2 x)
(do-section (mtypes-to-template 'or output-all/v1) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/v1)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 f-sym-ph-ov1)))))))))))))))))))))))))))))))))))))

(init-rnd 0.5598)

(setq v1-len-o
(p-replace-sections (mtypes-to-template 'lb output-all/v1)
(do-section (mtypes-to-template 'lb output-all/v1)
 '(length-bundle x) v1-sym-o)
(p-replace-sections (mtypes-to-template 'lg output-all/v1)
(do-section (mtypes-to-template 'lg output-all/v1)
 '(length-group x) v1-sym-o)
(do-section (mtypes-to-template 'gp output-all/v1)
'(gen-palindrome  x)
(do-section (mtypes-to-template 'lw output-all/v1)
'(length-variate nil (get-random 3 5) 4 x)
(do-section (mtypes-to-template 'lv output-all/v1)
'(length-variate nil (get-random 3 5) 2 x)
(do-section (mtypes-to-template 'xl output-all/v1)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'yl output-all/v1)
'(change-length :divide 4 x :ratio)
(do-section (mtypes-to-template 'cl output-all/v1)
'(change-length :times 2 x :ratio)
 (do-section (mtypes-to-template 'zl output-all/v1)
'(change-length :times 4 x :ratio)
(do-section (mtypes-to-template 'lr output-all/v1) ; length-
repeat
'(length-repeat 2 x)
; (p-replace-sections (mtypes-to-template 'or output-all/v1)
;(do-section (mtypes-to-template 'or output-all/v1) '(flatten
x)
;(do-section
;(mtypes-to-template 'or output-all/v1)
; '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
; f-sym-ph-o ))
f-len-ph-ov1)))))))))))

(init-rnd 0.5598)
```

```
(setq v1-dyn-o
(do-section (mtypes-to-template 'pp output-all/v1)
'(dyn-pp 25  x)
(do-section (mtypes-to-template 'p output-all/v1)
'(dyn-p 45  x)
(do-section (mtypes-to-template 'mp output-all/v1)
'(dyn-mp 60  x)
(do-section (mtypes-to-template 'mf output-all/v1)
'(dyn-mf 75  x)
(do-section (mtypes-to-template 'f output-all/v1)
'(dyn-f 96 x)
(do-section (mtypes-to-template 'ff output-all/v1)
'(dyn-ff 115 x)
(do-section (mtypes-to-template 'pf output-all/v1)
'(dyn-pf 45 96 x)
(do-section (mtypes-to-template 'fp output-all/v1)
'(dyn-fp 96 45 x)
(do-section (mtypes-to-template 'rd output-all/v1)
'(dyn-rnd 45 96 x)
set-dyn)))))))))))

(init-rnd 0.5598)

(setq v2-sym-o
(p-replace-sections
(mtypes-to-template 'ls output-all/v2) sw-syms
(do-section
(mtypes-to-template 'sr output-all/v2)
'(symbol-repeat 2 x)
(do-section
(mtypes-to-template 'mr output-all/v2)
'(make-rest x)
(do-section
(mtypes-to-template 'fb output-all/v2)
'(find-beat x)
(do-section
(mtypes-to-template 'ct output-all/v2)
'(symbol-cut-i 1 2 x nil :start)
(do-section
(mtypes-to-template 'sm output-all/v2)
'(symbol-mask-i x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'ts output-all/v2)
'(symbol-thin 2 5 x (pick1 nil '(0.1 0.2 0.21 0.22)))
```

Notice the same seed is used between every expression that contains a random seed setting nil. By setting the same seed throughout the output remains constant no matter how many times the score-file is evaluated

```
(do-section
(mtypes-to-template 'cs output-all/v2)
'(symbol-contract-i 4 8 x (pick1 nil '(0.3 0.4 0.51 0.62)))
(do-section
(mtypes-to-template 'st output-all/v2)
'(sequence-transpose x)
(do-section
(mtypes-to-template 'se output-all/v2)
'(symbol-list-expand x)
(do-section
(mtypes-to-template 'dt output-all/v2)
'(distort-transpose 1 x)
(do-section
(mtypes-to-template 'sh output-all/v2)
'(symbol-harmonize nil 'mix -6 6  x)
(do-section
(mtypes-to-template 'si output-all/v2)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/v2)
'(chord-creator (get-random 3 7) '2  x)
(do-section
 (mtypes-to-template 'fg output-all/v2)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/v2)
  '(play-registers (pick1 nil '(2 3 4)) '(-7 12 -6 6 -4 4) x)
  (do-section
 (mtypes-to-template 'bs output-all/v2)
 '(symbol-bundle-x 2 x nil)
  (do-section
 (mtypes-to-template 'su output-all/v2)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/v2)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/v2)
 '(symbol-floating x)
(do-section
(mtypes-to-template 'gp output-all/v2)
'(gen-palindrome x)
(do-section
(mtypes-to-template 'gc output-all/v2)
'(g-coda (length x) x)
(do-section
(mtypes-to-template 'gi output-all/v2)
'(g-intro (length x) x)
(do-section
(mtypes-to-template 'rv output-all/v2)
'(symbol-retrograde x)
(do-section
(mtypes-to-template 'ii output-all/v2)
'(interval-edit-i x)
(do-section
(mtypes-to-template 'ij output-all/v2)
'(interval-edit-j x)
(do-section
(mtypes-to-template 'ik output-all/v2)
'(interval-edit-k x)
(do-section
(mtypes-to-template 'il output-all/v2)
'(interval-edit-l x)
(do-section
(mtypes-to-template 'f1 output-all/v2)
'(filter-delete wt1 x)
(do-section
(mtypes-to-template 'f2 output-all/v2)
'(filter-delete wt2 x)
(do-section (mtypes-to-template 'or output-all/v2) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/v2)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 f-sym-ph-ov2))))))))))))))))))))))))))))))))))))))

(init-rnd 0.5598)

(setq v2-len-o
(p-replace-sections (mtypes-to-template 'lb output-all/v2)
(do-section (mtypes-to-template 'lb output-all/v2)
 '(length-bundle x) v2-sym-o)
(p-replace-sections (mtypes-to-template 'lg output-all/v2)
(do-section (mtypes-to-template 'lg output-all/v2)
 '(length-group x) v1-sym-o)
(do-section (mtypes-to-template 'gp output-all/v2)
'(gen-palindrome  x)
(do-section (mtypes-to-template 'lw output-all/v2)
'(length-variate nil (get-random 3 5) 4 x)
(do-section (mtypes-to-template 'lv output-all/v2)
'(length-variate nil (get-random 3 5) 2 x)
```

```
(do-section (mtypes-to-template 'xl output-all/v2)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'yl output-all/v2)
'(change-length :divide 4 x :ratio)
(do-section (mtypes-to-template 'cl output-all/v2)
'(change-length :times 2 x :ratio)
 (do-section (mtypes-to-template 'zl output-all/v2)
'(change-length :times 4 x :ratio)
(do-section (mtypes-to-template 'lr output-all/v2) ; length-
repeat
'(length-repeat 2 x)
; (p-replace-sections (mtypes-to-template 'or output-all/v2)
;(do-section (mtypes-to-template 'or output-all/v2) '(flatten
x)
;(do-section
;(mtypes-to-template 'or output-all/v2)
; '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
; f-sym-ph-o ))
f-len-ph-ov2)))))))))))

(init-rnd 0.5598)

(setq v2-dyn-o
(do-section (mtypes-to-template 'pp output-all/v2)
'(dyn-pp 25  x)
(do-section (mtypes-to-template 'p output-all/v2)
'(dyn-p 45  x)
(do-section (mtypes-to-template 'mp output-all/v2)
'(dyn-mp 60  x)
(do-section (mtypes-to-template 'mf output-all/v2)
'(dyn-mf 75  x)
(do-section (mtypes-to-template 'f output-all/v2)
'(dyn-f 96 x)
(do-section (mtypes-to-template 'ff output-all/v2)
'(dyn-ff 115 x)
(do-section (mtypes-to-template 'pf output-all/v2)
'(dyn-pf 45 96 x)
(do-section (mtypes-to-template 'fp output-all/v2)
'(dyn-fp 96 45 x)
(do-section (mtypes-to-template 'rd output-all/v2)
'(dyn-rnd 45 96 x)
set-dyn)))))))))))


(init-rnd 0.5598)
```

```
(setq va-sym-o
(p-replace-sections
(mtypes-to-template 'ls output-all/va) sw-syms
(do-section
(mtypes-to-template 'sr output-all/va)
'(symbol-repeat 2 x)
(do-section
(mtypes-to-template 'mr output-all/va)
'(make-rest x)
(do-section
(mtypes-to-template 'fb output-all/va)
'(find-beat x)
(do-section
(mtypes-to-template 'ct output-all/va)
'(symbol-cut-i 1 2 x nil :start)
(do-section
(mtypes-to-template 'sm output-all/va)
'(symbol-mask-i x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'ts output-all/va)
'(symbol-thin 2 5 x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'cs output-all/va)
'(symbol-contract-i 4 8 x (pick1 nil '(0.6 0.7 0.81 0.92)))
(do-section
(mtypes-to-template 'st output-all/va)
'(sequence-transpose x)
(do-section
(mtypes-to-template 'se output-all/va)
'(symbol-list-expand x)
(do-section
(mtypes-to-template 'dt output-all/va)
'(distort-transpose 1 x)
(do-section
(mtypes-to-template 'sh output-all/va)
'(symbol-harmonize nil 'mix -6 6  x)
(do-section
(mtypes-to-template 'si output-all/va)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/va)
'(chord-creator (get-random 3 7) '2  x)
(do-section
 (mtypes-to-template 'fg output-all/va)
```

```
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/va)
  '(play-registers (pick1 nil '(2 3 4)) '(-7 12 -6 6 -4 4) x)
  (do-section
 (mtypes-to-template 'bs output-all/va)
 '(symbol-bundle-x 2 x nil)
  (do-section
 (mtypes-to-template 'su output-all/va)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/va)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/va)
 '(symbol-floating x)
(do-section
(mtypes-to-template 'gp output-all/va)
'(gen-palindrome x)
(do-section
(mtypes-to-template 'gc output-all/va)
'(g-coda (length x) x)
(do-section
(mtypes-to-template 'gi output-all/va)
'(g-intro (length x) x)
(do-section
(mtypes-to-template 'rv output-all/va)
'(symbol-retrograde x)
(do-section
(mtypes-to-template 'ii output-all/va)
'(interval-edit-i x)
(do-section
(mtypes-to-template 'ij output-all/va)
'(interval-edit-j x)
(do-section
(mtypes-to-template 'ik output-all/va)
'(interval-edit-k x)
(do-section
(mtypes-to-template 'il output-all/va)
'(interval-edit-l x)
(do-section
(mtypes-to-template 'f1 output-all/va)
'(filter-delete wt1 x)
(do-section
(mtypes-to-template 'f2 output-all/va)
'(filter-delete wt2 x)
(do-section (mtypes-to-template 'or output-all/va) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/va)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 f-sym-ph-ova))))))))))))))))))))))))))))))))))))

(init-rnd 0.5598)

(setq va-len-o
(p-replace-sections (mtypes-to-template 'lb output-all/va)
(do-section (mtypes-to-template 'lb output-all/va)
 '(length-bundle x) va-sym-o)
(p-replace-sections (mtypes-to-template 'lg output-all/va)
(do-section (mtypes-to-template 'lg output-all/va)
 '(length-group x) v1-sym-o)
(do-section (mtypes-to-template 'gp output-all/va)
'(gen-palindrome  x)
(do-section (mtypes-to-template 'lw output-all/va)
'(length-variate nil (get-random 3 5) 4 x)
(do-section (mtypes-to-template 'lv output-all/va)
'(length-variate nil (get-random 3 5) 2 x)
(do-section (mtypes-to-template 'xl output-all/va)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'yl output-all/va)
'(change-length :divide 4 x :ratio)
(do-section (mtypes-to-template 'cl output-all/va)
'(change-length :times 2 x :ratio)
 (do-section (mtypes-to-template 'zl output-all/va)
'(change-length :times 4 x :ratio)
(do-section (mtypes-to-template 'lr output-all/va) ; length-
repeat
'(length-repeat 2 x)
; (p-replace-sections (mtypes-to-template 'or output-all/va)
;(do-section (mtypes-to-template 'or output-all/va) '(flatten
x)
;(do-section
;(mtypes-to-template 'or output-all/va)
; '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
; f-sym-ph-o ))
f-len-ph-ova))))))))))

(init-rnd 0.5598)

(setq va-dyn-o
```

```
(do-section (mtypes-to-template 'pp output-all/va)
'(dyn-pp 25  x)
(do-section (mtypes-to-template 'p output-all/va)
'(dyn-p 45  x)
(do-section (mtypes-to-template 'mp output-all/va)
'(dyn-mp 60  x)
(do-section (mtypes-to-template 'mf output-all/va)
'(dyn-mf 75  x)
(do-section (mtypes-to-template 'f output-all/va)
'(dyn-f 96 x)
(do-section (mtypes-to-template 'ff output-all/va)
'(dyn-ff 115 x)
(do-section (mtypes-to-template 'pf output-all/va)
'(dyn-pf 45 96 x)
(do-section (mtypes-to-template 'fp output-all/va)
'(dyn-fp 96 45 x)
(do-section (mtypes-to-template 'rd output-all/va)
'(dyn-rnd 45 96 x)
set-dyn))))))))))

(init-rnd 0.5598)

(setq vc-sym-o
(p-replace-sections
(mtypes-to-template 'ls output-all/vc) sw-syms
(do-section
(mtypes-to-template 'sr output-all/vc)
'(symbol-repeat 2 x)
(do-section
(mtypes-to-template 'mr output-all/vc)
'(make-rest x)
(do-section
(mtypes-to-template 'fb output-all/vc)
'(find-beat x)
(do-section
(mtypes-to-template 'ct output-all/vc)
'(symbol-cut-i 1 2 x nil :start)
(do-section
(mtypes-to-template 'sm output-all/vc)
'(symbol-mask-i x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'ts output-all/vc)
'(symbol-thin 2 5 x (pick1 nil '(0.1 0.2 0.21 0.22)))
(do-section
(mtypes-to-template 'cs output-all/vc)
'(symbol-contract-i 4 8 x (pick1 nil '(0.3 0.4 0.51 0.62)))
(do-section
(mtypes-to-template 'st output-all/vc)
'(sequence-transpose x)
(do-section
(mtypes-to-template 'se output-all/vc)
'(symbol-list-expand x)
(do-section
(mtypes-to-template 'dt output-all/vc)
'(distort-transpose 1 x)
(do-section
(mtypes-to-template 'sh output-all/vc)
'(symbol-harmonize nil 'mix -6 6  x)
(do-section
(mtypes-to-template 'si output-all/vc)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/vc)
'(chord-creator (get-random 3 7) '2  x)
(do-section
 (mtypes-to-template 'fg output-all/vc)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/vc)
  '(play-registers (pick1 nil '(2 3 4)) '(-7 12 -6 6 -4 4) x)
  (do-section
 (mtypes-to-template 'bs output-all/vc)
 '(symbol-bundle-x 2 x nil)
  (do-section
 (mtypes-to-template 'su output-all/vc)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/vc)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/vc)
 '(symbol-floating x)
(do-section
(mtypes-to-template 'gp output-all/vc)
'(gen-palindrome x)
(do-section
(mtypes-to-template 'gc output-all/vc)
'(g-coda (length x) x)
(do-section
(mtypes-to-template 'gi output-all/vc)
```

```
'(g-intro (length x) x)
(do-section
(mtypes-to-template 'rv output-all/vc)
'(symbol-retrograde x)
(do-section
(mtypes-to-template 'ii output-all/vc)
'(interval-edit-i x)
(do-section
(mtypes-to-template 'ij output-all/vc)
'(interval-edit-j x)
(do-section
(mtypes-to-template 'ik output-all/vc)
'(interval-edit-k x)
(do-section
(mtypes-to-template 'il output-all/vc)
'(interval-edit-l x)
(do-section
(mtypes-to-template 'f1 output-all/vc)
'(filter-delete wt1 x)
(do-section
(mtypes-to-template 'f2 output-all/vc)
'(filter-delete wt2 x)
(do-section (mtypes-to-template 'or output-all/vc) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/vc)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 f-sym-ph-ovc))))))))))))))))))))))))))))))))))))

(init-rnd 0.5598)

(setq vc-len-o
(p-replace-sections (mtypes-to-template 'lb output-all/vc)
(do-section (mtypes-to-template 'lb output-all/vc)
 '(length-bundle x) vc-sym-o)
(p-replace-sections (mtypes-to-template 'lg output-all/vc)
(do-section (mtypes-to-template 'lg output-all/vc)
 '(length-group x) v1-sym-o)
(do-section (mtypes-to-template 'gp output-all/vc)
'(gen-palindrome  x)
(do-section (mtypes-to-template 'lw output-all/vc)
'(length-variate nil (get-random 3 5) 4 x)
(do-section (mtypes-to-template 'lv output-all/vc)
'(length-variate nil (get-random 3 5) 2 x)
(do-section (mtypes-to-template 'xl output-all/vc)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'yl output-all/vc)
'(change-length :divide 4 x :ratio)
(do-section (mtypes-to-template 'cl output-all/vc)
'(change-length :times 2 x :ratio)
 (do-section (mtypes-to-template 'zl output-all/vc)
'(change-length :times 4 x :ratio)
(do-section (mtypes-to-template 'lr output-all/vc) ; length-
repeat
'(length-repeat 2 x)
; (p-replace-sections (mtypes-to-template 'or output-all/vc)
;(do-section (mtypes-to-template 'or output-all/vc) '(flatten
x)
;(do-section
;(mtypes-to-template 'or output-all/vc)
; '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
; f-sym-ph-o ))
f-len-ph-ovc))))))))))

(init-rnd 0.5598)

(setq vc-dyn-o
(do-section (mtypes-to-template 'pp output-all/vc)
'(dyn-pp 25  x)
(do-section (mtypes-to-template 'p output-all/vc)
'(dyn-p 45 x)
(do-section (mtypes-to-template 'mp output-all/vc)
'(dyn-mp 60  x)
(do-section (mtypes-to-template 'mf output-all/vc)
'(dyn-mf 75  x)
(do-section (mtypes-to-template 'f output-all/vc)
'(dyn-f 96 x)
(do-section (mtypes-to-template 'ff output-all/vc)
'(dyn-ff 115 x)
(do-section (mtypes-to-template 'pf output-all/vc)
'(dyn-pf 45 96 x)
(do-section (mtypes-to-template 'fp output-all/vc)
'(dyn-fp 96 45 x)
(do-section (mtypes-to-template 'rd output-all/vc)
'(dyn-rnd 45 96 x)
set-dyn)))))))))

(init-rnd 0.5598)
```

This section of the score-file includes extra processes that don't 'fit' the standard Process-List form

```
(setq v1-sym-ox
      (do-section
       (mtypes-to-template 'sg output-all/v1)
       '(symbol-group x)
       (do-section
        (mtypes-to-template 'dr output-all/v1)
        '(filter-delete '= x) v1-sym-o)))

(init-rnd 0.5598)

(setq v2-sym-ox
      (do-section
      (mtypes-to-template 'sg output-all/v2)
      '(symbol-group x)(do-section
      (mtypes-to-template 'dr output-all/v2)
      '(filter-delete '= x) v2-sym-o)))

(init-rnd 0.5598)

(setq va-sym-ox
      (do-section
       (mtypes-to-template 'sg output-all/va)
       '(symbol-group x)
       (do-section
        (mtypes-to-template 'dr output-all/va)
        '(filter-delete '= x) va-sym-o )))

(init-rnd 0.5598)

(setq vc-sym-ox
      (do-section
       (mtypes-to-template 'sg output-all/vc)
       '(symbol-group x)
       (do-section
        (mtypes-to-template 'dr output-all/vc)
        '(filter-delete '= x) vc-sym-o)))

;; accessing tremolando and pizzicato

(setq arco-v1 (build-list '(41) (length v1-len-o))
      arco-v2 (build-list '(41) (length v1-len-o))
      arco-va (build-list '(42) (length v1-len-o))
      arco-vc (build-list '(43) (length v1-len-o)))

(setq arco-v1-p (do-section
                  (mtypes-to-template 'tr output-all/v1)
                  '(trem '(45) x)
                  (do-section
                  (mtypes-to-template 'pz output-all/v1)
                  '(pizz '(46) x)
                  arco-v1)))

(setq arco-v2-p (do-section
                  (mtypes-to-template 'tr output-all/v2)
                  '(trem '(45) x)
                  (do-section
                  (mtypes-to-template 'pz output-all/v2)
                  '(pizz '(46) x)
                  arco-v2)))

(setq arco-va-p (do-section
                  (mtypes-to-template 'tr output-all/va)
                  '(trem '(45) x)
                  (do-section
                  (mtypes-to-template 'pz output-all/va)
                  '(pizz '(46) x)
                  arco-va)))

(setq arco-vc-p (do-section
                  (mtypes-to-template 'tr output-all/vc)
                  '(trem '(45) x)
                  (do-section
                  (mtypes-to-template 'pz output-all/vc)
                  '(pizz '(46) x)
                  arco-vc)))

(setq stacc-v1
      (do-section
                  (mtypes-to-template 'so output-all/v1)
                  '(stacc 2 x)
                  (mapcar 'phrase-articulation v1-len-o)))

(setq stacc-v2
      (do-section
                  (mtypes-to-template 'so output-all/v2)
                  '(stacc 2 x)
                  (mapcar 'phrase-articulation v2-len-o)))
```

The Output of the main Process-List

Here any changes to that default list of program-changes is triggered from the Score-Sheet

Here a default list of program-change values is created

```
(setq stacc-va
      (do-section
                 (mtypes-to-template 'so output-all/va)
                 '(stacc 2 x)
                 (mapcar 'phrase-articulation va-len-o)))
```

```
(setq stacc-vc
      (do-section
                 (mtypes-to-template 'so output-all/vc)
                 '(stacc 2 x)
                 (mapcar 'phrase-articulation vc-len-o)))
```

```
(setq tempo-map
      '(105  30   45  30   75   30  90   30   75    30   90   30  60   30  30  45   30  75  30   105  30  75))
;     (15/4 5/8 11/4 13/8 87/8 3/2 51/8 13/8 213/8 1/1 105/8 9/8 13/2 3/8 3/8 17/8 3/2 9/2 11/8 77/4 9/8 53/8)
```

The Tempo values here in quarter-beats per minute are aligned to the zone-structure of each section.

```
;; score-template


(def-tonality
v1 (activate-tonality (chromatic f 5))
v2 (activate-tonality (chromatic f 5))
va (activate-tonality (chromatic f 4))
vc (activate-tonality (chromatic f 3))
)


(def-symbol
  v1 v1-sym-ox
  v2 v2-sym-ox
  va va-sym-ox
  vc vc-sym-ox

)

(def-length
v1 v1-len-o
v2 v2-len-o
va va-len-o
vc vc-len-o
)
```

```
(def-duration
v1 stacc-v1
v2 stacc-v2
va stacc-va
vc stacc-vc
)


(def-velocity
v1 v1-dyn-o
v2 v2-dyn-o
va va-dyn-o
vc vc-dyn-o
)

(def-zone
tempo (zone-ratio-sc (symbol-divide zone-lengths nil nil  (zone-ratio-sc v1-len-o )))
default (zone-ratio-sc v1-len-o ) ; zone-ratio-sc used here because of value 1 appears with make-zone
)

#| ;; zone structure

(13/8 5/8 7/8 5/4 5/4 9/8 13/8 13/4 5/8 1/8 1/4 7/16 9/8 1/2 5/4 1/2 7/16 9/8
7/8 1/2 1/4 3/2 3/1 1/1 5/8 1/2 5/4 7/16 13/8 13/4 1/1 9/8 3/4 2/1 7/8 5/8 7/8
1/1 1/4 1/1 3/2 9/8 3/4 3/2 1/2 3/4 13/8 5/8 11/8 13/8 3/2 11/8 1/4 3/4 1/2 3/8
1/1 5/4 11/8 1/4 1/2 2/1 11/8 3/8 1/8 5/8 3/2 1/8 1/2 5/4 3/8 5/16 1/1 3/8 3/4
11/8 3/2 9/8 9/4 3/2 7/8 1/1 3/4 1/1 11/8 3/2 3/4 3/4 5/8 3/2 3/1 3/4 1/4 3/4
7/8 2/1 11/8 11/4 1/8 3/16 3/8 5/8 13/8 1/8 1/1 1/2 13/8 1/1 3/2 1/4 1/1 11/8
9/8 11/8 3/2 1/1 1/1 7/8 9/8 9/4 1/1 1/1 3/4 13/8 1/2 1/1 7/8 3/4)
|#

(def-channel
v1 1
v2 2
va 3
vc 4
)

(def-controller-set gm-controllers
   group general
      Bank-change-0 0
      Bank-change-1 32
)

;; enables vc to use SC55 bank - no velocity change on other banks
```

```
(def-controller gm-controllers
(vc
bank-change-0 '((0))
bank-change-1 '((1)))
)


(def-program nil
v1  (absolutes arco-v1-p)
v2  (absolutes arco-v2-p)
va  (absolutes arco-va-p)
vc  (absolutes arco-vc-p)
)


(def-tempo
  tempo-map
)

(compile-instrument-p "ccl;output:" "OofM1ii "
v1
v2
va
vc
)
```

This is a necessary prefix before a list of program-change values

I- Functions (February 2005)

Structure (zone/phrase-length)

lr - length-repeat (div 2)
li - length-repeat (get-random 2 4)
cl - change-length (times 2) *
xl - change-length (divide 2) *
yl - change-length (divide 4) *
zl - change-length (times 4) *

se - symbol-list-expand *
st - sequence-transpose *
gp - gen-palindrome **
gi - g-intro **
gc - g-coda **

Rhythm (keeping symbol sequence and/or position intact)

lv - length-variate
lw - length-variate(2)
ls - length-swallow
lg - length-group (with sg)
lb - length-bundle (with dr)

Rhythm (altering symbol content = creating new beat/space relationships)

symbol-bundle-i (x)
cc - create-chords

sk - symbol-skip-i
ct - symbol-cut-i (:start)
kt - symbol-cut-i (:end)
ts - symbol-thin
sm - symbol-mask-i

fd/dr - filter-delete
f1/f2 - filter-delete (extracting whole-tone tonalities)
ii - interval-edit-i
ij - interval-edit-j
ik - interval-edit-k
il - interval-edit-l
cs - symbol-contract

Pitch (altering content and/or position of phrase items)

ss - symbol-scale
sc - symbol-compress
sf - symbol-fold
sg - symbol-group
si - symbol-inversion (@ 'g)
sh - symbol-harmonize
sz - symbol-chordize
fg - symbol-figurate
or - symbol-ornamentation
dt - distort-transpose
pr - play-registers
su - symbol-upward
sd - symbol-downward
fl - symbol-floating
vi - verbal-intervention
rv - reverse
sr - symbol-repeat

Dynamics

pp - dyn-pp
p - dyn-p
mp - dyn-mp
mf - dyn-mf
f - dyn-f
ff - dyn-ff
pf - dyn-pf
fp - dyn-fp
rd - dyn-rnd

Orchestration

mr - make-rest
cs - symbol-contract
fd/dr - filter-delete
f1/f2 - filter-delete
ii - interval-edit-i
ij - interval-edit-j
ik - interval-edit-k
il - interval-edit-l

pz - pizzicato
tr - tremolando