rsc3: SuperCollider client

Version 8.13

Nik Gaffney, et al.

June 30, 2024

```
(require rsc3)
```
package: `rsc3`

The `rsc3` module is a port of the rsc3 SuperCollider client to Racket. rsc3 is written by Rohan Drape and initially ported to Racket by Mustafa Khafateh using the Thu Jun 27 2013 version.

Documentation, help files and tutorials are included, along with the utilities to generate scribble files from the upstream supercollideer documentation.

The current version of rd–rsc3 is available at gitlab

1 Reference & resuscitation

Contents

1 Reference & resuscitation **2**

2 Further **4**

3 rsc3: server commands **5**

3.1 /g_deepFree Free all synths in this group and all its sub-groups. 7

3.2 /c_setn Set ranges of bus value(s) 7

3.3 /sync Notify when async commands have completed. 7

3.4 /s_getn Get ranges of control value(s) 7

3.5 /s_get Get control value(s) 7

3.6 /n_mapn Map a node's controls to read from buses 8

3.7 /b_gen Call a command to fill a buffer 8

3.8 /n_trace Trace a node 9

3.9 /b_alloc Allocate buffer space. 9

3.10 /n_query Get info about a node 9

3.11 /d_load Load synth definition 10

3.12 /n_fill Fill ranges of a node's control value(s) 10

3.13 /n_before Place a node before another 10

3.14 /b_close 10

3.15 see s-new 10

3.16 /s_new Create a new synth 10

3.17 /b_setn Set ranges of sample value(s) 11

3.18 /c_set Set bus value(s) 11

3.19 /n_free Delete a node. 12

3.20 see n-set 12

3.21 /n_run Turn node on or off 12
3.22 /b_read Read sound file data into an existing buffer. 12
3.23 /nrt_end end real time mode, close file 12
3.24 /b_get Get sample value(s) 13
3.25 /n_after Place a node after another 13
3.26 /d_free delete synth definition 13
3.27 /n_set Set a node's control value(s) 13
3.28 /b_write Write sound file data. 14
3.29 /b_getn Get ranges of sample value(s) 14
3.30 /b_fill Fill ranges of sample value(s) 14
3.31 /s_noid Auto-reassign synth's ID to a reserved value 14
3.32 /d_recv Receive a synth definition file 15
3.33 /u_cmd send a command to a unit generator 15
3.34 /b_allocRead Allocate buffer space and read a sound file. 15
3.35 /n_map Map a node's controls to read from a bus 15
3.36 /status Query the status 16
3.37 /g_new Create a new group 16
3.38 /c_get Get bus value(s) 16
3.39 /b_set Set sample value(s) 16
3.40 /b_zero Zero sample data 17
3.41 /tr A trigger message 17
3.42 /dumpOSC Display incoming OSC messages 17
3.43 /n_setn Set ranges of a node's control value(s) 17
3.44 /b_free Free buffer data. 18
3.45 /notify Register to receive notifications from server 18

3.46 /b_query . . . 18
3.47 /g_tail Add node to tail of group . . . 18
3.48 /clearSched Clear all scheduled bundles. . . . 18
3.49 /g_freeAll Delete all nodes in a group. . . . 19
3.50 /g_head Add node to head of group . . . 19
3.51 /c_getn Get ranges of bus value(s) . . . 19
3.52 /quit Quit program . . . 19
3.53 see g-new . . . 19
3.54 /c_fill Fill ranges of bus value(s) . . . 19

4 rsc3: ugens **20**

4.1 (Rlpf in freq rq) . . . 22
4.2 (fold in lo hi) . . . 23
4.3 (formlet in freq attackTime decayTime) . . . 23
4.4 (lin-lin in srclo srchi dstlo dsthi) . . . 24
4.5 (fos in a0 a1 b1) . . . 24
4.6 (bpf in freq rq) . . . 24
4.7 (wrap in lo hi) . . . 25
4.8 (bpz2 in) . . . 25
4.9 (amp-comp freq root exp) . . . 25
4.10 (lag2 in lagTime) . . . 26
4.11 (one-zero in coef) . . . 26
4.12 (klank in freqScale freqOffset decayScale spec) . . . 27
4.13 (lpz2 ar in) . . . 28
4.14 (lin-exp in srclo srchi dstlo dsthi) . . . 28
4.15 (hpz2 in) . . . 28

4.16 (leak-dc in coef) 28
4.17 (hpf in freq) 29
4.18 (sos in a0 a1 a2 b1 b2) 29
4.19 (normalizer in level dur) 29
4.20 (two-zero in freq radius) 29
4.21 (clip in lo hi) 30
4.22 (limiter input level lookAheadTime) 30
4.23 (median length in) 30
4.24 (lag3 in lagTime) 31
4.25 (amp-compA freq root minAmp rootAmp) 31
4.26 (two-pole in freq radius) 32
4.27 (lpf in freq) 32
4.28 (hpz1 in) 32
4.29 (freq-shift input shift phase) 33
4.30 (lpz1 ar in) 33
4.31 (moog-ff in freq gain reset) 34
4.32 (resonz in freq bwr) 34
4.33 (brz2 in) 35
4.34 (Rhpf in freq rq) 35
4.35 (brf in freq rq) 36
4.36 (one-pole in coef) 36
4.37 (lag in lagTime) 36
4.38 (ringz in freq decayTime) 36
4.39 (dyn-klank in freqScale freqOffset decayScale spec) 37
4.40 (Grainin nc tr dur in pan envbuf) 39

4.41 (grain-buf nc tr dur sndbuf rate pos interp pan envbuf) 39
4.42 (grain-sin nc tr dur freq pan envbuf) 40
4.43 (warp1 nc buf ptr freqScale windowSize envbuf overlaps windowrandRatio interp) 41
4.44 (grain-fm nc tr dur carfreq modfreq index pan envbuf) 41
4.45 (disk-in num-channels rate bufnum) 42
4.46 (disk-out bufnum channels) 43
4.47 num-input-buses 44
4.48 num-control-buses 44
4.49 num-buffers 44
4.50 num-output-buses 44
4.51 (buf-frames rate bufnum) 44
4.52 (buf-rate-scale rate bufnum) 45
4.53 sample-dur 45
4.54 sample-rate 45
4.55 radians-per-sample 45
4.56 (buf-dur rate bufnum) 45
4.57 (buf-channels rate bufnum) 46
4.58 (buf-sample-rate rate bufnum) 46
4.59 num-audio-buses 46
4.60 num-running-synths 46
4.61 subsample-offset 46
4.62 (zero-crossing in) 47
4.63 (slope in) 48
4.64 (running-sum in numsamp) 48
4.65 (pitch in initFreq minFreq maxFreq execFreq maxBinsPerOctave 49

4.66 (compander input control thresh slopeBelow slopeAbove clampTime relaxTime) . . . 49
4.67 (amplitude rate in attackTime releaseTime) . . . 51
4.68 (pulse-divider trig div start) . . . 51
4.69 (gate in trig) . . . 51
4.70 (poll trig in trigid label) . . . 51
4.71 (most-change a b) . . . 52
4.72 (trig in dur) . . . 53
4.73 (pulse-count trig reset) . . . 53
4.74 (stepper trig reset min max step resetval) . . . 53
4.75 (last-value in diff) . . . 55
4.76 (peak-follower in decay) . . . 55
4.77 (running-max in trig) . . . 56
4.78 (trig1 in dur) . . . 57
4.79 (phasor trig rate start end resetpos) . . . 57
4.80 (schmidt in lo hi) . . . 57
4.81 (peak trig reset) . . . 57
4.82 (toggle-ff trig) . . . 58
4.83 (sweep trig rate) . . . 58
4.84 (send-trig in id value) . . . 59
4.85 (in-range in lo hi) . . . 59
4.86 (timer trig) . . . 60
4.87 (t-delay trigger delayTime) . . . 60
4.88 (running-min in trig) . . . 60
4.89 (set-reset-ff trig reset) . . . 60
4.90 (saw rate freq) . . . 61

4.91 (pm-osc rate carfreq modfreq index modphase) 61
4.92 (lf-tri rate freq iphase) 62
4.93 (t-grains numChannels trigger bufnum rate centerPos dur pan amp interp) . 62
4.94 (tw-index in normalize array) 63
4.95 (osc-n rate bufnum freq phase) 64
4.96 (osc rate bufnum freq phase) 64
4.97 (lf-saw rate freq iphase) 65
4.98 (tw-choose trig array weights normalize) 66
4.99 gendy1 66
4.100 (pulse rate freq width) 69
4.101 (shaper bufnum in) 69
4.102 SC2: Note extra iphase argument. 70
4.103 (v-osc rate bufpos freq phase) 70
4.104 (var-saw rate freq iphasewidth) 71
4.105 buf-wr 72
4.106 impulse 72
4.107 blip 73
4.108 (select which array) 73
4.109 formant 73
4.110 c-osc 74
4.111 (v-osc3 rate bufpos freq1 freq2 freq3) 74
4.112 (lf-cub rate freq iphase) 76
4.113 (lf-pulse rate freq iphase width) 77
4.114 index 77
4.115 See lf-cub. 77

4.116(sync-saw rate syncFreq sawFreq) 77
4.117(t-choose trig array) 78
4.118(sin-osc rate freq phase) 78
4.119(klang rate freqScale freqOffset spec) 79
4.120(lag-in num-channels bus lag) 79
4.121(sound-in channel) 79
4.122(in num-channels rate bus) 80
4.123(in-trig num-channels bus) 80
4.124(replace-out bufferindex inputs) 81
4.125(local-in num-channels rate) 81
4.126(offset-out bufferindex inputs) 81
4.127(in-feedback num-channels bus) 82
4.128(x-out buffer-index xfade inputs) 83
4.129(out bufferindex inputs) 83
4.130(mix UGen) 83
4.131(mix-fill n f) 84
4.132(latch in trig) 84
4.133(decay in decayTime) 84
4.134(wrap-index bufnum in) 85
4.135(mouse-y rate minval maxval warp lag) 85
4.136(degree-to-key bufnum in octave) 85
4.137(key-state rate keynum minval maxval lag) 86
4.138(mrg2 left right) 86
4.139(mouse-button rate minval maxval lag) 87
4.140(slew in up dn) 87

4.141(mouse-x rate minval maxval warp lag) 87
4.142(decay2 in attackTime decayTime) 88
4.143(k2a in) 88
4.144(mul-add a b c) 89
4.145(clip2 a b) 89
4.146(Atan2 x y) 89
4.147(trunc a b) 90
4.148(sub a b) 90
4.149(round-up a b) 90
4.150(ring4 a b) 91
4.151(pow a b) 91
4.152(ring1 a b) 92
4.153(gt a b) 92
4.154(add a b) 93
4.155(abs-dif a b) 93
4.156(am-clip a b) 93
4.157(ge a b) 93
4.158(max a b) 94
4.159(ring3 a b) 94
4.160(thresh a b) 94
4.161(dif-sqr a b) 94
4.162(excess a b) 95
4.163(fold2 a b) 95
4.164(sqr-dif a b) 95
4.165(hypot x y) 96

4.166(sqr-sum a b) . . . 96
4.167(sum-sqr a b) . . . 97
4.168(le a b) . . . 97
4.169eq . . . 97
4.170(scale-neg a b) . . . 97
4.171(ring2 a b) . . . 98
4.172(Mod a b) . . . 98
4.173(fdiv a b) . . . 98
4.174(mul a b) . . . 98
4.175(min a b) . . . 99
4.176(lt a b) . . . 99
4.177(wrap2 a b) . . . 99
4.178(round a b) . . . 99
4.179(hasher in) . . . 100
4.180(rand-seed rate trig seed) . . . 100
4.181(lfd-noise0 rate freq) . . . 100
4.182(lfclip-noise rate freq) . . . 102
4.183See lf-noise0 . . . 102
4.184(clip-noise rate) . . . 102
4.185(ti-rand lo hi trig) . . . 102
4.186(lf-noise0 rate freq) . . . 103
4.187(pink-noise rate) . . . 104
4.188(rand lo hi) . . . 104
4.189(gray-noise rate) . . . 104
4.190See lfd-noise0 . . . 104

4.191 (i-rand lo hi) . . . 104
4.192 (n-rand lo hi n) . . . 104
4.193 (lfdclip-noise rate freq) . . . 105
4.194 (coin-gate prob in) . . . 105
4.195 (t-exp-rand lo hi trig) . . . 106
4.196 (t-rand lo hi trig) . . . 106
4.197 (white-noise rate) . . . 106
4.198 (dust2 rate density) . . . 107
4.199 See lfd-noise0 . . . 107
4.200 (rand-id rate id) . . . 107
4.201 See lf-noise0 . . . 108
4.202 (mantissa-mask in bits) . . . 108
4.203 (dust rate density) . . . 108
4.204 (lin-rand lo hi minmax) . . . 108
4.205 (exp-rand lo hi) . . . 109
4.206 brown-noise . . . 109
4.207 (u:log a) . . . 109
4.208 (frac a) . . . 109
4.209 (arc-tan a) . . . 110
4.210 (distort a) . . . 110
4.211 (tan-h a) . . . 110
4.212 (u:floor a) . . . 110
4.213 (cps-oct a) . . . 110
4.214 (db-amp a) . . . 111
4.215 (u:sqrt a) . . . 111

4.216(soft-clip a) . . . 111
4.217(cps-midi a) . . . 111
4.218(is-strictly-positive a) . . . 112
4.219(u:tan a) . . . 112
4.220(cos-h a) . . . 112
4.221(amp-db a) . . . 112
4.222(abs a) . . . 113
4.223(log10 a) . . . 113
4.224(midi-cps a) . . . 113
4.225(is-positive a) . . . 113
4.226(sign a) . . . 114
4.227(neg a) . . . 114
4.228(log2 a) . . . 114
4.229(ceil a) . . . 114
4.230(arc-cos a) . . . 114
4.231(exp a) . . . 115
4.232(squared a) . . . 115
4.233(arc-sin a) . . . 115
4.234(cubed a) . . . 115
4.235(is-negative a) . . . 116
4.236(oct-cps a) . . . 116
4.237(u:sin a) . . . 116
4.238(sin-h a) . . . 117
4.239(cos a) . . . 117
4.240(pv-mul bufferA bufferB) . . . 117

4.241(pv-mag-squared buffer) 117
4.242(pv-min bufferA bufferB) 117
4.243(pv-mag-noise buffer) 117
4.244(pv-mag-below buffer threshold) 118
4.245(convolution in kernel framesize) 118
4.246(pv-jensen-andersen buffer propsc prophfe prophfc propsf threshold waittime) 118
4.247(pv-phase-shift270 buffer) 119
4.248(pv-hainsworth-foote buffer proph propf threshold waittime) 119
4.249(pv-phase-shift buffer shift) 121
4.250(pv-copy bufferA bufferB) 121
4.251(pv-brick-wall buffer wipe) 122
4.252(pv-mag-smear buffer bins) 122
4.253(pv-mag-above buffer threshold) 123
4.254(pv-bin-shift buffer stretch shift) 123
4.255(fft buffer in hop wintype active) 124
4.256(pv-bin-wipe bufferA bufferB wipe) 125
4.257(pv-copyPhase bufferA bufferB) 125
4.258(pv-phase-shift90 buffer) 126
4.259(convolution2 in bufnum trigger framesize) 126
4.260(pv-rect-comb2 bufferA bufferB numTeeth phase width) 128
4.261(pv-add bufferA bufferB) 128
4.262(pv-rand-comb buffer wipe trig) 129
4.263(pv-local-max buffer threshold) 129
4.264(pv-mag-mul bufferA bufferB) 129
4.265(pv-conformal-map buffer real imag) 129

4.266(pv-diffuser buffer trig) 130
4.267(pv-max bufferA bufferB) 131
4.268(Ifft buffer wintype) 131
4.269(pv-bin-scramble buffer wipe width trig) 131
4.270(pv-rand-wipe bufferA bufferB wipe trig) 132
4.271(Packfft chain bufsize frombin tobin zeroothers magsphases) 133
4.272(pvcollect chain numframes func frombin tobin zeroothers) 134
4.273(PV_Magclip buffer threshold) 135
4.274(PV_Magfreeze buffer freeze) 135
4.275(pv-rect-comb buffer numTeeth phase width) 136
4.276(pv-mag-shift buffer stretch shift) 137
4.277See allpass-n 137
4.278See comb-n 137
4.279See Bufallpass-c 137
4.280(free-verb in mix room damp) 137
4.281See freeVerb 137
4.282(play-buf numChannels bufnum rate trigger startPos loop) 137
4.283See buf-delay-c 139
4.284(delay2 in) 139
4.285(comb-n in maxDelayTime delayTime decayTime) 139
4.286See Bufallpass-c 140
4.287(allpass-n in maxDelayTime delayTime decayTime) 140
4.288See comb-n 141
4.289(buf-allpass-c buf in delaytime decaytime) 141
4.290See allpass-n 142

4.291 See buf-delay-c . . . 142
4.292 (buf-delay-c buf in delaytime) . . . 142
4.293 (pluck in tr maxdelaytime delaytime decaytime coef) . . . 142
4.294 (pitch-shift in winSize pchRatio pchDispersion timeDispersion) . . . 143
4.295 See buf-comb-c . . . 143
4.296 (buf-comb-c buf in delaytime decaytime) . . . 143
4.297 (buf-rd numChannels rate bufnum phase loop interpolation) . . . 144
4.298 (delay1 in) . . . 145
4.299 (record-buf bufnum offset reclevel prelevel run loop trigger inputs) . . . 145
4.300 See delay-n . . . 147
4.301 See buf-comb-c . . . 147
4.302 See delay-n . . . 147
4.303 (delay-n in maxDelayTime delayTime) . . . 147
4.304 (ball in g damp friction) . . . 147
4.305 (dswitch1 index array) . . . 147
4.306 (t-duty rate duration reset doneAction level gap) . . . 148
4.307 See dwhite . . . 149
4.308 (dwhite length lo hi) . . . 149
4.309 (dbufrd bufnum phase loop) . . . 149
4.310 (demand-env-gen rate levels times shapes curves gate reset . . . 150
4.311 (demand trig reset ugens) . . . 151
4.312 (duty rate duration reset doneAction level) . . . 152
4.313 (dser length array) . . . 153
4.314 (dgeom length start grow) . . . 153
4.315 (drand length array) . . . 154

4.316 See drand 154
4.317 (dseries length start step) 154
4.318 (dswitch index array) 155
4.319 See dbrown 155
4.320 (dbrown length lo hi step) 155
4.321 (dseq length array) 156
4.322 See latoocarfian-c. 156
4.323 (fb-sine-c rate freq im fb a c xi yi) 156
4.324 See fb-sine-c 157
4.325 (quad-n rate freq a b c xi) 157
4.326 (lorenz-l rate freq s r b h xi yi zi) 158
4.327 See standard-l. 159
4.328 See quad-n 159
4.329 See cusp-n 159
4.330 See latoocarfian-c. 159
4.331 (logistic rate chaosParam freq) 159
4.332 See fb-sine-c 160
4.333 (latoocarfian-c rate freq a b c d xi yi) 160
4.334 (rossler rate chaosParam dt) 160
4.335 (standard-l rate freq k xi yi) 161
4.336 See quad-n 161
4.337 (cusp-n rate freq a b xi) 161
4.338 (lin-cong-c rate freq a c m xi) 162
4.339 (crackle rate chaosParam) 163
4.340 (henon-n rate freq a b x0 x1) 163

4.341 See henon-n . . . 165
4.342 See lin-cong-c. . . . 165
4.343 See GbmanL. . . . 165
4.344 See GbmanL. . . . 165
4.345 See henon-n . . . 165
4.346 See lin-cong-c. . . . 165
4.347 (gbman-c rate freq xi yi) . . . 165
4.348 (lin-pan2 in pos level) . . . 166
4.349 (rotate2 x y pos) . . . 166
4.350 (decode-b2 numChannels w x y orientation) . . . 167
4.351 (pan2 in pos level) . . . 168
4.352 (pan-b2 in azimuth gain) . . . 168
4.353 (detect-silence in amp time doneAction) . . . 168
4.354 (line rate start end dur doneAction) . . . 169
4.355 (free trig nodeID) . . . 169
4.356 (pause-self-when-done src) . . . 169
4.357 (pause-self src) . . . 170
4.358 (env-gen rate gate levelScale levelBias timeScale doneAction envelope) . . . 170
4.359 (free-self-when-done src) . . . 171
4.360 (pause gate nodeID) . . . 171
4.361 (x-line rate start end dur doneAction) . . . 172
4.362 (done src) . . . 172
4.363 (linen gate attackTime susLevel releaseTime doneAction) . . . 172
4.364 (free-self src) . . . 173

5 rsc3: tutorials 174

Index **177**

Index **177**

2 Further

Partially regenerated documentation (rsc.help.scm → rsc.scrbl)

3 rsc3: server commands

server commands

Contents

3.1 /g_deepFree Free all synths in this group and all its sub-groups.

[int - group ID] * N

Traverses all groups below this group and frees all the synths. Sub-groups are not freed. A list of groups may be specified.

3.2 /c_setn Set ranges of bus value(s)

[int - starting bus index int - number of sequential buses to change (M) [float - a control value] * M] * N

Set contiguous ranges of buses to sets of values. For each range, the starting bus index is given followed by the number of channels to change, followed by the values.

3.3 /sync Notify when async commands have completed.

int - a unique number identifying this command.

Replies with a /synced message when all asynchronous commands received before this one have completed. The reply will contain the sent unique ID.

Asynchronous. Replies to sender with /synced, ID when complete.

3.4 /s_getn Get ranges of control value(s)

int - synth ID [int|string - a control index or name int - number of sequential controls to get (M)] * N

Get contiguous ranges of controls. Replies to sender with the corresponding /n_setn command.

3.5 /s_get Get control value(s)

int - synth ID [int or string - a control index or name] * N

Replies to sender with the corresponding /n_set command.

3.6 /n_mapn Map a node's controls to read from buses

int - node ID [int or string - a control index or name int - control bus index int - number of controls to map] * N

Takes a list of triplets of control names or indices, bus indices, and number of controls to map and causes those controls to be mapped sequentially to buses. If the node is a group, then it maps the controls of every node in the group. If the control bus index is -1 then any current mapping is undone and control reverts to normal.

See also: /n_map

3.7 /b_gen Call a command to fill a buffer

int - buffer number string - command name .. command arguments

Plug-ins can define commands that operate on buffers. The arguments after the command name are defined by the command. The currently defined buffer fill commands are listed below in a separate section.

Buffer Fill Commands

These are the currently defined fill routines for use with the /b_gen command.

Common flags are defined as follows:

1 - normalize Normalize peak amplitude of wave to 1.0. 2 - wavetable If set, then the buffer is written in wavetable format so that it can be read by interpolating oscillators. 4 - clear If set then the buffer is cleared before new partials are written into it. Otherwise the new partials are summed with the existing contents of the buffer.

sine1 int - flags, see above [float - partial amplitude] * N

Fills a buffer with a series of sine wave partials. The first float value specifies the amplitude of the first partial, the second float value specifies the amplitude of the second partial, and so on.

sine2 int - flags, see above [float - partial frequency (in cycles per buffer) float - partial amplitude] * N

Similar to sine1 except that each partial frequency is specified explicitly instead of being an integer series of partials. Non-integer partial frequencies are possible.

sine3 int - flags, see above [float - partial frequency (in cycles per buffer) float - partial amplitude float - partial phase] * N

Similar to sine2 except that each partial may have a nonzero starting phase.

cheby int - flags, see above [float - amplitude] * N

Fills a buffer with a series of chebyshev polynomials, which can be defined as: cheby(n) = amplitude * cos(n * acos(x)). The first float value specifies the amplitude for n = 1, the second float value specifies the amplitude for n = 2, and so on. To eliminate a DC offset when used as a waveshaper, the wavetable is offset so that the center value is zero.

copy int - sample position in destination int - source buffer number int - sample position in source int - number of samples to copy

Copy samples from the source buffer to the destination buffer specified in the b_gen command. If the number of samples to copy is negative, the maximum number of samples possible is copied. Asynchronous. Replies to sender with /done when complete.

3.8 /n_trace Trace a node

[int - node ID] * N

Causes a synth to print out the values of the inputs and outputs of its unit generators for one control period. Causes a group to print the node IDs and names of each node in the group for one control period.

3.9 /b_alloc Allocate buffer space.

int - buffer number int - number of frames int - number of channels (optional. default = 1 channel) bytes - an OSC message to execute upon completion. (optional)

Allocates zero filled buffer to number of channels and samples.

Asynchronous. Replies to sender with /done when complete.

3.10 /n_query Get info about a node

[int - node ID] * N

The server sends an /n_info message for each node to registered clients.

See Node Notifications below for the format of the /n_info message.

3.11 /d_load Load synth definition

string - pathname of file. Can be a pattern like "synthdefs/perc-*" bytes - an OSC message to execute upon completion. (optional)

Loads a file of synth definitions. Resident definitions with the same names are overwritten.

Asynchronous. Replies to sender with /done when complete.

3.12 /n_fill Fill ranges of a node's control value(s)

int - node ID [int or string - a control index or name int - number of values to fill (M) float - value] * N

Set contiguous ranges of control indices to single values. For each range, the starting control index is given followed by the number of controls to change, followed by the value to fill. If the node is a group, then it sets the controls of every node in the group.

3.13 /n_before Place a node before another

[int - the ID of the node to place (A) int - the ID of the node before which the above is placed (B)] * N

Places node A in the same group as node B, to execute immediately before node B.

3.14 /b_close

int - buffer number

After using a buffer with DiskOut, close the soundfile and write header information.

3.15 see s-new

3.16 /s_new Create a new synth

string - synth definition name int - synth ID int - add action (0,1,2, 3 or 4 see below) int - add target ID [int or string - a control index or name float - a control value] * N

Create a new synth from a synth definition, give it an ID, and add it to the tree of nodes.

There are four ways to add the node to the tree as determined by the add action argument which is defined as follows:

0 - add the new node to the the head of the group specified by the add target ID.

1 - add the new node to the the tail of the group specified by the add target ID.

2 - add the new node just before the node specified by the add target ID.

3 - add the new node just after the node specified by the add target ID.

4 - the new node replaces the node specified by the add target ID. The target node is freed.

Controls may be set when creating the synth. The control arguments are the same as for the n_set command.

If you send /s_new with a synth ID of -1, then the server will generate an ID for you. The server reserves all negative IDs. Since you don't know what the ID is, you cannot talk to this node directly later. So this is useful for nodes that are of finite duration and that get the control information they need from arguments and buses or messages directed to their group. In addition no notifications are sent when there are changes of state for this node, such as /go, /end, /on, /off.

If you use a node ID of -1 for any other command, such as /n_map, then it refers to the most recently created node by /s_new (auto generated ID or not). This is how you can map the controls of a node with an auto generated ID. In a multi-client situation, the only way you can be sure what node -1 refers to is to put the messages in a bundle.

3.17 /b_setn Set ranges of sample value(s)

int - buffer number [int - sample starting index int - number of sequential samples to change (M) [float - a sample value] * M] * N

Set contiguous ranges of sample indices to sets of values. For each range, the starting sample index is given followed by the number of samples to change, followed by the values.

3.18 /c_set Set bus value(s)

[int - a bus index float - a control value] * N

Takes a list of pairs of bus indices and values and sets the buses to those values.

3.19 /n_free Delete a node.

[int - node ID] * N

Stops a node abruptly, removes it from its group, and frees its memory. A list of node IDs may be specified. Using this method can cause a click if the node is not silent at the time it is freed.

3.20 see n-set

3.21 /n_run Turn node on or off

[int - node ID int - run flag] * N

If the run flag set to zero then the node will not be executed. If the run flag is set back to one, then it will be executed. Using this method to start and stop nodes can cause a click if the node is not silent at the time run flag is toggled.

3.22 /b_read Read sound file data into an existing buffer.

int - buffer number string - path name of a sound file. int - starting frame in file (optional. default = 0) int - number of frames to read (optional. default = -1, see below) int - starting frame in buffer (optional. default = 0) int - leave file open (optional. default = 0) bytes - an OSC message to execute upon completion. (optional)

Reads sound file data from the given starting frame in the file and writes it to the given starting frame in the buffer. If number of frames is less than zero, the entire file is read. If reading a file to be used by DiskIn ugen then you will want to set "leave file open" to one, otherwise set it to zero.

Asynchronous. Replies to sender with /done when complete.

3.23 /nrt_end end real time mode, close file

NOT YET IMPLEMENTED

no arguments.

This message should be sent in a bundle in non real time mode. The bundle timestamp will establish the ending time of the file. This command will end non real time mode and close the sound file. Replies to sender with /done when complete.

3.24 /b_get Get sample value(s)

int - buffer number [int - a sample index] * N

Replies to sender with the corresponding /b_set command.

3.25 /n_after Place a node after another

[int - the ID of the node to place (A) int - the ID of the node after which the above is placed (B)] * N

Places node A in the same group as node B, to execute immediately after node B.

3.26 /d_free delete synth definition

[string - synth def name] * N

Removes a synth definition once all synths using it have ended.

3.27 /n_set Set a node's control value(s)

int - node ID [int or string - a control index or name float - a control value] * N

Takes a list of pairs of control indices and values and sets the controls to those values. If the node is a group, then it sets the controls of every node in the group.

```
(with-sc3
 (lambda (fd)
   (letc ((f 440)
          (a 0.1))
     (send-synth fd "sin" (out 0 (mul (sin-osc ar f 0) a))))
   (send fd (s-new0 "sin" 1001 add-to-tail 1))))

(with-sc3
 (lambda (fd)
   (send fd (n-set1 1001 "f" 1280))))

(with-sc3
 (lambda (fd)
   (send fd (n-set 1001 (list (tuple2 "f" (random 60 900))
                              (tuple2 "a" (random 0.05 0.25)))))))
```

3.28 /b_write Write sound file data.

int - buffer number string - path name of a sound file. string - header format. string - sample format. int - number of frames to write (optional. default = -1, see below) int - starting frame in buffer (optional. default = 0) int - leave file open (optional. default = 0) bytes - an OSC message to execute upon completion. (optional)

Write a buffer as a sound file. Header format is one of: "aiff", "next", "wav", "ircam"", "raw" Sample format is one of: "int8", "int16", "int24", "int32", "float", "double", "mulaw", "alaw"

Not all combinations of header format and sample format are possible. If number of frames is less than zero, all samples from the starting frame to the end of the buffer are written. If opening a file to be used by DiskOut ugen then you will want to set "leave file open" to one, otherwise set it to zero. If "leave file open" is set to one then the file is created, but no frames are written until the DiskOut ugen does so.

Asynchronous. Replies to sender with /done when complete.

3.29 /b_getn Get ranges of sample value(s)

int - buffer number [int - starting sample index int - number of sequential samples to get (M)] * N

Get contiguous ranges of samples. Replies to sender with the corresponding /b_setn command. This is only meant for getting a few samples, not whole buffers or large sections.

3.30 /b_fill Fill ranges of sample value(s)

int - buffer number [int - sample starting index int - number of samples to fill (M) float - value] * N

Set contiguous ranges of sample indices to single values. For each range, the starting sample index is given followed by the number of samples to change, followed by the value to fill. This is only meant for setting a few samples, not whole buffers or large sections.

3.31 /s_noid Auto-reassign synth's ID to a reserved value

[int - synth ID] * N

This command is used when the client no longer needs to communicate with the synth and wants to have the freedom to reuse the ID. The server will reassign this synth to a reserved

negative number. This command is purely for bookkeeping convenience of the client. No notification is sent when this occurs.

3.32 /d_recv Receive a synth definition file

bytes - buffer of data. bytes - an OSC message to execute upon completion. (optional)

Loads a file of synth definitions from a buffer in the message. Resident definitions with the same names are overwritten.

Asynchronous. Replies to sender with /done when complete.

3.33 /u_cmd send a command to a unit generator

int - node ID int - unit generator index string - command name ...any arguments

Sends all arguments following the command name to the unit generator to be performed. Commands are defined by unit generator plug ins.

3.34 /b_allocRead Allocate buffer space and read a sound file.

int - buffer number string - path name of a sound file. int - starting frame in file (optional. default = 0) int - number of frames to read (optional. default = 0, see below) bytes - an OSC message to execute upon completion. (optional)

Allocates buffer to number of channels of file and number of samples requested, or fewer if sound file is smaller than requested. Reads sound file data from the given starting frame in the file. If the number of frames argument is less than or equal to zero, the entire file is read.

Asynchronous. Replies to sender with /done when complete.

3.35 /n_map Map a node's controls to read from a bus

int - node ID [int or string - a control index or name int - control bus index] * N

Takes a list of pairs of control names or indices and bus indices and causes those controls to be read continuously from a global control bus instead of responding to n_set, n_setn and n_fill commands. If the node is a group, then it maps the controls of every node in the group. If the control bus index is -1 then any current mapping is undone and control reverts to normal.

3.36 /status Query the status

No arguments.

Replies to sender with the following message.

/status.reply int - 1. unused. int - number of unit generators. int - number of synths. int - number of groups. int - number of loaded synth definitions. float - average percent CPU usage for signal processing float - peak percent CPU usage for signal processing double - nominal sample rate double - actual sample rate

3.37 /g_new Create a new group

[int - new group ID int - add action (0,1,2, 3 or 4 see below) int - add target ID] * N

Create a new group and add it to the tree of nodes.

There are four ways to add the group to the tree as determined by the add action argument which is defined as follows (the same as for "/s_new"):

0 - add the new group to the the head of the group specified by the add target ID.

1 - add the new group to the the tail of the group specified by the add target ID.

2 - add the new group just before the node specified by the add target ID.

3 - add the new group just after the node specified by the add target ID.

4 - the new node replaces the node specified by the add target ID. The target node is freed.

Multiple groups may be created in one command by adding arguments.

3.38 /c_get Get bus value(s)

[int - a bus index] * N

Takes a list of buses and replies to sender with the corresponding /c_set command.

3.39 /b_set Set sample value(s)

int - buffer number [int - a sample index float - a sample value] * N

Takes a list of pairs of sample indices and values and sets the samples to those values.

3.40 /b_zero Zero sample data

int - buffer number bytes - an OSC message to execute upon completion. (optional)

Sets all samples in the buffer to zero.

Asynchronous. Replies to sender with /done when complete.

3.41 /tr A trigger message

int - node ID int - trigger ID float - trigger value

This command is the mechanism that synths can use to trigger events in clients.

The node ID is the node that is sending the trigger. The trigger ID and value are determined by inputs to the SendTrig unit generator which is the originator of this message.

3.42 /dumpOSC Display incoming OSC messages

int - code

Turns on and off printing of the contents of incoming Open Sound Control messages. This is useful when debugging your command stream.

The values for the code are as follows: 0 - turn dumping OFF. 1 - print the parsed contents of the message. 2 - print the contents in hexadecimal. 3 - print both the parsed and hexadecimal representations of the contents.

3.43 /n_setn Set ranges of a node's control value(s)

int - node ID [int or string - a control index or name int - number of sequential controls to change (M) [float - a control value] * M] * N

Set contiguous ranges of control indices to sets of values. For each range, the starting control index is given followed by the number of controls to change, followed by the values. If the node is a group, then it sets the controls of every node in the group.

3.44 /b_free Free buffer data.

int - buffer number bytes - an OSC message to execute upon completion. (optional)

Frees buffer space allocated for this buffer.

Asynchronous. Replies to sender with /done when complete.

3.45 /notify Register to receive notifications from server

int - one to receive notifications, zero to stop receiving them.

If argument is one, server will remember your return address and send you notifications. if argument is zero, server will stop sending you notifications.

Asynchronous. Replies to sender with /done when complete.

3.46 /b_query

[int - buffer number] * N

Responds to the sender with a /b_info message. The arguments to /b_info are as follows:

[int - buffer number int - number of frames int - number of channels float - sample rate] * N

```
(with-sc3
 (lambda (fd)
   (async fd (/b_alloc 10 6 1))
   (async fd (/b_query 10))))
```

3.47 /g_tail Add node to tail of group

[int - group ID int - node ID] * N

Adds the node to the tail (last to be executed) of the group.

3.48 /clearSched Clear all scheduled bundles.

Removes all bundles from the scheduling queue.

3.49 /g_freeAll Delete all nodes in a group.

[int - group ID] * N

Frees all nodes in the group. A list of groups may be specified.

3.50 /g_head Add node to head of group

[int - group ID int - node ID] * N

Adds the node to the head (first to be executed) of the group.

3.51 /c_getn Get ranges of bus value(s)

[int - starting bus index int - number of sequential buses to get (M)] * N

Get contiguous ranges of buses. Replies to sender with the corresponding /c_setn command.

3.52 /quit Quit program

No arguments.

Exits the synthesis server.

Asynchronous. Replies to sender with /done just before completion.

3.53 see g-new

3.54 /c_fill Fill ranges of bus value(s)

[int - starting bus index int - number of buses to fill (M) float - value] * N

Set contiguous ranges of buses to single values. For each range, the starting sample index is given followed by the number of buses to change, followed by the value to fill.

4 rsc3: ugens

Contents

filters

granular

disk-io

information

analysis

triggers

oscillators

io

composite

controls

ternary-ops

binary-ops

noise

unary-ops

fft

delays

physical-models

demand

chaos

panners

envelopes

4.1 (Rlpf in freq rq)

A resonant low pass filter.

```
(let* ((f1 (x-line kr 0.7 300 20 remove-synth))
       (f2 (mul-add (f-sin-osc kr f1 0) 3600 4000)))
```

```
  (audition
   (out 0 (rlpf (mul (saw ar 200) 0.1) f2 0.2))))
```

4.2 (fold in lo hi)

fold a signal outside given thresholds.

This differs from the BinaryOpUGen fold2 in that it allows one to set both low and high thresholds.

in - signal to be foldped lo - low threshold of foldping hi - high threshold of foldping

```
(let ((o (mul (sin-osc ar 440 0) 0.2))
      (l (rand -0.175 -0.025))
      (r (rand 0.025 0.175)))
  (audition (out 0 (fold o l r))))
```

lo and hi are i-rate only.

```
(let ((o (mul (sin-osc ar 440 0) 0.2))
      (x (mouse-x kr -0.175 -0.025 1 0.1))
      (y (mouse-y kr 0.025 0.175 1 0.1)))
  (audition (out 0 (fold o x y))))
```

4.3 (formlet in freq attackTime decayTime)

FOF-like filter

```
(let ((i (impulse ar 20 0.5)))
  (audition (out 0 (formlet i 1000 0.01 0.1))))


(let* ((f (x-line kr 10 400 8 remove-synth))
       (i (mul (blip ar f 1000) 0.1)))
  (audition (out 0 (formlet i 1000 0.01 0.1))))
```

Modulating formant frequency.

```
(let ((i (mul (blip ar (mul-add (sin-osc kr 5 0) 20 300) 1000) 0.1))
      (f (x-line kr 1500 700 8 remove-synth)))
  (audition (out 0 (formlet i f 0.005 0.04))))
```

4.4 (lin-lin in srclo srchi dstlo dsthi)

Map a linear range to another linear range.

in - input to convert - kr, ar srclo - lower limit of input range - ir srchi - upper limit of input range - ir dstlo - lower limit of output range - ir dsthi - upper limit of output range - ir

```
(audition
 (out 0 (mul (sin-osc ar (lin-lin (mouse-x kr 0 1 0 0.1) 0 1 440 660) 0)
             (lin-lin (mouse-y kr 0 1 0 0.1) 0 1 0.01 0.25))))
```

4.5 (fos in a0 a1 b1)

First order filter section.

Same as one-pole.

```
(let ((x (mul (lf-tri ar 0.4 0) 0.99))
      (i (mul (lf-saw ar 200 0) 0.2)))
  (audition (out 0 (fos i (sub 1 (u:abs x)) 0 x))))
```

Same as one-zero

```
(let ((x (mul (lf-tri ar 0.4 0) 0.99))
      (i (mul (lf-saw ar 200 0) 0.2)))
  (audition (out 0 (fos i (sub 1 (u:abs x)) x 0))))
```

4.6 (bpf in freq rq)

Second order Butterworth bandpass filter

in - input signal to be processed freq - cutoff frequency in Hertz. rq - the reciprocal of Q. bandwidth / cutoffFreq.

```
(let* ((f1 (x-line kr 0.7 300 20 remove-synth))
       (f2 (mul-add (f-sin-osc kr f1 0) 3600 4000)))
  (audition (out 0 (bpf (mul (saw ar 200) 0.25) f2 0.3))))


(let* ((f1 (mouse-x kr 220 440 0 0.1))
       (f2 (mce2 f1 (sub 550 f1)))
       (rq (mouse-y kr 0 0.01 0 0.1)))
  (audition (out 0 (bpf (white-noise ar) f2 rq))))
```

4.7 (wrap in lo hi)

wrap a signal outside given thresholds.

This differs from the BinaryOpUGen wrap2 in that it allows one to set both low and high thresholds.

in - signal to be wrapped lo - low threshold of wrapping hi - high threshold of wrapping

```
(let ((o (mul (sin-osc ar 440 0) 0.2))
      (l (rand -0.175 -0.025))
      (r (rand 0.025 0.175)))
  (audition (out 0 (wrap o l r))))
```

lo and hi are i-rate only.

```
(let ((o (mul (sin-osc ar 440 0) 0.2))
      (x (mouse-x kr -0.175 -0.025 1 0.1))
      (y (mouse-y kr 0.025 0.175 1 0.1)))
  (audition (out 0 (wrap o x y))))
```

4.8 (bpz2 in)

Two zero fixed midpass. This filter cuts out 0 Hz and the Nyquist frequency.

```
(audition (out 0 (bpz2 (mul (white-noise ar) 0.25))))
```

4.9 (amp-comp freq root exp)

Basic psychoacoustic amplitude compensation.

Implements the (optimized) formula: compensationFactor = (root / freq) ** exp. Higher frequencies are normally perceived as louder, which amp-comp compensates.

See also amp-compA

freq - input frequency value. For freq == root, the output is 1.0.

root - root freq relative to which the curve is calculated (usually lowest freq) default value: C (60.midicps)

exp - exponent. how steep the curve decreases for increasing freq (see plots below). default value 0.3333

Note that for frequencies very much smaller than root the amplitudes can become very high. in this case limit the freq with freq.max(minval), or use amp-compA.

compare a sine without compensation with one that uses amplitude compensation

```
(let* ((x (mouse-x kr 300 15000 1 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (o (mul (sin-osc ar x 0) 0.1))
       (c (amp-comp x 300 0.333)))
  (audition (out 0 (mce2 (mul o y) (mul3 o (sub 1 y) c)))))
```

different sounds cause quite different loudness perception, and the desired musical behavior can vary, so the exponent can be tuned:

```
(let* ((x (mouse-x kr 300 15000 1 0.1))
       (o (mul (pulse ar x 0.5) 0.1))
       (c (amp-comp x 300 1.3)))
  (audition (out 0 (mul o c))))
```

amplitude compensation in frequency modulation

```
(let* ((x (mouse-x kr 300 15000 1 0.1))
       (y (mouse-y kr 3 200 1 0.1))
       (m (mul x (mul-add (sin-osc ar y 0) 0.5 1)))
       (a (amp-comp m 300 0.333))
       (c (mul3 (sin-osc ar m 0) 0.1 a)))
  (audition (out 0 c)))
```

4.10 (lag2 in lagTime)

lag2 is the same as (lag kr (Lag kr in time) time).

```
(let* ((x (mouse-x kr 220 440 0 0.1))
       (f (mce2 x (lag2 x 1))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.11 (one-zero in coef)

One zero filter

```
(audition
 (out 0 (one-zero (mul (white-noise ar) 0.5) 0.5)))
```

```
(audition
 (out 0 (one-zero (mul (white-noise ar) 0.5) -0.5)))


(audition
 (out 0 (one-zero (mul (white-noise ar) 0.5)
                  (line kr -0.5 0.5 10 remove-synth))))
```

4.12 (klank in freqScale freqOffset decayScale spec)

klank is a bank of fixed frequency resonators which can be used to simulate the resonant modes of an object. Each mode is given a ring time, which is the time for the mode to decay by 60 dB.

The UGen assistant klank-data can help create the 'spec' entry. Note that the SC3 language reorders the inputs, the RSC client does not.

input - the excitation input to the resonant filter bank.

freqscale - a scale factor multiplied by all frequencies at initialization time.

freqoffset - an offset added to all frequencies at initialization time.

decayscale - a scale factor multiplied by all ring times at initialization time.

```
(let ((i (mul (impulse ar 2 0) 0.1))
      (d (klank-data '(800 1071 1153 1723)
                     (replicate 5 1)
                     (replicate 5 1))))
  (audition (out 0 (klank i 1 0 1 d))))


(let ((i (mul (dust ar 8) 0.1))
      (d (klank-data '(800 1071 1353 1723)
                     (replicate 4 1)
                     (replicate 4 1))))
  (audition (out 0 (klank i 1 0 1 d))))


(let ((i (mul (pink-noise ar) 0.007))
      (d (klank-data '(800 1071 1353 1723)
                     (replicate 4 1)
                     (replicate 4 1))))
  (audition (out 0 (klank i 1 0 1 d))))


(let ((i (mul (pink-noise ar) (mce2 0.007 0.007)))
```

```
      (d (klank-data '(200 671 1153 1723)
                     (replicate 4 1)
                     (replicate 4 1))))
  (audition (out 0 (klank i 1 0 1 d))))

(let ((i (mul (decay (impulse ar 4 0) 0.03) (mul (clip-
noise ar) 0.01)))
      (d (klank-data (replicate-m 12 (rand 800 4000))
                     (replicate 12 1)
                     (replicate-m 12 (rand 0.1 2)))))
  (audition (out 0 (klank i 1 0 1 d))))
```

4.13 (lpz2 ar in)

Two zero fixed lowpass filter

```
(audition
 (out 0 (lpz2 (mul (white-noise ar) 0.25))))
```

4.14 (lin-exp in srclo srchi dstlo dsthi)

Map a linear range to an exponential range.

in - input to convert - kr, ar srclo - lower limit of input range - ir srchi - upper limit of input range - ir dstlo - lower limit of output range - ir dsthi - upper limit of output range - ir

```
(audition
 (out 0 (mul (sin-osc ar (lin-exp (mouse-x kr 0 1 0 0.1) 0 1 440 660) 0)
             (lin-exp (mouse-y kr 0 1 0 0.1) 0 1 0.01 0.25))))
```

4.15 (hpz2 in)

Two zero fixed highpass filter.

```
(audition (out 0 (hpz2 (mul (white-noise ar) 0.25))))
```

4.16 (leak-dc in coef)

Remove DC. This filter removes a DC offset from a signal.

in - input signal coef - leak coefficient

```
(let ((a (mul (lf-pulse ar 800 0.5 0.5) 0.1)))
  (audition (out 0 (mce2 a (leak-dc a 0.995)))))
```

4.17 (hpf in freq)

Second order Butterworth highpass filter.

```
(let* ((i (mul (saw ar 200) 0.1))
       (f1 (x-line kr 0.7 300 20 do-nothing))
       (f2 (mul-add (f-sin-osc kr f1 0) 3600 4000)))
  (audition (out 0 (mul (hpf i f2) 5))))
```

4.18 (sos in a0 a1 a2 b1 b2)

Second order filter section (biquad). A standard second order filter section. Filter coefficients are given directly rather than calculated for you.

Same as two-pole

```
(let* ((theta (line kr (* 0.2 pi) pi 5 remove-synth))
       (rho (line kr 0.6 0.99 5 remove-synth))
       (b1 (mul 2 (mul rho (u:cos theta))))
       (b2 (neg (squared rho))))
  (audition (out 0 (sos (lf-saw ar 200 0.1) 1 0 0 b1 b2))))
```

4.19 (normalizer in level dur)

Flattens dynamics.

```
(let ((z (mul (decay2 (impulse ar 8 (mul (lf-
saw kr 0.25 -0.6) 0.7))
                      0.001
                      0.3)
              (f-sin-osc ar 500 0))))
  (audition (out 0 (mce2 z (normalizer z 0.4 0.01)))))
```

4.20 (two-zero in freq radius)

Two zero filter

```
(audition
 (out 0 (two-zero (mul (white-noise ar) 0.125)
                  (x-line kr 20 20000 8 remove-synth)
                  1)))
```

4.21 (clip in lo hi)

clip 'in' to lie between 'lo' and 'hi', which are i-rate inputs.

```
(audition (out 0 (clip (mul (sin-osc ar 440 0) 0.4) -0.25 0.25)))
```

4.22 (limiter input level lookAheadTime)

peak limiter. Limits the input amplitude to the given level. limiter will not overshoot like compander will, but it needs to look ahead in the audio. Thus there is a delay equal to twice the lookAheadTime. limiter, unlike compander, is completely transparent for an in range signal.

```
(let* ((t (impulse ar 8 (mul (lf-saw kr 0.25 -0.6) 0.7)))
       (i (mul (decay2 t 0.001 0.3) (f-sin-osc ar 500 0))))
  (audition (out 0 (mce2 (mul i 0.1) (limiter i 0.2 0.01)))))
```

4.23 (median length in)

median filter.

Signal with impulse noise.

```
(audition
 (out 0 (median 3 (add (mul (saw ar 500) 0.1) (mul (dust2 ar 100) 0.9)))))
```

The median length can be increased for longer duration noise.

```
(audition
 (out 0 (median 5 (add (mul (saw ar 500) 0.1) (lpz1 (mul (dust2 ar 100) 0.9))))))
```

Long median filters begin chopping off the peaks of the waveform

```
(audition
 (out 0 (let ((x (mul (sin-osc ar 1000 0) 0.2)))
          (mce2 x (median 31 x)))))
```

Another noise reduction application. Use median filter for high frequency noise. Use leak-dc for low frequency noise.

```
(audition
 (out 0 (let* ((s0 (mul-add (white-noise ar) 0.1 (mul (sin-
osc ar 800 0) 0.1)))
               (s1 (median 31 s0)))
          (leak-dc s1 0.9))))
```

4.24 (lag3 in lagTime)

lag3 is the same as (lag (Lag (Lag in time) time) time).

```
(let* ((x (mouse-x kr 220 440 0 0.1))
       (f (mce2 x (lag3 x 1))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.25 (amp-compA freq root minAmp rootAmp)

ANSI A-weighting curve.

Basic psychoacoustic amplitude compensation

Higher frequencies are normally perceived as louder, which amp-compA compensates. Following the measurings by Fletcher and Munson, the ANSI standard describes a function for loudness vs. frequency. Note that this curve is only valid for standardized amplitude. For a simpler but more flexible curve, see amp-comp.

freq - input frequency value. For freq == root, the output is rootAmp. (default freq 0 Hz)

root - root freq relative to which the curve is calculated (usually lowest freq) (default 0 Hz) default value: C (60.midicps)

minAmp - amplitude at the minimum point of the curve (around 2512 Hz) (default -10dB)

rootAmp - amplitude at the root frequency. (default 1) apart from freq, the values are not modulatable

compare a sine without compensation with one that uses amplitude compensation

```
(let* ((x (mouse-x kr 300 15000 1 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (o (mul (sin-osc ar x 0) 0.1))
       (c (amp-comp-a x 300 (db-amp -10) 1)))
  (audition (out 0 (mce2 (mul o y) (mul3 o (sub 1 y) c)))))
```

adjust the minimum and root amp (in this way one can flatten out the curve for higher amplitudes)

```
(let* ((x (mouse-x kr 300 18000 1 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (o (mul (formant ar 300 x 20) 0.1))
       (c (amp-comp-a x 300 0.6 0.3)))
  (audition (out 0 (mce2 (mul o y) (mul3 o (sub 1 y) c)))))
```

amplitude compensation in frequency modulation (using Fletscher-Munson curve)

```
(let* ((x (mouse-x kr 300 15000 1 0.1))
       (y (mouse-y kr 3 200 1 0.1))
       (m (mul x (mul-add (sin-osc ar y 0) 0.5 1)))
       (a (amp-comp-a m 300 (db-amp -10) 1))
       (c (mul3 (sin-osc ar m 0) 0.1 a)))
  (audition (out 0 c)))
```

4.26 (two-pole in freq radius)

Two pole filter. This provides lower level access to setting of pole location. For general purposes resonz is better.

```
(audition
 (out 0 (two-pole (mul (white-noise ar) 0.005) 2000 0.95)))

(audition
 (out 0 (two-pole (mul (white-noise ar) 0.005)
                  (x-line kr 800 8000 8 remove-synth)
                  0.95)))
```

4.27 (lpf in freq)

Second order Butterworth lowpass filter.

```
(audition
 (let ((f (x-line kr 0.7 300 20 remove-synth)))
   (out 0 (lpf (mul (saw ar 200) 0.1)
               (mul-add (f-sin-osc kr f 0) 3600 4000)))))
```

4.28 (hpz1 in)

Two point difference filter.

```
(audition (out 0 (hpz1 (mul (white-noise ar) 0.25))))
```

4.29 (freq-shift input shift phase)

freq-shift implements single sideband amplitude modulation, also known as frequency shifting, but not to be confused with pitch shifting. Frequency shifting moves all the components of a signal by a fixed amount but does not preserve the original harmonic relationships.

input - audio input shift - amount of shift in cycles per second phase - phase of the frequency shift (0 - 2pi)

shifting a 100Hz tone by 1 Hz rising to 500Hz

```
(let ((i (sin-osc ar 100 0))
      (s (x-line kr 1 500 5 remove-synth)))
  (audition (out 0 (mul (freq-shift i s 0) 0.1))))
```

shifting a complex tone by 1 Hz rising to 500Hz

```
(let ((i (klang ar 1 0 (klang-data (list 101 303 606 808)
                                   (replicate 4 1)
                                   (replicate 4 1))))
      (s (x-line kr 1 500 5 remove-synth)))
  (audition (out 0 (mul (freq-shift i s 0) 0.1))))
```

modulating shift and phase

```
(let ((i (sin-osc ar 10 0))
      (s (mul (lf-noise2 ar 0.3) 1500))
      (p (lin-lin (sin-osc ar 500 0) -1 1 0 (* 2 pi))))
  (audition (out 0 (mul (freq-shift i s p) 0.1))))
```

shifting bandpassed noise

```
(let ((i (bpf (white-noise ar) 1000 0.001))
      (s (mul (lf-noise0 ar 5.5) 1000)))
  (audition (out 0 (mul (freq-shift i s 0) 32))))
```

4.30 (lpz1 ar in)

Two point average filter

```
(audition
 (out 0 (lpz1 (mul (white-noise ar) 0.25))))
```

4.31 (moog-ff in freq gain reset)

Moog VCF implementation, designed by Federico Fontana. A digital implementation of the Moog VCF (filter).

in - the input signal freq - the cutoff frequency gain - the filter resonance gain, between zero and 4 reset - when greater than zero, this will reset the state of the digital filters at the beginning of a computational block.

The design of this filter is described in the conference paper Fontana, F. (2007) Preserving the Digital Structure of the Moog VCF. in Proc. ICMC07, Copenhagen, 25-31 August 2007

```
(let ((n (mul (white-noise ar) 0.1))
      (y (mouse-y kr 100 10000 1 0.1))
      (x (mouse-x kr 0 4 0 0.1)))
  (audition (out 0 (moog-ff n y x 0))))


(let* ((p (pulse ar (mce2 40 121) (mce2 0.3 0.7)))
       (f0 (lin-lin (lf-noise0 kr 0.43) -1 1 0.001 2.2))
       (f1 (lin-lin (sin-osc kr f0 0) -1 1 30 4200))
       (y (mouse-y kr 1 4 0 0.1)))
  (audition (out 0 (moog-ff p f1 (mul 0.83 y) 0))))
```

4.32 (resonz in freq bwr)

Resonant filter.

A two pole resonant filter with zeroes at z = +/- 1. Based on K. Steiglitz, "A Note on Constant-Gain Digital Resonators," Computer Music Journal, vol 18, no. 4, pp. 8-10, Winter 1994. The reciprocal of Q is used rather than Q because it saves a divide operation inside the unit generator.

in - input signal to be processed freq - resonant frequency in Hertz rq - bandwidth ratio (reciprocal of Q). rq = bandwidth / centerFreq

```
(audition (out 0 (resonz (mul (white-noise ar) 0.5) 2000 0.1)))
```

Modulate frequency

```
(let ((f (x-line kr 1000 8000 10 remove-synth)))
  (audition (out 0 (resonz (mul (white-noise ar) 0.5) f 0.05))))
```

Modulate bandwidth

```
(let ((rq (x-line kr 1 0.001 8 remove-synth)))
  (audition (out 0 (resonz (mul (white-noise ar) 0.5) 2000 rq))))
```

Modulate bandwidth opposite direction

```
(let ((rq (x-line kr 0.001 1 8 remove-synth)))
  (audition (out 0 (resonz (mul (white-noise ar) 0.5) 2000 rq))))
```

random resonator at a random location, run as often as you like...

```
(let ((freq (choose (map (lambda (z) (* z 120)) (enum-from-
to 1 16))))
      (bw 1/4)
      (gain 8))
  (audition (out 0 (pan2 (resonz (white-noise ar) freq (/ bw freq))
                         (rand -1 1)
                         gain))))
```

4.33 (brz2 in)

A two zero fixed midcut filter. A special case fixed filter. Implements the formula:

out(i) = 0.5 * (in(i) + in(i-2))

This filter cuts out frequencies around 1/2 of the Nyquist frequency.

Compare:

```
(audition (out 0 (mul (white-noise ar) 0.15)))

(audition (out 0 (brz2 (mul (white-noise ar) 0.15))))
```

4.34 (Rhpf in freq rq)

A resonant high pass filter.

```
(audition
 (out
  0
  (rhpf (mul (saw ar 200) 0.1)
        (mul-add (f-sin-osc kr (x-line kr 0.7 300 20 remove-
synth) 0) 3600 4000)
        0.2)))
```

4.35 (brf in freq rq)

Second order Butterworth band reject filter.

```
(let* ((f1 (x-line kr 0.7 300 20 remove-synth))
       (f2 (mul-add (f-sin-osc kr f1 0) 3600 4000)))
  (audition (out 0 (brf (mul (saw ar 200) 0.1) f2 0.3))))
```

4.36 (one-pole in coef)

A one pole filter. Implements the formula: out(i) = ((1 - abs(coef)) * in(i)) + (coef * out(i-1)).

in - input signal to be processed coef - feedback coefficient. Should be between -1 and +1

```
(audition
 (out 0 (one-pole (mul (white-noise ar) 0.5) 0.95)))


(audition
 (out 0 (one-pole (mul (white-noise ar) 0.5) -0.95)))


(audition
 (out 0 (one-pole (mul (white-noise ar) 0.5)
                  (line kr -0.99 0.99 10 remove-synth))))
```

4.37 (lag in lagTime)

A simple averaging filter.

```
(let* ((x (mouse-x kr 220 440 0 0.1))
       (f (mce2 x (lag x 1))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.38 (ringz in freq decayTime)

Ringing filter. This is the same as resonz, except that instead of a resonance parameter, the bandwidth is specified in a 60dB ring decay time. One ringz is equivalent to one component of the klank UGen.

```
(audition
 (out 0 (ringz (mul (dust ar 3) 0.3) 2000 2)))
```

```
(audition
 (out 0 (ringz (mul (white-noise ar) 0.005) 2000 0.5)))
```

Modulate frequency

```
(audition
 (out 0 (ringz (mul (white-noise ar) 0.005)
               (x-line kr 100 3000 10 do-nothing)
               0.5)))


(audition
 (out 0 (ringz (mul (impulse ar 6 0) 0.3)
               (x-line kr 100 3000 10 do-nothing)
               0.5)))
```

Modulate ring time

```
(audition
 (out 0 (ringz (mul (impulse ar 6 0) 0.3)
               2000
               (x-line kr 4 0.04 8 do-nothing))))
```

Modulate ring time opposite direction

```
(audition
 (out 0 (ringz (mul (impulse ar 6 0) 0.3)
               2000
               (x-line kr 0.04 4 8 do-nothing))))


(audition
 (out 0 (let ((n (mul (white-noise ar) 0.001)))
          (mix-fill
           10
           (lambda (_)
             (let ((f (x-line kr (rand 100 5000) (rand 100 5000) 20 do-
nothing)))
               (ringz n f 0.5)))))))
```

4.39 (dyn-klank in freqScale freqOffset decayScale spec)

Dynklank is a bank of frequency resonators which can be used to simulate the resonant modes of an object. Each mode is given a ring time, which is the time for the mode to decay by 60 dB.

Unlike klank, the parameters in specificationsArrayRef can be changed after it has been started.

```
(let ((i (mul (impulse ar 2 0) 0.1))
      (d (klank-data '(800 1071 1153 1723)
                     (replicate 4 1)
                     (replicate 4 1))))
  (audition (out 0 (dyn-klank i 1 0 1 d))))


(let ((i (mul (dust ar 8) 0.1))
      (d (klank-data '(800 1071 1353 1723)
                     (replicate 4 1)
                     (replicate 4 1))))
  (audition (out 0 (dyn-klank i 1 0 1 d))))


(let* ((i (mul (impulse ar 3 0) 0.1))
       (f (list 800 1071 1153 1723))
       (r (list 1 1 1 1))
       (x (mouse-x kr 0.5 2 1 0.1))
       (y (mouse-y kr 0.1 10 1 0.1))
       (d (klank-data (map (lambda (e) (mul e x)) f)
                      (replicate 4 1)
                      (map (lambda (e) (mul e y)) r))))
  (audition (out 0 (dyn-klank i 1 0 1 d))))


(let* ((i (lambda (f)
            (mul (impulse ar (lin-lin (lf-
noise0 kr f) -1 1 3 12) 0) 0.1)))
       (t (lambda (i d l r)
            (map (lambda (e) (mul e (t-rand l r i))) d)))
       (d (lambda (i f r)
            (klank-data (t i f 0.5 2)
                        (replicate 4 1)
                        (t i r 0.1 10))))
       (f1 (list 800 1071 1153 1723))
       (f2 (list 786 1083 1169 1715))
       (r1 (list 1 0.95 0.75 1.25))
       (r2 (list 1 1.35 0.95 1.15))
       (i1 (i 1.5))
       (i2 (i 1.25)))
  (audition (out 0 (mce2 (dyn-klank i1 1 0 1 (d i1 f1 r1))
                         (dyn-klank i2 1 0 1 (d i2 f2 r2))))))
```

4.40 (Grainin nc tr dur in pan envbuf)

Granulate an input signal

nc - the number of channels to output. If 1, mono is returned and pan is ignored.

tr - a kr or ar trigger to start a new grain. If ar, grains after the start of the synth are sample accurate.

The following args are polled at grain creation time

dur - size of the grain.

in - the input to granulate

pan - a value from -1 to 1. Determines where to pan the output in the same manner as pan-az.

envbuf - the buffer number containing a singal to use for the grain envelope. -1 uses a built-in Hanning envelope.

```
(let* ((x (mouse-x kr -0.5 0.5 0 0.1))
       (y (mouse-y kr 5 25 0 0.1))
       (n (pink-noise ar))
       (t (impulse kr y 0)))
  (audition (out 0 (mul (grain-in 2 t 0.1 n x -1) 0.1))))
```

4.41 (grain-buf nc tr dur sndbuf rate pos interp pan envbuf)

Granular synthesis with sound stored in a buffer

nc - the number of channels to output. If 1, mono is returned and pan is ignored.

tr - a kr or ar trigger to start a new grain. If ar, grains after the start of the synth are sample accurate.

The following args are polled at grain creation time

dur - size of the grain.

sndbuf - the buffer holding an audio signal

rate - the playback rate of the sampled sound

pos - the playback position for the grain to start with (0 is beginning, 1 is end of file)

interp - the interpolation method used for pitchshifting grains. 1 = no interpolation. 2 = linear. 4 = cubic interpolation (more computationally intensive).

pan - a value from -1 to 1. Determines where to pan the output in the same manner as pan-az.

envb - the buffer number containing a singal to use for the grain envelope. -1 uses a built-in Hanning envelope.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 10 "/home/rohan/audio/metal.wav" 0 0))
   (let ((g (letc ((gate 1)
                   (amp 1)
                   (sndbuf 0)
                   (envbuf -1))
              (let* ((x (mouse-x kr -1 1 0 0.1))
                     (y (mouse-y kr 10 45 0 0.1))
                     (i (impulse kr y 0))
                     (r (lin-lin (lf-noise1 kr 500) -1 1 0.5 2))
                     (p (lin-lin (lf-noise2 kr 0.1) -1 1 0 1)))
                (out 0 (grain-buf 2 i 0.1 sndbuf r p 2 x envbuf))))))
     (send-synth fd "g" g)
     (send fd (s-new2 "g" -1 add-to-tail 1 "sndbuf" 10 "envbuf" -1)))))
```

4.42 (grain-sin nc tr dur freq pan envbuf)

Granular synthesis with sine tones

nc - the number of channels to output. If 1, mono is returned and pan is ignored.

tr - a kr or ar trigger to start a new grain. If ar, grains after the start of the synth are sample accurate.

The following args are polled at grain creation time

dur - size of the grain.

freq - the input to granulate

pan - a value from -1 to 1. Determines where to pan the output in the same manner as pan-az.

envbuf - the buffer number containing a singal to use for the grain envelope. -1 uses a built-in Hanning envelope.

```
(let* ((x (mouse-x kr -0.5 0.5 0 0.1))
       (y (mouse-y kr 0 400 0 0.1))
       (n (white-noise kr))
       (f (add 440 (mul n y)))
       (t (impulse kr 10 0)))
```

```
(audition (out 0 (mul (grain-sin 2 t 0.1 f x -1) 0.1))))
```

4.43 (warp1 nc buf ptr freqScale windowSize envbuf overlaps windowrandRatio interp)

Warp a buffer with a time pointer

inspired by Chad Kirby's SuperCollider2 warp1 class, which was inspired by Richard Karpen's sndwarp for CSound. A granular time strecher and pitchshifter.

nc - the number of channels in the soundfile used in bufnum.

buf - the buffer number of a mono soundfile.

ptr - the position in the buffer. The value should be between 0 and 1, with 0 being the begining of the buffer, and 1 the end.

freqScale - the amount of frequency shift. 1.0 is normal, 0.5 is one octave down, 2.0 is one octave up. Negative values play the soundfile backwards.

windowSize - the size of each grain window.

envbuf - the buffer number containing a singal to use for the grain envelope. -1 uses a built-in Hanning envelope.

overlaps - the number of overlaping windows.

windowrandRatio - the amount of randomness to the windowing function. Must be between 0 (no randomness) to 1.0 (probably to random actually)

interp - the interpolation method used for pitchshifting grains. 1 = no interpolation. 2 = linear. 4 = cubic interpolation (more computationally intensive).

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 10 "/home/rohan/audio/metal.wav" 0 0))))

(let* ((p (lin-lin (lf-saw kr 0.05 0) -1 1 0 1))
       (x (mouse-x kr 0.5 2 0 0.1))
       (w (warp1 1 10 p x 0.1 -1 8 0.1 2)))
  (audition (out 0 w)))
```

4.44 (grain-fm nc tr dur carfreq modfreq index pan envbuf)

Granular synthesis with frequency modulated sine tones

nc - the number of channels to output. If 1, mono is returned and pan is ignored.

tr - a kr or ar trigger to start a new grain. If ar, grains after the start of the synth are sample accurate.

The following args are polled at grain creation time

dur - size of the grain.

carfreq - the carrier freq of the grain generators internal oscillator

modfreq - the modulating freq of the grain generators internal oscillator

index - the index of modulation

pan - a value from -1 to 1. Determines where to pan the output in the same manner as pan-az.

envbuf - the buffer number containing a singal to use for the grain envelope. -1 uses a built-in Hanning envelope.

```
(let* ((x (mouse-x kr -0.5 0.5 0 0.1))
       (y (mouse-y kr 0 400 0 0.1))
       (n (white-noise kr))
       (fd (add 440 (mul n y)))
       (t (impulse kr 10 0))
       (i (lin-lin (lf-noise1 kr 500) -1 1 1 10)))
  (audition (out 0 (mul (grain-fm 2 t 0.1 fd 200 i x -1) 0.1))))
```

4.45 (disk-in num-channels rate bufnum)

Continously play a soundfile from disk. This requires a buffer to be preloaded with one buffer size of sound. The buffer size must be a multiple of twice the synth block size. The default block size is 64.

Note that disk-in reads the number of outputs to create from what looks like an input, but it is not an input, and cannot be set using a control.

```
(let ((f "/home/rohan/audio/metal.wav")
      (n 1))
  (with-sc3
   (lambda (fd)
     (async fd (b-alloc 0 8192 n))
     (async fd (b-read 0 f 0 -1 0 1))
     (play fd (out 0 (disk-in n ar 0))))))


(with-sc3 reset)
```

```
(with-sc3
 (lambda (fd)
   (async fd (b-close 0))
   (async fd (b-free 0))))
```

4.46 (disk-out bufnum channels)

Note: There are constraints on the size of the buffer, it must be greater than or equal to twice the size of the audio bus. There must be the same number of channels at the buffer and the disk-out ugen.

```
(let ((bus-size 1024)
      (bufferexpt 15))
  (= 0 (fxand (expt 2 bufferexpt)
              (- (fxarithmetic-shift bus-size 1) 1))))
```

```
(let ((g (letc ((bufnum 0))
           (let ((z (clip2
                     (rlpf
                      (lf-pulse ar
                                (mul-add (sin-osc kr 0.2 0) 10 21)
                                (mce2 0 0.1)
                                0.1)
                      100
                      0.1)
                     0.4)))
             (mrg2 (disk-out bufnum z)
                   (out 0 z))))))
  (with-sc3
   (lambda (fd)
     (send-synth fd "disk-out-help" g)
     (async fd (b-alloc 10 32768 2))
     (async fd (b-write 10
                        "/tmp/test.aiff"
                        "aiff"
                        "float"
                        32768
                        0
                        1))
     (send fd (s-new1 "disk-out-help" 1001 1 1 "bufnum" 10)))))
```

```
(with-sc3
 (lambda (fd)
```

```
    (send fd (n-free1 1001))
    (async fd (b-close 10))
    (async fd (b-free 10))))

(system "sndfile-info /tmp/test.aiff")

(system "jack.play /tmp/test.aiff")
```

4.47 num-input-buses

The number of audio buses allocated to input. input buses follow output buses which begin at zero.

```
(let ((bus (add num-output-buses num-input-buses)))
  (audition (mrg2 (out 0 (in 1 ar bus))
                  (out bus (mul (sin-osc ar 440 0) 0.1)))))
```

4.48 num-control-buses

4.49 num-buffers

4.50 num-output-buses

The number of audio buses allocated to output.

```
(let ((bus num-output-buses))
  (audition (out 0 (mul (pulse ar 90 0.3)
                        (amplitude kr (lag (in 1 ar bus) 0.1) 0.01 0.01)))))
```

4.51 (buf-frames rate bufnum)

Current duration of buffer.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 0 "/home/rohan/audio/metal.wav" 0 0))))

(let ((p (phasor ar 0 (buf-rate-scale kr 0) 0 (buf-
frames kr 0) 0)))
  (audition (out 0 (buf-rd 1 ar 0 p 0 2))))
```

```
(let ((p (k2a (mouse-x kr 0 (buf-frames kr 0) 0 0.1))))
  (audition (out 0 (buf-rd 1 ar 0 p 0 2))))
```

4.52 (buf-rate-scale rate bufnum)

Buffer rate scaling in respect to server samplerate. Returns a ratio by which the playback of a soundfile is to be scaled.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 0 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((r (mul (rand 0.5 2) (buf-rate-scale kr 0)))
       (p (phasor ar 0 r 0 (buf-frames kr 0) 0)))
  (audition (out 0 (buf-rd 1 ar 0 p 0 2))))
```

4.53 sample-dur

Duration of one sample. Equivalent to 1 / sample-rate.

Compare a sine tone derived from sample rate with a 440Hz tone.

```
(let ((freq (mce2 (mul (recip sample-dur) 0.01) 440)))
  (audition (out 0 (mul (sin-osc ar freq 0) 0.1))))
```

4.54 sample-rate

Server sample rate.

Compare a sine tone derived from sample rate with a 440Hz tone.

```
(let ((freq (mce2 (mul sample-rate 0.01) 440)))
  (audition (out 0 (mul (sin-osc ar freq 0) 0.1))))
```

4.55 radians-per-sample

4.56 (buf-dur rate bufnum)

Current duration of buffer.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 0 "/home/rohan/audio/metal.wav" 0 0))))

(let* ((t (impulse ar (recip (buf-dur kr 0)) 0))
       (p (sweep t (buf-sample-rate kr 0))))
  (audition (out 0 (buf-rd 1 ar 0 p 0 2))))
```

4.57 (buf-channels rate bufnum)

Current number of channels of buffer. Using at .ir is not the safest choice. Since a buffer can be reallocated at any time, using ir will not track the changes.

4.58 (buf-sample-rate rate bufnum)

Buffer sample rate.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 0 "/home/rohan/audio/metal.wav" 0 0))))
```

Compare a sine tone derived from sample rate of a buffer with a 440Hz tone.

```
(let ((freq (mce2 (mul (buf-sample-rate ir 0) 0.01) 440)))
  (audition (out 0 (mul (sin-osc ar freq 0) 0.1))))
```

4.59 num-audio-buses

4.60 num-running-synths

Number of currently running synths.

```
(audition
 (out 0 (mul (sin-osc ar (mul-add num-running-synths 200 400) 0)
             0.1)))
```

4.61 subsample-offset

Offset from synth start within one sample.

When a synth is created from a time stamped osc-bundle, it starts calculation at the next possible block (normally 64 samples). Using an offset-out ugen, one can delay the audio so that it matches sample accurately. For some synthesis methods, one needs subsample accuracy. subsample-offset provides the information where, within the current sample, the synth was scheduled. It can be used to offset envelopes or resample the audio output.

See also offset-out.

Demonstrate cubic subsample interpolation. An impulse train that can be moved between samples.

```
(with-sc3
 (lambda (fd)
   (send-synth
    fd
    "s"
    (letc ((out 0)
           (add-offset 0))
      (let* ((i (mul (impulse ar 2000 0) 0.3))
             (d sample-dur)
             (x 4)
             (o (add (sub 1 subsample-offset)
                     (mouse-x kr 0 add-offset 0 0.1)))
             (r (delay-c i (mul d (add 1 x)) (mul d (add o x)))))
        (offset-out out r))))))
```

Create two pulse trains one sample apart, move one relative to the other. When cursor is at the left, the impulses are adjacent, on the right, they are exactly 1 sample apart. View this with an oscilloscope.

```
(with-sc3
 (lambda (fd)
   (let ((t (utc))
         (dt (/ 1 (server-sample-rate-actual fd))))
     (send fd (bundle (+ t 0.2)
                      (list (s-new1 "s" -1 1 1 "addOffset" 3))))
     (send fd (bundle (+ t 0.2 dt)
                      (list (s-new1 "s" -1 1 1 "addOffset" 0)))))))
```

4.62 (zero-crossing in)

Zero crossing frequency follower.

outputs a frequency based upon the distance between interceptions of the X axis. The X

intercepts are determined via linear interpolation so this gives better than just integer wavelength resolution. This is a very crude pitch follower, but can be useful in some situations.

in - input signal.

```
(let* ((a (mul (sin-osc ar (mul-add (sin-osc kr 1 0) 600 700) 0) 0.1))
       (b (mul (impulse ar (zero-crossing a) 0) 0.25)))
  (audition (out 0 (mce2 a b))))
```

4.63 (slope in)

slope of signal. Measures the rate of change per second of a signal. Formula implemented is:

out[i] = (in[i] - in[i-1]) * sampling_rate

in - input signal to measure.

a = quadratic noise, b = first derivative line segments, c = second derivative constant segments

```
(let* ((r 2)
       (a (lf-noise2 kr r))
       (scale (recip r))
       (b (mul (slope a) scale))
       (c (mul (slope b) (squared scale)))
       (o (sin-osc ar (mul-add (mce3 a b c) 220 220) 0)))
  (audition (out 0 (mix (mul o 1/3)))))
```

4.64 (running-sum in numsamp)

A running sum over a user specified number of samples, useful for running RMS power windowing.

in - input signal numsamp - How many samples to take the running sum over (initialisation rate)

```
(let ((n 40))
  (audition
   (out 0 (foldl1 mul (list (sin-osc ar 440 0)
                            (running-sum (sound-in (mce2 0 1)) n)
                            (recip n))))))
```

4.65 (pitch in initFreq minFreq maxFreq execFreq maxBinsPerOctave

median ampThreshold peakThreshold downSample)

Autocorrelation pitch follower

This is a better pitch follower than zero-crossing, but more costly of CPU. For most purposes the default settings can be used and only in needs to be supplied. pitch returns two values (via an Array of outputProxys, see the outputProxy help file), a freq which is the pitch estimate and hasFreq, which tells whether a pitch was found. Some vowels are still problematic, for instance a wide open mouth sound somewhere between a low pitched short 'a' sound as in 'sat', and long 'i' sound as in 'fire', contains enough overtone energy to confuse the algorithm.

sclang default argument values are: in = 0.0, initFreq = 440.0, minFreq = 60.0, maxFreq = 4000.0, execFreq = 100.0, maxBinsPerOctave = 16, median = 1, ampThreshold = 0.01, peakThreshold = 0.5, downSample = 1.

```
(define (pitch* in median ampThreshold)
  (pitch in 444.0 60.0 4000.0 100.0 16 median ampThreshold 0.5 1))

(let* ((in (mul (sin-osc ar (mouse-x kr 220 660 0 0.1) 0)
                (mouse-y kr 0.05 0.25 0 0.1)))
       (amp (amplitude kr in 0.05 0.05))
       (freq+ (pitch* in 7 0.02))
       (f (fdiv (car (mce-channels freq+)) 2))
       (o (mul (sin-osc ar f 0) amp)))
  (audition (out 0 (mce2 in o))))

(let* ((in (sound-in 0))
       (amp (amplitude kr in 0.05 0.05))
       (freq+ (pitch* in 7 0.02))
       (f (car (mce-channels freq+)))
       (o (mul (sin-osc ar f 0) amp)))
  (audition (out 0 (mce2 in o))))
```

4.66 (compander input control thresh slopeBelow slopeAbove clampTime relaxTime)

Compressor, expander, limiter, gate, ducker. General purpose dynamics processor.

input: The signal to be compressed / expanded / gated.

control: The signal whose amplitude determines the gain applied to the input signal. Often the same as in (for standard gating or compression) but should be different for ducking.

thresh: Control signal amplitude threshold, which determines the break point between slopeBelow and slopeAbove. Usually 0..1. The control signal amplitude is calculated using RMS.

slopeBelow: slope of the amplitude curve below the threshold. If this slope > 1.0, the amplitude will drop off more quickly the softer the control signal gets when the control signal is close to 0 amplitude, the output should be exactly zero – hence, noise gating. Values < 1.0 are possible, but it means that a very low-level control signal will cause the input signal to be amplified, which would raise the noise floor.

slopeAbove: Same thing, but above the threshold. Values < 1.0 achieve compression (louder signals are attenuated) > 1.0, you get expansion (louder signals are made even louder). For 3:1 compression, you would use a value of 1/3 here.

clampTime: The amount of time it takes for the amplitude adjustment to kick in fully. This is usually pretty small, not much more than 10 milliseconds (the default value).

relaxTime: The amount of time for the amplitude adjustment to be released. Usually a bit longer than clampTime if both times are too short, you can get some (possibly unwanted) artifacts.

Example signal to process.

```
(define z
  (mul (decay2 (mul (impulse ar 8 0) (mul (lf-
saw kr 0.3 0) 0.3)) 0.001 0.3)
       (mix (pulse ar (mce2 80 81) 0.3))))

(audition (out 0 z))
```

Noise gate

```
(let ((x (mouse-x kr 0.01 1 0 0.1)))
  (audition (out 0 (mce2 z (compander z z x 10 1 0.01 0.01)))))
```

Compressor

```
(let ((x (mouse-x kr 0.01 1 0 0.1)))
  (audition (out 0 (mce2 z (compander z z x 1 0.5 0.01 0.01)))))
```

limiter

```
(let ((x (mouse-x kr 0.01 1 0 0.1)))
  (audition (out 0 (mce2 z (compander z z x 1 0.1 0.01 0.01)))))
```

Sustainer

```
(let ((x (mouse-x kr 0.01 1 0 0.1)))
  (audition (out 0 (mce2 z (compander z z x 0.1 1.0 0.01 0.01)))))
```

4.67 (amplitude rate in attackTime releaseTime)

Amplitude follower. Tracks the peak amplitude of a signal.

```
(audition
 (out 0 (mul (pulse ar 90 0.3)
             (amplitude kr (in 1 ar num-output-
buses) 0.01 0.01))))


(let* ((a (amplitude kr (in 1 ar num-output-buses) 0.01 0.01))
       (f (mul-add a 1200 400)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.3))))
```

4.68 (pulse-divider trig div start)

outputs one impulse each time it receives a certain number of triggers at its input. A trigger happens when the signal changes from non-positive to positive.

```
(let* ((p (impulse ar 8 0))
       (d (pulse-divider p (mce2 4 7) 0))
       (a (mul (sin-osc ar 1200 0) (decay2 p 0.005 0.1)))
       (b (mul (sin-osc ar 600 0)  (decay2 d 0.005 0.5))))
  (audition (out 0 (mul (add a b) 0.4))))
```

4.69 (gate in trig)

The signal at ‘in’ is passed while ‘trig’ is greater than zero.

```
(let ((s (mul (f-sin-osc ar 500 0) 1/4))
      (t (lf-pulse ar 1 0 1/10)))
  (audition (out 0 (gate s t))))
```

4.70 (poll trig in trigid label)

Print/query the current output value of a UGen.

trig - a non-positive to positive transition telling poll to return a value

in - the signal you want to poll

trigid - if greater then 0, a ’/tr’ message is sent back to the client (similar to send-trig)

label - a string or symbol to be printed with the polled value

poll returns its in signal (and is therefore transparent). WARNING: Printing values from the Server in intensive for the CPU. poll should be used for debugging purposes.

```
(define (string->ugen s)
  (make-mce
   (cons (string-length s)
         (map char->integer (string->list s)))))


(let ((t (impulse kr 2 0))
      (i (line kr 0 1 5 remove-synth)))
  (audition (poll t i 0 (string->ugen "Test"))))


(with-sc3
 (lambda (fd)
   (letrec ((print (lambda (e) (display e) (newline)))
            (showing (lambda (f) (lambda () (let ((v (f))) (print v) v))))
            (repeat (lambda (f) (if (f) (repeat f) #f))))
     (async fd (/notify 1))
     (repeat (showing (lambda () (wait fd "/tr"))))
     (async fd (/notify 0)))))
```

multichannel Expansion (Broken...)

```
(define (poll* trig in trigId label)
  (poll trig in trigId (string->ugen label)))


(poll* (impulse kr (mce2 10 5) 0)
       (line kr 0 (mce2 1 5) (mce2 1 2) do-nothing)
       0
       "Test")


(with-sc3 server-status)
```

4.71 (most-change a b)

output the input that changed most.

```
(let* ((x (mouse-x kr 200 300 0 0.1))
       (f (most-change (mul-add (lf-noise0 kr 1) 400 900) x)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.72 (trig in dur)

When 'in' is trigerred output the trigger value for 'dur' seconds.

```
(let ((s (f-sin-osc ar 800 0))
      (g (trig (dust ar 1) 0.2)))
  (audition (out 0 (mul3 s g 0.5))))


(audition (out 0 (trig (dust ar 4) 0.1)))
```

4.73 (pulse-count trig reset)

This outputs the number of pulses received at 'trig' and outputs that value until 'reset' is triggered.

```
(let ((f (mul (pulse-count (impulse ar 10 0) (impulse ar 0.4 0)) 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.05))))
```

4.74 (stepper trig reset min max step resetval)

stepper pulse counter. Each trigger increments a counter which is output as a signal. The counter wraps between min and max.

```
(with-sc3
 (lambda (fd)
   (let ((a (list 97.999 195.998 523.251 466.164 195.998
                  233.082 87.307 391.995 87.307 261.626
                  195.998 77.782 233.082 195.998 97.999
                  155.563)))
     (async fd (b-alloc 10 128 1))
     (send fd (b-setn1 10 0 a)))))
```

Function composition...

```
(define (seq s l)
  (if (null? l)
      s
      (seq ((car l) s) (cdr l))))


(let* ((rate (mouse-x kr 1 5 1 0.1))
       (clock (impulse kr rate 0))
```

```
        (env (decay2 clock 0.002 2.5))
        (index (stepper clock 0 0 15 1 0))
        (freq (buf-rd 1 kr 10 index 1 1))
        (ffreq (add (if #t
                        (lag2 freq 0.1)
                        (mul (mouse-y kr 80 1600 1 0.1) (add (Mul env 4) 2)))
                    (mce2 0 0.3)))
        (lfo (sin-osc kr 0.2 (mce4 0 (/ pi 2) 0.0024 0.0025)))
        (rvb (lambda (s) (allpass-n s
                                    0.05
                                    (clone 2 (rand 0 0.05))
                                    (rand 1.5 2.0))))
        (proc (list
               (lambda (s) (mul (rlpf s ffreq 0.3) env))
               (lambda (s) (mul (rlpf s ffreq 0.3) env))
               (lambda (s) (mul s 0.2))

               (lambda (s) (mul-add (comb-l s 1 (fdiv 0.66 rate) 2) 0.8 s))

               (lambda (s) (add s (mul (seq s (replicate 5 rvb)) 0.3)))
               (lambda (s) (leak-dc s 0.1))

               (lambda (s) (add (delay-l s 0.1 lfo) s))

               (lambda (s) (one-pole s 0.9))))
        (init (mix (lf-pulse ar (mul freq (mce3 1 3/2 2)) 0 0.3))))
  (audition (out 0 (seq init proc))))
```

Pattern randomizer....

```
(with-sc3
 (lambda (fd)
   (let ((p (map (lambda (e)
                   (midi-cps (+ 36 (s:degree-to-
key e (list 0 3 5 7 10) 12))))
                 (map floor (replicate-m 16 (random 0 15))))))
     (send fd (b-setn1 10 0 p)))))
```

A shorter variant, using some simple syntax...

```
(define-syntax seq*
  (syntax-rules ()
    ((_ i s f ...)
     (seq i (list (lambda (s) f) ...)))))
```

```
(let* ((rate (mouse-x kr 1 5 1 0.1))
       (clock (impulse kr rate 0))
       (env (decay2 clock 0.002 2.5))
       (index (stepper clock 0 0 15 1 0))
       (freq (buf-rd 1 kr 10 index 1 1))
       (ffreq (add (lag2 freq 0.1) (mce2 0 0.3)))
       (lfo (sin-osc kr 0.2 (mce4 0 (/ pi 2) 0.0024 0.0025)))
       (rvb (lambda (s) (allpass-n s
                                   0.05
                                   (clone 2 (rand 0 0.05))
                                   (rand 1.5 2.0))))
       (init (mix (lf-pulse ar (mul freq (mce3 1 3/2 2)) 0 0.3)))
       (proc (seq* init
                   s
                   (mul (rlpf s ffreq 0.3) env)
                   (mul (rlpf s ffreq 0.3) env)
                   (mul s 0.2)
                   (mul-add (comb-l s 1 (fdiv 0.66 rate) 2) 0.8 s)
                   (add s (mul (seq s (replicate 5 rvb)) 0.3))
                   (leak-dc s 0.1)
                   (add (delay-l s 0.1 lfo) s)
                   (one-pole s 0.9))))
  (audition (out 0 proc)))
```

4.75 (last-value in diff)

output the last value before the input changed more than a threshhold.

```
(let ((f (last-value (mouse-x kr 100 400 0 0.1) 40)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))


(let* ((x (mouse-x kr 0.1 4 0 0.1))
       (f (mul-add (u:abs (sub x (last-value x 0.5))) 400 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.76 (peak-follower in decay)

Track peak signal amplitude. outputs the peak amplitude of the signal received at the input. If level is below maximum, the level decreases by the factor given in decay.

in - input signal. decay - decay factor.

internally, the absolute value of the signal is used, to prevent underreporting the peak value if there is a negative DC offset. To obtain the minimum and maximum values of the signal as is, use the running-min and running-max UGens.

No decay

```
(let* ((s (mul (dust ar 20) (line kr 0 1 4 do-nothing)))
       (f (mul-add (peak-follower s 1.0) 1500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

A little decay

```
(let* ((s (mul (dust ar 20) (line kr 0 1 4 do-nothing)))
       (f (mul-add (peak-follower s 0.999) 1500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

Mouse x controls decay

```
(let* ((x (mouse-x kr 0.99 1.0001 1 0.1))
       (s (mul (dust ar 20) (line kr 0 1 4 do-nothing)))
       (f (mul-add (peak-follower s (u:min x 1.0)) 1500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

Follow a sine lfo

```
(let* ((x (mouse-x kr 0.99 1.0001 1 0.1))
       (s (sin-osc kr 0.2 0))
       (f (mul-add (peak-follower s (u:min x 1.0)) 200 500)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.77 (running-max in trig)

Track maximum level. outputs the maximum value received at the input. When triggered, the maximum output value is reset to the current value.

in - input signal trig - reset the output value to the current input value

```
(let* ((t (impulse ar 0.4 0))
       (f (mul-add (running-max (dust ar 20) t) 500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

```
(let* ((t (impulse kr (mouse-x kr 0.01 2 1 0.1) 0))
       (f (mul-add (running-max (sin-osc kr 2 0) t) 500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.78 (trig1 in dur)

When 'in' is trigered output a unit signal for 'dur' seconds.

```
(let ((a (trig1 (dust ar 1) 0.2)))
  (audition (out 0 (mul3 (f-sin-osc ar 800 0) a 0.2))))
```

4.79 (phasor trig rate start end resetpos)

triggered linear ramp between two levels. Starts a linear ramp when trig input crosses from non-positive to positive.

trig - sets phase to resetPos (default: 0, equivalent to start) rate - rate value in 1 / frameDur (at 44.1 kHz sample rate: rate 1 is eqivalent to 44100/sec) start, end - start and end points of ramp resetPos - determines where to jump to on recieving a trigger. the value at that position can be calculated as follows: (end - start) * resetPos

phasor controls sine frequency: end frequency matches a second sine wave.

```
(let* ((r (mouse-x kr 0.2 2 1 0.1))
       (t (impulse ar r 0))
       (x (phasor ar t (fdiv r sample-rate) 0 1 0))
       (f (mce2 (lin-lin x 0 1 600 1000) 1000)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.80 (schmidt in lo hi)

schmidt trigger. When in crosses to greater than hi, output 1.0, then when signal crosses lower than lo output 0.0. output is initially zero.

in - signal to be tested lo - low threshold hi - high threshold

```
(let* ((in (lf-noise1 kr 3))
       (octave (add (schmidt in -0.15 0.15) 1))
       (f (add (mul in 200) (mul 500 octave))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.81 (peak trig reset)

outputs the maximum value read at the 'trig' input until 'reset' is triggered.

```
(let* ((p (peak (dust ar 20) (impulse ar 0.4 0)))
```

```
        (f (mul-add p 500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.82 (toggle-ff trig)

Toggle flip flop. Toggles between zero and one upon receiving a trigger.

trig - trigger input

```
(let* ((t (dust ar (x-line kr 1 1000 60 do-nothing)))
       (s (sin-osc ar (mul-add (toggle-ff t) 400 800) 0)))
  (audition (out 0 (mul s 0.1))))
```

4.83 (sweep trig rate)

triggered linear ramp. Starts a linear raise by rate/sec from zero when trig input crosses from non-positive to positive.

Using sweep to modulate sine frequency

```
(let* ((t (impulse kr (mouse-x kr 0.5 20 1 0.1) 0))
       (f (add (sweep t 700) 500)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

Using sweep to index into a buffer

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 0 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((t (impulse ar (mouse-x kr 0.5 20 1 0.1) 0))
       (i (sweep t (buf-sample-rate ir 0))))
  (audition (out 0 (buf-rd 1 ar 0 i 0 2))))
```

Backwards, variable offset

```
(let* ((t (impulse ar (mouse-x kr 0.5 10 1 0.1) 0))
       (r (buf-sample-rate ir 0))
       (i (add (sweep t (neg r)) (mul (buf-frames ir 0) (lf-
noise0 kr 15)))))
  (audition (out 0 (buf-rd 1 ar 0 i 0 2))))
```

Raising rate

```
(let* ((t (impulse ar (mouse-x kr 0.5 10 1 0.1) 0))
       (r (add (sweep t 2) 0.5))
       (i (sweep t (mul (buf-sample-rate ir 0) r))))
  (audition (out 0 (buf-rd 1 ar 0 i 0 2))))
```

4.84 (send-trig in id value)

On receiving a trigger (0 to non-zero transition), send a trigger message from the server back to all registered clients. Clients register by sending a /notify message to the server.

input - the trigger

id - an integer that will be passed with the trigger message. This is useful if you have more than one send-trig in a SynthDef

value - a UGen or float that will be polled at the time of trigger, and its value passed with the trigger message

```
(let ((s (lf-noise0 kr 10)))
  (audition (mrg2 (send-trig s 0 s)
                  (out 0 (mul (sin-osc ar (mul-
add s 200 500) 0) 0.1)))))


(with-sc3
 (lambda (fd)
   (async fd (notify 1))
   (sleep 2.0)
   (let ((r (wait fd "/tr")))
     (async fd (notify 0))
     r)))
```

4.85 (in-range in lo hi)

Tests if a signal is within a given range.

If in is >= lo and <= hi output 1.0, otherwise output 0.0. output is initially zero.

in - signal to be tested lo - low threshold hi - high threshold

```
(let ((a (in-range (mul (sin-osc kr 1 0) 0.2) -0.15 0.15)))
  (audition (out 0 (mul a (mul (brown-noise ar) 0.1)))))
```

4.86 (timer trig)

Returns time since last triggered

Using timer to modulate sine frequency: the slower the trigger is the higher the frequency

```
(let* ((t (impulse kr (mouse-x kr 0.5 20 1 0.1) 0))
       (s (sin-osc ar (mul-add (timer t) 500 500) 0)))
  (audition (out 0 (mul s 0.2))))
```

4.87 (t-delay trigger delayTime)

Delays a trigger by a given time. Any triggers which arrive in the time between an input trigger and its delayed output, are ignored.

trigger - input trigger signal. delayTime - delay time in seconds.

```
(let* ((s (mul (sin-osc ar 440 0) 0.1))
       (z (impulse ar 2 0))
       (l (mul z 0.1))
       (r (mul (toggle-ff (t-delay z 0.5)) s)))
  (audition (out 0 (mce2 l r))))
```

4.88 (running-min in trig)

Track minimum level. outputs the minimum value received at the input. When triggered, the minimum output value is reset to the current value.

in - input signal trig - reset the output value to the current input value

```
(let* ((t (impulse ar 2.0 0))
       (f (mul-add (running-min (sub 1 (dust ar 20)) t) 500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))

(let* ((t (impulse kr (mouse-x kr 0.5 4 1 0.1) 0))
       (f (mul-add (running-min (sub 2 (sin-
osc kr 2 0)) t) 500 200)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.89 (set-reset-ff trig reset)

Set-reset flip flop. output is set to 1.0 upon receiving a trigger in the set input, and to 0.0 upon receiving a trigger in the reset input. Once the flip flop is set to zero or one further

triggers in the same input are have no effect. One use of this is to have some precipitating event cause something to happen until you reset it.

trig - trigger sets output to one reset - trigger resets output to zero

```
(let ((n (brown-noise ar))
      (g (set-reset-ff (dust ar 5) (dust ar 5))))
  (audition (out 0 (mul3 n g 0.2))))
```

4.90 (saw rate freq)

Band limited sawtooth wave generator.

```
(let ((f (x-line kr 40 4000 6 do-nothing)))
  (audition (out 0 (mul (saw ar f) 0.2))))
```

Two band limited sawtooth waves thru a resonant low pass filter

```
(let ((f (x-line kr 8000 400 5 do-nothing)))
  (audition (out 0 (rlpf (mul (saw ar (mce2 100 250)) 0.1) f 0.05))))
```

4.91 (pm-osc rate carfreq modfreq index modphase)

Phase modulation oscillator pair.

carfreq - carrier frequency in cycles per second. modfreq - modulator frequency in cycles per second. index - modulation index in radians. modphase - a modulation input for the modulator's phase in radians

```
(let* ((f (line kr 600 900 5 remove-synth))
       (o (mul (pm-osc ar f 600 3 0) 0.1)))
  (audition (out 0 o)))


(let* ((mf (line kr 600 900 5 remove-synth))
       (o (mul (pm-osc ar 300 mf 3 0) 0.1)))
  (audition (out 0 o)))


(let* ((i (line kr 0 20 8 remove-synth))
       (o (mul (pm-osc ar 300 550 i 0) 0.1)))
  (audition (out 0 o)))
```

4.92 (lf-tri rate freq iphase)

A non-band-limited triangular waveform oscillator. output ranges from -1 to +1.

```
(audition (out 0 (mul (lf-tri ar 500 1) 0.1)))
```

Used as both oscillator and LFO.

```
(let ((f (mul-add (lf-tri kr 4 0) 400 400)))
  (audition (out 0 (mul (lf-tri ar f 0) 0.1))))
```

4.93 (t-grains numChannels trigger bufnum rate centerPos dur pan amp interp)

Buffer granulator. triggers generate grains from a buffer. Each grain has a Hanning envelope (sin^2(x) for x from 0 to pi) and is panned between two channels of multiple outputs.

numChannels - number of output channels.

trigger - at each trigger, the following arguments are sampled and used as the arguments of a new grain. A trigger occurs when a signal changes from <= 0 to > 0. If the trigger is audio rate then the grains will start with sample accuracy.

bufnum - the index of the buffer to use. It must be a one channel (mono) buffer.

rate - 1.0 is normal, 2.0 is one octave up, 0.5 is one octave down and -1.0 is backwards normal rate ... etc. Unlike play-buf, the rate is multiplied by BufRate, so you needn't do that yourself.

centerPos - the position in the buffer in seconds at which the grain envelope will reach maximum amplitude.

dur - duration of the grain in seconds.

pan - a value from -1 to 1. Determines where to pan the output in the same manner as pan-az.

amp - amplitude of the grain.

interp - 1, 2, or 4. Determines whether the grain uses (1) no interpolation, (2) linear interpolation, or (4) cubic interpolation.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 10 "/home/rohan/audio/metal.wav" 0 0))))
```

```
(audition
 (let* ((b 10)
        (trate (mouse-y kr 2 200 1 0.1))
        (dur (fdiv 4 trate))
        (t (impulse ar trate 0))
        (i (mouse-x kr 0 (buf-dur kr b) 0 0.1)))
  (out 0 (t-grains 2 t b 1 i dur 0 0.1 2))))

(audition
 (let* ((b 10)
        (trate (mouse-y kr 8 120 1 0.1))
        (dur (fdiv 12 trate))
        (clk (impulse ar trate 0))
        (x (mouse-x kr 0 (buf-dur kr b) 0 0.1))
        (pos (add x (t-rand 0 0.01 clk)))
        (pan (mul (white-noise kr) 0.6)))
   (out 0 (t-grains 2 clk b 1 pos dur pan 0.1 2))))

(audition
 (let* ((b 10)
        (trate (mouse-y kr 8 120 1 0.1))
        (dur (fdiv 4 trate))
        (clk (dust ar trate))
        (x (mouse-x kr 0 (buf-dur kr b) 0 0.1))
        (pos (add x (t-rand 0 0.01 clk)))
        (pan (mul (white-noise kr) 0.6)))
   (out 0 (t-grains 2 clk b 1 pos dur pan 0.1 2))))
```

The SC3 ** operator is the ShiftLeft binary UGen.

```
(audition
 (let* ((b 10)
        (trate (mouse-y kr 2 120 1 0.1))
        (dur (fdiv 1.2 trate))
        (clk (impulse ar trate 0))
        (pos (mouse-x kr 0 (buf-dur kr b) 0 0.1))
        (pan (mul (white-noise kr) 0.6))
        (rate (shift-left 1.2 (u:round (mul (white-
noise kr) 3) 1))))
   (out 0 (t-grains 2 clk b rate pos dur pan 0.1 2))))
```

4.94 (tw-index in normalize array)

triggered windex. When triggered, returns a random index value based on array as a list of probabilities. By default the list of probabilities should sum to 1.0, when the normalize flag

is set to 1, the values get normalized by the ugen (less efficient)

Assuming normalized values

```
(audition
 (let* ((prob (mce3 1/5 2/5 2/5))
        (freq (mce3 400 500 600))
        (f (select (tw-index (impulse kr 6 0) 0.0 prob) freq)))
   (out 0 (mul (sin-osc ar f 0) 0.2))))
```

Modulating probability values

```
(audition
 (let* ((t (impulse kr 6 0))
        (a (mce3 1/4 1/2 (mul-add (sin-osc kr 0.3 0) 0.5 0.5)))
        (f (select (tw-index t 1.0 a) (mce3 400 500 600))))
   (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.95 (osc-n rate bufnum freq phase)

Noninterpolating wavetable lookup oscillator with frequency and phase modulation inputs. It is usually better to use the interpolating oscillator.

The buffer size must be a power of 2. The buffer should NOT be filled using Wavetable format (b_gen commands should set wavetable flag to false.

4.96 (osc rate bufnum freq phase)

linear interpolating wavetable lookup oscillator with frequency and phase modulation inputs.

This oscillator requires a buffer to be filled with a wavetable format signal. This preprocesses the Signal into a form which can be used efficiently by the oscillator. The buffer size must be a power of 2.

This can be acheived by creating a Buffer object and sending it one of the "b_gen" messages (sine1, sine2, sine3) with the wavetable flag set to true.

Note about wavetables: oscN requires the b_gen sine1 wavetable flag to be OFF. osc requires the b_gen sine1 wavetable flag to be ON.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 512 1))
   (async fd (b-gen1 10 "sine1" (list (+ 1 2 4) 1 1/2 1/3 1/4 1/5)))))
```

```
(audition (out 0 (mul (osc ar 10 220 0) 0.1)))
```

Modulate freq

```
(let ((f (x-line kr 2000 200 1 remove-synth)))
  (audition (out 0 (mul (osc ar 10 f 0) 0.5))))
```

Modulate freq

```
(let* ((f1 (x-line kr 1 1000 9 remove-synth))
       (f2 (mul-add (osc ar 10 f1 0) 200 800)))
  (audition (out 0 (mul (osc ar 10 f2 0) 0.25))))
```

Modulate phase

```
(let* ((f (x-line kr 20 8000 10 remove-synth))
       (p (mul (osc ar 10 f 0) (* 2 pi))))
  (audition (out 0 (mul (osc ar 10 800 p) 0.25))))
```

Change the buffer while its playing

```
(audition (out 0 (mul (osc ar 10 220 0) 0.1)))


(with-sc3
 (lambda (fd)
   (async fd (b-gen1 10 "sine1" (list (+ 1 2 4) 1 (random 0 1) 1/4)))))
```

4.97 (lf-saw rate freq iphase)

sawtooth oscillator. A non-band-limited sawtooth oscillator. output ranges from -1 to +1.

freq - frequency in Hertz iphase - phase in radians

```
(audition (out 0 (mul (lf-saw ar 500 1) 0.1)))
```

Used as both oscillator and LFO.

```
(let ((f (mul-add (lf-saw kr 4 0) 400 400)))
  (audition (out 0 (mul (lf-saw ar f 0) 0.1))))
```

4.98 (tw-choose trig array weights normalize)

The output is selected randomly on recieving a trigger from an array of inputs. The weights of this choice are determined from the weights array. If normalize is set to 1 the weights are continuously normalized, which means an extra calculation overhead. When using fixed values the normalizeSum method can be used to normalize the values. TWChoose is a composite of TWindex and select

```
(let ((a (mce3 (sin-osc ar 220 0)
               (saw ar 440)
               (pulse ar 110 0.1)))
      (t (dust ar (mouse-x kr 1 1000 1 0.1)))
      (w (mce3 0.6 0.15 0.05)))
  (audition (out 0 (mul (tw-choose t a w 1) 0.1))))
```

Note: all the ugens are continously running. This may not be the most efficient way if each input is cpu-expensive.

4.99 gendy1

sclang defaults

```
(audition (out 0 (pan2 (gendy1 ar 1 1 1 1 440 660 0.5 0.5 12 12) 0 0.15)))
```

Wandering bass

```
(audition (out 0 (pan2 (gendy1 ar 1 1 1.0 1.0 30 100 0.3 0.05 5 5) 0 0.15)))
```

Play me

```
(let* ((x (mouse-x* kr 100 1000 1 0.1))
       (g (gendy1 ar 2 3 1 1 20 x 0.5 0.0 40 40)))
  (audition (out 0 (pan2 (mul (rlpf g 500 0.3) 0.2) 0.0 0.25))))
```

Scream!

```
(let ((x (mouse-x kr 220 440 1 0.1))
      (y (mouse-y kr 0.0 1.0 0 0.1)))
 (audition (out 0 (pan2 (gendy1 ar 2 3 1 1 x (mul 8 x) y y 7 7) 0.0 0.3))))
```

1 CP = random noise

```
(audition (out 0 (pan2 (gendy1 ar 1 1 1 1 440 660 0.5 0.5 1 1) 0 0.15)))
```

2 CPs = an oscillator

```
(audition (out 0 (pan2 (gendy1 ar 1 1 1 1 440 660 0.5 0.5 2 2) 0 0.15)))
```

Used as an LFO

```
(let* ((ad (mul-add (sin-osc kr 0.1 0) 0.49 0.51))
       (dd (mul-add (sin-osc kr 0.13 0) 0.49 0.51))
       (as (mul-add (sin-osc kr 0.17 0) 0.49 0.51))
       (ds (mul-add (sin-osc kr 0.19 0) 0.49 0.51))
       (g  (gendy1 kr 2 4 ad dd 3.4 3.5 as ds 10 10)))
  (audition (out 0 (pan2 (sin-osc ar (mul-
add g 50 350) 0) 0.0 0.3))))
```

Wasp

```
(let ((ad (mul-add (sin-osc kr 0.1 0) 0.1 0.9)))
  (audition (out 0 (pan2 (gendy1 ar 0 0 ad 1.0 50 1000 1 0.005 12 12) 0.0 0.2))))
```

Modulate distributions. Change of pitch as distributions change the duration structure and spectrum

```
(let* ((x (mouse-x* kr 0 7 0 0.1))
       (y (mouse-y* kr 0 7 0 0.1))
       (g (gendy1 ar x y 1 1 440 660 0.5 0.5 12 12)))
  (audition (out 0 (pan2 g 0 0.2))))
```

Modulate number of CPs.

```
(let* ((x (mouse-x* kr 1 13 0 0.1))
       (g (gendy1 ar 1 1 1 1 440 660 0.5 0.5 12 x)))
  (audition (out 0 (pan2 g 0 0.2))))
```

Self modulation.

```
(let* ((x  (mouse-x* kr 1 13 0 0.1))
       (y  (mouse-y* kr 0.1 10 0 0.1))
       (g0 (gendy1 kr 5 4 0.3 0.7 0.1 y 1.0 1.0 5 5))
       (g1 (gendy1 ar 1 1 1 1 440 (mul-add g0 500 600) 0.5 0.5 12 x)))
  (audition (out 0 (pan2 g1 0.0 0.2))))
```

Use SINUS to track any oscillator and take CP positions from it use adparam and ddparam as the inputs to sample.

```
(let* ((p (lf-pulse kr 100 0 0.4))
       (s (mul (sin-osc kr 30 0) 0.5))
       (g (gendy1 ar 6 6 p s 440 660 0.5 0.5 12 12)))
  (audition (out 0 (pan2 g 0.0 0.2))))
```

Near the corners are interesting.

```
(let* ((x (mouse-x* kr 0 200 0 0.1))
       (y (mouse-y* kr 0 200 0 0.1))
       (p (lf-pulse kr x 0 0.4))
       (s (mul (sin-osc kr y 0) 0.5))
       (g (gendy1 ar 6 6 p s 440 660 0.5 0.5 12 12)))
  (audition (out 0 (pan2 g 0.0 0.2))))
```

Texture

```
(let*
    ((o (let* ((f (rand 130.0 160.3))
               (ad (mul-add (sin-osc kr 0.1 0) 0.49 0.51))
               (dd (mul-add (sin-osc kr 0.13 0) 0.49 0.51))
               (as (mul-add (sin-osc kr 0.17 0) 0.49 0.51))
               (ds (mul-add (sin-osc kr 0.19 0) 0.49 0.51))
               (g  (gendy1 ar (rand 0 6) (rand 0 6) ad dd f f as ds 12 12)))
          (pan2 (sin-osc ar (mul-add g 200 400) 0)
                (rand -1 1)
                0.1)))
     (x (mix-fill 10 (lambda (_) o))))
  (audition (out 0 x)))
```

Try durscale 10.0 and 0.0 too.

```
(let* ((x (mouse-x* kr 10 700 0 0.1))
       (y (mouse-y* kr 50 1000 0 0.1))
       (g (gendy1 ar 2 3 1 1 1 x 0.5 0.1 10 10)))
  (audition (out 0 (pan2 (comb-n (resonz g y 0.1) 0.1 0.1 5) 0.0 0.6))))
```

Overkill

```
(define (overkill i)
  (mix-fill
   i
   (lambda (_)
     (let* ((f (rand 50 560.3))
            (n (rand 2 20))
```

```
            (k (mul-add (sin-osc kr (exp-rand 0.02 0.2) 0)
                        (fdiv n 2)
                        (fdiv n 2)))
            (g (gendy1 ar
                       (rand 0 6) (rand 0 6) (rand 0 1) (rand 0 1) f f
                       (rand 0 1) (rand 0 1) n k)))
       (pan2 g (rand -1 1) (fdiv 0.5 (sqrt i)))))))

(audition (out 0 (overkill 10)))
```

Another traffic moment

```
(let ((x (mouse-x* kr 100 2000 0 0.1))
      (y (mouse-y* kr 0.01 1.0 0 0.1)))
  (audition (out 0 (resonz (overkill 10) x y))))
```

4.100 (pulse rate freq width)

Bandlimited pulse wave generator.

Modulate frequency

```
(audition
 (let ((f (x-line kr 40 4000 6 remove-synth)))
   (out 0 (mul (pulse ar f 0.1) 0.2))))
```

modulate pulse width

```
(audition
 (let ((w (line kr 0.01 0.99 8 remove-synth)))
   (out 0 (mul (pulse ar 200 w) 0.2))))
```

two band limited square waves thru a resonant low pass filter

```
(audition
 (out 0 (rlpf (mul (pulse ar (mce2 100 250) 0.5) 0.1)
              (x-line kr 8000 400 5 remove-synth)
              0.05)))
```

4.101 (shaper bufnum in)

Wave shaper. Performs waveshaping on the input signal by indexing into the table.

bufnum - the number of a buffer filled in wavetable format containing the transfer function.
in - the input signal.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 512 1))
   (async fd (b-gen1 10 "cheby" (list 0 1 0 1 1 0 1)))))


(audition
 (let* ((a (line kr 0 1 6 remove-synth))
        (s (mul (sin-osc ar 300 0) a)))
   (out 0 (mul (shaper 10 s) 0.5))))
```

4.102 SC2: Note extra iphase argument.

```
(import (rsc3))


(audition (out 0 (mul (f-sin-osc ar (mce2 440 550) 0) 0.05)))


(let ((f (x-line kr 200 4000 1 remove-synth)))
  (audition (out 0 (mul (f-sin-osc ar f 0) 0.25))))
```

Loses amplitude towards the end

```
(let ((f (mul-add (f-sin-osc ar (x-line kr 4 401 8 remove-
synth) 0)
                  200
                  800)))
  (audition (out 0 (mul (f-sin-osc ar f 0) 0.25))))
```

4.103 (v-osc rate bufpos freq phase)

Variable wavetable oscillator. A wavetable lookup oscillator which can be swept smoothly across wavetables. All the wavetables must be allocated to the same size. Fractional values of table will interpolate between two adjacent tables.

This oscillator requires a buffer to be filled with a wavetable format signal. This preprocesses the Signal into a form which can be used efficiently by the oscillator. The buffer size must be a power of 2.

This can be acheived by creating a Buffer object and sending it one of the "b_gen" messages (sine1, sine2, sine3) with the wavetable flag set to true.

This can also be acheived by creating a Signal object and sending it the 'asWavetable' message, saving it to disk, and having the server load it from there.

Note about wavetables: Vosc requires the b_gen sine1 wavetable flag to be ON.

Allocate and fill tables 0 to 7 [see also Vosc3]

```
(with-sc3
 (lambda (fd)
   (let* ((square
           (lambda (a) (* a a)))
          (nth
           (lambda (i)
             (async fd (b-alloc i 1024 1))
             (let* ((n (expt (+ i 1) 2))
                    (a (map1 (lambda (j) (square (/ (- n j) n)))
                             (enum-from-to 0 (- n 1)))))
               (async fd (b-gen1 i "sine1" (cons 7 a)))))))
     (for-each nth (enum-from-to 0 7)))))
```

oscillator at buffers 0 through 7.

```
(let ((b (mouse-x kr 0 7 0 0.1))
      (f (mce2 120 121)))
  (audition (out 0 (mul (v-osc ar b f 0) 0.3))))
```

Reallocate buffers while oscillator is running.

```
(with-sc3
 (lambda (fd)
   (for-each
    (lambda (i)
      (async fd (b-gen1  i "sine1" (cons 7 (replicate-
m 16 (random 0 1))))))
    (enum-from-to 0 7))))
```

4.104 (var-saw rate freq iphasewidth)

Variable duty saw

freq - frequency in Hertz iphase - initial phase offset in cycles (0..1) width - duty cycle from zero to one.

```
(let ((f (mul-add (lf-pulse kr (mce2 3 3.03) 0 0.3) 200 200))
      (w (lin-lin (lf-tri kr 1 0) -1 1 0 1)))
  (audition (out 0 (mul (var-saw ar f 0 w) 0.1))))
```

4.105 buf-wr

```
(let ((a (letc ((r 1))
           (let* ((r* (mul (buf-rate-scale kr 0) r))
                  (p (phasor ar 0 r* 0 (buf-frames kr 0) 0))
                  (f (mul-add (lf-noise1 kr 2) 300 400))
                  (i (mul (sin-osc ar f 0) 0.1)))
             (mrg2 (buf-wr 0 p 1 i)
                   (out 0 0.0)))))
      (b (letc ((r 1))
           (let* ((r* (mul (buf-rate-scale kr 0) r))
                  (p (phasor ar 0 r* 0 (buf-frames kr 0) 0)))
             (out 0 (buf-rd 1 ar 0 p 1 2))))))
  (with-sc3
   (lambda (fd)
     (async fd (b-alloc 0 (* 44100 2) 1))
     (send-synth fd "a" a)
     (send-synth fd "b" b)
     (send fd (s-new0 "a" 1001 1 0))
     (send fd (s-new0 "b" 1002 1 0)))))

(define (do-send m)
  (with-sc3 (lambda (fd) (send fd m))))

(do-send (n-set1 1002 "r" 5))

(do-send (n-set1 1001 "r" (random 0 2)))

(do-send (n-set1 1002 "r" 2))
```

4.106 impulse

```
(import (rsc3))

(audition (out 0 (mul (impulse ar 800 0) 0.1)))

(let ((f (x-line kr 800 10 5 remove-synth)))
  (audition (out 0 (mul (impulse ar f 0.0) 0.5))))

(let ((f (mouse-y* kr 4 8 0 0.1))
      (p (mouse-x* kr 0 1 0 0.1)))
  (audition (out 0 (mul (impulse ar f (mce2 0 p)) 0.2))))
```

4.107 blip

```
(import (rsc3))

(audition (out 0 (mul (blip ar 440 200) 0.15)))
```

Modulate frequency

```
(let ((f (x-line kr 20000 200 6 remove-synth)))
  (audition (out 0 (mul (blip ar f 100) 0.2))))
```

Modulate number of harmonics.

```
(let ((h (line kr 1 100 20 remove-synth)))
  (audition (out 0 (mul (blip ar 200 h) 0.2))))
```

4.108 (select which array)

The output is selected from an array of inputs.

```
(audition
 (let* ((a (mce3 (sin-osc ar 440 0) (saw ar 440) (pulse ar 440 0.1)))
        (cycle 3/2)
        (w (mul-add (lf-saw kr 1 0) cycle cycle)))
   (out 0 (mul (select w a) 0.2))))
```

Note: all the ugens are continously running. This may not be the most efficient way if each input is cpu-expensive.

Here used as a sequencer:

```
(audition
 (let* ((n 32)
        (a (make-mce (map (compose midi-cps u:floor)
                          (replicate-m n (rand 30 80)))))
        (cycle (/ n 2))
        (w (mul-add (lf-saw kr 1/2 0) cycle cycle)))
   (out 0 (mul (saw ar (select w a)) 0.2))))
```

4.109 formant

```
(import (rsc3))
```

Modulate fundamental frequency, formant frequency stays constant.

```
(audition
 (let ((f (x-line kr 400 1000 8 remove-synth)))
   (out 0 (mul (formant ar f 2000 800) 0.125))))
```

Modulate formant frequency, fundamental frequency stays constant.

```
(audition
 (let ((f (x-line kr 400 4000 8 remove-synth)))
   (out 0 (mul (formant ar (mce2 200 300) f 200) 0.125))))
```

Modulate width frequency, other frequencies stay constant.

```
(audition
 (let ((w (x-line kr 800 8000 8 remove-synth)))
   (out 0 (mul (formant ar 400 2000 w) 0.125))))
```

4.110 c-osc

```
(import (rsc3))


(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 512 1))
   (async fd (b-gen1 10 "sine1" (list (+ 1 2 4) 1 1/2 1/3 1/4 1/5 1/6 1/7 1/8 1/9 1/10)))))


(audition (out 0 (mul (c-osc ar 10 200 0.7) 0.25)))


(audition (out 0 (mul (c-osc ar 10 200 (mouse-
x* kr 0 4 0 0.1)) 0.25)))
```

Compare with:

```
(audition (out 0 (mul (osc ar 10 200 0.0) 0.25)))
```

4.111 (v-osc3 rate bufpos freq1 freq2 freq3)

Three variable wavetable oscillators.

A wavetable lookup oscillator which can be swept smoothly across wavetables. All the wavetables must be allocated to the same size. Fractional values of table will interpolate between two adjacent tables. This unit generator contains three oscillators at different frequencies, mixed together.

This oscillator requires a buffer to be filled with a wavetable format signal. This preprocesses the Signal into a form which can be used efficiently by the oscillator. The buffer size must be a power of 2.

This can be acheived by creating a Buffer object and sending it one of the "b_gen" messages (sine1, sine2, sine3) with the wavetable flag set to true.

Note about wavetables: Vosc3 requires the b_gen sine1 wavetable flag to be ON.

Allocate and fill tables 0 to 7 with a generated list of harmonic amplitudes.

```
(with-sc3
 (lambda (fd)
   (let* ((square
           (lambda (a) (* a a)))
          (nth
           (lambda (i)
             (async fd (b-alloc i 1024 1))
             (let* ((n (expt (+ i 1) 2))
                    (a (map (lambda (j)
                              (square (/ (- n j) n)))
                            (enum-from-to 0 (- n 1)))))
               (async fd (b-gen1 i "sine1" (cons 7 a)))))))
     (for-each nth (enum-from-to 0 7)))))
```

oscillator at buffers 0 through 7.

```
(let ((p (mouse-x kr 0 7 0 0.1))
      (f1 (mce2 240 241))
      (f2 (mce2 240.27 241.1))
      (f3 (mce2 240.43 239.71)))
  (audition (out 0 (mul (v-osc3 ar p f1 f2 f3) 0.2))))
```

Reallocate buffers while oscillator is running.

```
(with-sc3
 (lambda (fd)
   (for-each
    (lambda (i)
      (async fd (b-gen1  i "sine1" (cons 7 (replicate-
m 16 (random 0 1))))))
    (enum-from-to 0 7))))
```

4.112 (lf-cub rate freq iphase)

a sine like shape made of two cubic pieces. smoother than lf-par.

```
(let ((f (mul-add (lf-cub kr (mul-add (lf-
cub kr 0.2 0) 8 10) 0) 400 800)))
  (mul (lf-cub ar f 0) 0.1))

(mul (lf-cub ar (mul-add (lf-cub kr 0.2 0) 400 800) 0) 0.1)

(mul (lf-cub ar 800 0) 0.1)

(mul (lf-cub ar (x-line kr 100 8000 30 do-nothing) 0) 0.1)
```

compare:

```
(let ((f (mul-add (lf-par kr (mul-add (lf-
par kr 0.2 0) 8 10) 0) 400 800)))
  (mul (lf-par ar f 0) 0.1))

(mul (lf-par ar (mul-add (lf-par kr 0.2 0) 400 800) 0) 0.1)

(mul (lf-par ar 800 0) 0.1)

(mul (lf-par ar (x-line kr 100 8000 30 do-nothing) 0) 0.1)
```

compare:

```
(let ((f (mul-add (sin-osc kr (mul-add (sin-
osc kr 0.2 0) 8 10) 0) 400 800)))
  (mul (sin-osc ar f 0) 0.1))

(mul (sin-osc ar (mul-add (sin-osc kr 0.2 0) 400 800) 0) 0.1)

(mul (sin-osc ar 800 0) 0.1)

(mul (sin-osc ar (x-line kr 100 8000 30 do-nothing) 0) 0.1)
```

compare:

```
(let ((f (mul-add (lf-tri kr (mul-add (lf-
tri kr 0.2 0) 8 10) 0) 400 800)))
  (mul (lf-tri ar f 0) 0.1))

(mul (lf-tri ar (mul-add (lf-tri kr 0.2 0) 400 800) 0) 0.1)

(mul (lf-tri ar 800 0) 0.1)

(mul (lf-tri ar (x-line kr 100 8000 30 do-nothing) 0) 0.1)
```

4.113 (lf-pulse rate freq iphase width)

A non-band-limited pulse oscillator. outputs a high value of one and a low value of zero. Note that the iphase argument was not present in SC2.

freq - frequency in Hertz iphase - initial phase offset in cycles (0..1) width - pulse width duty cycle from zero to one.

```
(let ((f (mul-add (lf-pulse kr 3 0 0.3) 200 200)))
  (audition (out 0 (mul (lf-pulse ar f 0 0.2) 0.1))))
```

4.114 index

```
(import (sosc) (rsc3))
```

Allocate and set values at buffer 10.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 6 1))
   (send fd (b-setn1 10 0 (list 50 100 200 400 800 1600)))))
```

Index into the above buffer for frequency values.

```
(let ((f (mul (index 10 (mul (lf-saw kr 2 3) 4)) (mce2 1 9))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

Free buffer

```
(with-sc3
 (lambda (fd)
   (async fd (b-free 10))))
```

4.115 See lf-cub.

A sine-like shape made of two parabolas. Has audible odd harmonics.

4.116 (sync-saw rate syncFreq sawFreq)

A sawtooth wave that is hard synched to a fundamental pitch. This produces an effect similar to moving formants or pulse width modulation. The sawtooth oscillator has its phase reset

when the sync oscillator completes a cycle. This is not a band limited waveform, so it may alias.

The frequency of the slave synched sawtooth wave should always be greater than the syncFreq.

```
(audition
 (let ((f (line kr 100 800 12 remove-synth)))
   (out 0 (mul (sync-saw ar 100 f) 0.1))))
```

4.117 (t-choose trig array)

The output is selected randomly on recieving a trigger from an array of inputs. t-choose is a composite of ti-rand and select.

```
(audition
 (let* ((t (dust ar (mouse-x kr 1 1000 1 0.1)))
        (f (midi-cps (ti-rand 48 60 t)))
        (a (mce3 (sin-osc ar f 0)
                 (saw ar (mul f 2))
                 (pulse ar (mul f 0.5) 0.1))))
   (out 0 (mul (t-choose t a) 0.1))))
```

Note: all the ugens are continously running. This may not be the most efficient way if each input is cpu-expensive.

4.118 (sin-osc rate freq phase)

interpolating sine wavetable oscillator. This is the same as osc except that the table is a sine table of 8192 entries.

freq - frequency in Hertz phase - phase offset or modulator in radians

```
(audition (out 0 (mul (sin-osc ar 440 0) (mce2 0.15 0.25))))
```

Modulate freq

```
(let ((f (x-line kr 2000 200 1 remove-synth)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.5))))
```

Modulate freq

```
(let* ((f1 (x-line kr 1 1000 9 remove-synth))
       (f2 (mul-add (sin-osc ar f1 0) 200 800)))
  (audition (out 0 (mul (sin-osc ar f2 0) 0.25))))
```

Modulate phase

```
(let* ((f (x-line kr 20 8000 10 remove-synth))
       (p (mul (sin-osc ar f 0) (* pi 2))))
  (audition (out 0 (mul (sin-osc ar 800 p) 0.25))))
```

4.119 (klang rate freqScale freqOffset spec)

Bank of fixed oscillators. The UGen assistant klang.spec can help create the 'spec' entry. Note that the SC3 language reorders the inputs, the rsc3 client does not.

```
(let ((d (klang-data '(440 550 660 770 880 990 1000)
                     '(0.05 0.02 0.07 0.04 0.05 0.02 0.03)
                     (replicate 7 0))))
  (audition (out 0 (klang ar 1 0 d))))
```

4.120 (lag-in num-channels bus lag)

Set bus 10.

```
(with-sc3
 (lambda (fd)
   (send fd (c-set1 10 (random 200 8000)))))
```

Smooth a control rate input signal.

```
(audition (out 0 (mul (sin-osc ar (lag-in 1 10 1) 0) 0.1)))
```

4.121 (sound-in channel)

Read audio from the sound input hardware.

channel - input channel number to read, indexed from zero, can be mce.

```
(audition (out 0 (sound-in 0)))

(audition (out 0 (sound-in (mce2 0 1))))

(audition (out 0 (sound-in (mce4 0 2 1 3))))
```

4.122 (in num-channels rate bus)

Read signal from an audio or control bus.

Patching input to output.

```
(audition (out 0 (in 2 ar num-output-buses)))
```

Patching input to output, with summed delay.

```
(let ((i (in 2 ar num-input-buses)))
  (audition (out 0 (add i (delay-n i 0.5 0.5)))))
```

Write noise to bus 10, then read it out. The Mrg is ordered.

```
(audition (mrg2 (out 0 (in 1 ar 10))
                (out 10 (mul (pink-noise ar) 0.3))))
```

Reading a control bus.

```
(with-sc3
 (lambda (fd)
   (send fd (c-set1 0 (random 200 5000)))))

(audition (out 0 (mul (sin-osc ar (in 1 kr 0) 0) 0.1)))
```

4.123 (in-trig num-channels bus)

Generate a trigger anytime a bus is set.

Any time the bus is "touched" ie. has its value set (using "/c_set" etc.), a single impulse trigger will be generated. Its amplitude is the value that the bus was set to.

Run an oscillator with the trigger at bus 10.

```
(let* ((t (in-trig 1 10))
       (p (env-perc 0.01 1 1 (list -4 -4)))
       (e (env-gen kr t t 0 1 do-nothing p))
       (f (mul-add (latch t t) 440 880)))
  (audition (out 0 (mul (sin-osc ar f 0) e))))
```

Set bus 10.

```
(with-sc3
 (lambda (fd)
   (send fd (c-set1 10 0.5))))
```

4.124 (replace-out bufferindex inputs)

Send signal to a bus, overwrite existing signal.

```
(audition
 (mrg3 (out 0 (mul (sin-osc ar (mce2 330 331) 0) 0.1))
       (replace-out 0 (mul (sin-osc ar (mce2 880 881) 0) 0.1))
       (out 0 (mul (sin-osc ar (mce2 120 121) 0) 0.1))))
```

Compare to:

```
(audition
 (mrg3 (out 0 (mul (sin-osc ar (mce2 330 331) 0) 0.1))
       (out 0 (mul (sin-osc ar (mce2 880 881) 0) 0.1))
       (out 0 (mul (sin-osc ar (mce2 120 121) 0) 0.1))))
```

4.125 (local-in num-channels rate)

Define and read from buses local to a SynthDef

num-channels - the number of channels of local buses.

Localin defines buses that are local to the SynthDef. These are like the global buses, but are more convenient if you want to implement a self contained effect that uses a feedback processing loop. There can only be one audio rate and one control rate Localin per SynthDef. The audio can be written to the bus using local-out.

```
(let* ((a0 (mul (decay (impulse ar 0.3 0) 0.1) (mul (white-
noise ar) 0.2)))
       (a1 (add (local-in 2 ar) (mce2 a0 0)))
       (a2 (delay-n a1 0.2 0.2)))
  (audition (mrg2 (local-out (mul (mce-reverse a2) 0.8))
                  (out 0 a2))))
```

4.126 (offset-out bufferindex inputs)

output signal to a bus, the sample offset within the bus is kept exactly. This ugen is used where sample accurate output is needed.

```
(audition
 (mrg2 (offset-out 0 (impulse ar 5 0))
       (out 0 (mul (sin-osc ar 60 0) 0.1))))
```

```
(audition
 (mrg2 (out 0 (impulse ar 5 0))
       (out 0 (mul (sin-osc ar 60 0) 0.1))))
```

4.127 (in-feedback num-channels bus)

Read signal from a bus without erasing it, audio rate.

The output (out) ugens overwrite data on the bus, giving this bus a new timestamp so that any input (in) ugen can check if the data was written within the current cycle. The next cycle this data is still there, but in case of audio one normally doesn't want an in ugen to read it again, as it might cause feedback.

This is the reason why in ar checks the timestamp and ignores everything that was not written within this cycle. This means that nodes can only read data from a bus that was written by a preceeding node when using the in ar ugen which overwrites the old data. This is good for audio, but for control data it is more convenient to be able to read a bus from any place in the node order.

This is why in kr behaves differently and reads also data with a timestamp that is one cycle old. Now in some cases we want to be able to read audio from a bus independant of the current node order, which is the use of inFeedback. The delay introduced by this is at a maximum one block size, which equals about 0.0014 sec at the default block size and sample rate.

Audio feedback modulation.

```
(let ((f (mul-add (in-feedback 1 0) 1300 300)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.4))))
```

Evaluate these in either order and hear both tones.

```
(let ((b (add num-input-buses num-output-buses)))
  (audition (out 0 (in-feedback 1 b))))
```

```
(let ((b (add num-input-buses num-output-buses)))
  (audition (mrg2 (out b (mul (sin-osc ar 440 0) 0.1))
                  (out 0 (mul (sin-osc ar 660 0) 0.1)))))
```

Doubters consult this.

```
(let ((b (add num-input-buses num-output-buses)))
  (audition (out 0 (in 1 ar b))))
```

Resonator, see localout for variant.

```
(let* ((b (add num-input-buses num-output-buses))
       (p (in-feedback 1 b))
       (i (impulse ar 1 0))
       (d (delay-c (add i (mul p 0.995))
                   1
                   (sub (recip 440) (recip control-rate)))))
  (audition (mrg2 (offset-out b d) (offset-out 0 p))))
```

Compare with oscillator.

```
(audition (out 1 (mul (sin-osc ar 440 0) 0.2)))
```

4.128 (x-out buffer-index xfade inputs)

Send signal to a bus, crossfading with existing contents.

```
(let ((pair (lambda (a b) (mul (sin-osc ar (mce2 a b) 0) 0.1))))
  (audition
   (mrg4 (out  0 (pair 220 221))
         (x-out 0 (mouse-x kr 0 1 0 0.1) (pair 330 331))
         (x-out 0 (mouse-y kr 0 1 0 0.1) (pair 440 441))
         (out  0 (pair 120 121)))))
```

4.129 (out bufferindex inputs)

Send signal to an audio or control buss, mix with existing signal. The user is responsible for making sure that the number of channels match and that there are no conflicts.

```
(audition (out 0 (mul (sin-osc ar (mce2 330 331) 0) 0.1)))
```

4.130 (mix UGen)

Force multiple channel expansion and sum signals.

```
(let ((f (make-mce (list 600.2 622.0 641.3 677.7))))
  (audition (out 0 (mul (mix (f-sin-osc ar f 0)) 0.1))))
```

Expansion nests.

```
(let ((l (f-sin-osc ar (mce2 100  500) 0))
      (r (f-sin-osc ar (mce2 5000 501) 0)))
  (audition (out 0 (mul 0.05 (mix (mce2 l r))))))
```

4.131 (mix-fill n f)

```
(let ((n 6)
      (o (lambda (_) (mul (f-sin-osc ar (rand 200 700) 0) 0.1))))
  (audition (out 0 (mix-fill n o))))
```

4.132 (latch in trig)

Sample and hold. Holds input signal value when triggered.

in - input signal. trig - trigger. The trigger can be any signal. A trigger happens when the signal changes from non-positive to positive.

```
(audition
 (out 0 (mul (blip ar (mul-add (latch (white-
noise ar) (impulse ar 9 0))
                               400 500) 4)
             0.2)))
```

The above is just meant as example. lf-noise0 is a faster way to generate random steps :

```
(audition
 (out 0 (mul (blip ar (mul-add (lf-noise0 kr 9) 400 500) 4) 0.2)))
```

http://create.ucsb.edu/pipermail/sc-users/2006-December/029991.html

```
(let* ((n0 (mul-add (lf-noise2 kr 8) 200 300))
       (n1 (mul-add (lf-noise2 kr 3) 10 20))
       (s  (blip ar n0 n1))
       (x  (mouse-x kr 1000 (mul sample-rate 0.1) 1 0.1)))
  (audition
   (out 0 (latch s (impulse ar x 0)))))
```

4.133 (decay in decayTime)

Exponential decay. This is essentially the same as integrator except that instead of supplying the coefficient directly, it is caculated from a 60 dB decay time. This is the time required for the integrator to lose 99.9 % of its value or -60dB. This is useful for exponential decaying envelopes triggered by impulses.

Used as an envelope.

```
(audition
 (out 0 (mul (decay (impulse ar (x-line kr 1 50 20 remove-
synth) 0.25) 0.2)
             (pink-noise ar))))
```

4.134 (wrap-index bufnum in)

index into a table with a signal.

The input signal value is truncated to an integer value and used as an index into the table. out of range index values are wrapped cyclically to the valid range.

bufnum - index of the buffer in - the input signal.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 6 1))
   (send fd (b-setn1 0 0 (list 200 300 400 500 600 800)))
   (let ((f (wrap-index 0 (mouse-x kr 0 18 0 0.1))))
     (play fd (out 0 (mul (sin-osc ar f 0) 0.5))))))
```

4.135 (mouse-y rate minval maxval warp lag)

Report mouse location on root window of the machine that the synthesis server is running on. For a linear mapping set warp to 0, for an exponential mapping set warp to 1.

```
(audition
 (out 0 (mul (sin-osc ar (mouse-x kr 20 2000 0 0.1) 0)
             (mouse-y kr 0.01 0.1 0 0.1))))
```

4.136 (degree-to-key bufnum in octave)

Convert signal to modal pitch

The input signal value is truncated to an integer value and used as an index into an octave repeating table of note values. indices wrap around the table and shift octaves as they do.

bufnum - index of the buffer which contains the steps for each scale degree.

in - the input signal.

octave - the number of steps per octave in the scale.

```
(let* ((b 0)
       (p (list 0 2 3.2 5 7 9 10))
       (x (mouse-x kr 0 15 0 0.1))
       (k (degree-to-key 0 x 12))
       (c (lambda (n r)
            (let* ((o (mul (sin-osc ar (midi-
cps (add3 r k n)) 0) 0.1))
                   (t (lf-pulse ar (midi-cps (mce2 48 55)) 0.15 0.5))
                   (f (midi-cps (mul-add (sin-
osc kr 0.1 0) 10 r)))
                   (d (mul (rlpf t f 0.1) 0.1))
                   (m (add o d)))
              (add (comb-n m 0.31 0.31 2) m)))))
  (with-sc3
   (lambda (fd)
     (async fd (b-alloc 0 7 1))
     (send fd (b-setn1 0 0 p))
     (let ((n (mul (lf-noise1 kr (mce2 3 3.05)) 0.04)))
       (play fd (out 0 (mul (add (c n 48) (c n 72)) 0.25)))))))
```

4.137 (key-state rate keynum minval maxval lag)

Report the status of a particular key. A key is either pressed, or not pressed.

The keycode 38 is the A key on my keyboard. Under X the xev(1) command is useful in determining your keyboard layout.

```
(audition
 (out 0 (mul (sin-osc ar 800 0)
             (key-state kr 38 0 0.1 0.5))))
```

4.138 (mrg2 left right)

mrg2 defines a node indicating a multiple root graph.

```
(let ((l (out 0 (mul (sin-osc ar 300 0) 0.1)))
      (r (out 1 (mul (sin-osc ar 900 0) 0.1))))
  (audition
   (mrg2 l r)))
```

there is a leftmost rule, so that mrg nodes need not be terminal.

```
(let ((l (mul (sin-osc ar 300 0) 0.1))
```

```
      (r (out 1 (mul (sin-osc ar 900 0) 0.1))))
  (audition (out 0 (mrg2 l r))))
```

the leftmost node may be an mce node

```
(let ((l (mul (sin-osc ar (mce2 300 400) 0) 0.1))
      (r (out 1 (mul (sin-osc ar 900 0) 0.1))))
  (audition (out 0 (mrg2 l r))))
```

the implementation is not thorough

```
(let ((l (mul (sin-osc ar (mce2 300 400) 0) 0.1))
      (r (out 1 (mul (sin-osc ar 900 0) 0.1))))
  (audition (out 0 (add (mrg2 l r)
                        (mrg2 l r)))))
```

4.139 (mouse-button rate minval maxval lag)

Report the status of the first pointer button. The button is either pressed, or not pressed.

```
(audition
 (out 0 (mul (sin-osc ar 800 0)
             (mouse-button kr 0 0.1 0.1))))
```

4.140 (slew in up dn)

Has the effect of removing transients and higher frequencies.

```
(audition
 (out 0 (slew (mul (saw ar 800) 0.2) 400 400)))
```

4.141 (mouse-x rate minval maxval warp lag)

Cursor UGen. Report mouse location on root window of the machine that the synthesis server is running on. For a linear mapping set warp to 0, for an exponential mapping set warp to 1.

```
(import (rsc3))


(audition
 (out 0 (mul (sin-osc ar (mouse-x kr 40 10000 1 0.1) 0) 0.1)))
```

```
(audition
 (out 0 (mce2 (mul (sin-osc ar (mouse-x kr 20 2000 1 0.1) 0)
                   (mouse-y kr 0.01 0.1 0 0.1))
              (mul (sin-osc ar (mouse-y kr 20 2000 1 0.1) 0)
                   (mouse-x kr 0.01 0.1 0 0.1)))))
```

Auto-pilot variant

```
(audition
 (out 0 (mul (sin-osc ar (mouse-x* kr 40 10000 1 0.1) 0) 0.1)))
```

4.142 (decay2 in attackTime decayTime)

Exponential decay. decay has a very sharp attack and can produce clicks. decay2 rounds off the attack by subtracting one decay from another.

(decay2 in a d) is equivalent to (sub (decay in d) (Decay in a)).

Used as an envelope

```
(audition
 (out 0 (mul (decay2 (impulse ar (x-line kr 1 50 20 remove-
synth) 0.25)
                     0.01
                     0.2)
             (mul (f-sin-osc ar 600 0) 0.25))))
```

Compare the above with decay used as the envelope.

```
(audition
 (out 0 (mul (decay (impulse ar (x-line kr 1 50 20 remove-
synth) 0.25)
                    0.01)
             (mul (f-sin-osc ar 600 0) 0.25))))
```

4.143 (k2a in)

Control rate to audio rate converter.

To be able to play a control rate UGen into an audio rate UGen, sometimes the rate must be converted. k2a converts via linear interpolation.

in - input signal

```
(audition
 (out 0 (k2a (mul (white-noise kr) 0.3))))


(audition
 (out 0 (mce2 (k2a (mul (white-noise kr) 0.3))
              (mul (white-noise ar) 0.3))))


(let* ((block-size 64)
       (freq (mul (fdiv (mouse-x kr 0.1 40 1 0.1) block-
size) sample-rate)))
  (audition
   (out 0 (mul (mce2 (k2a (lf-noise0 kr freq))
                     (lf-noise0 ar freq))
               0.3))))
```

4.144 (mul-add a b c)

Functionally equivalent to (add (mul a b) c).

```
(let ((f (mul-add (lf-saw kr (mce2 10 9) 0) 200 400)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))


(let ((f (add (mul (lf-saw kr (mce2 10 9) 0) 200) 400)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.145 (clip2 a b)

Bilateral clipping. clips a to +/- b

```
(audition
 (out 0 (clip2 (f-sin-osc ar 400 0) 0.2)))


(audition
 (out 0 (clip2 (f-sin-osc ar 400 0) (line kr 0 1 8 remove-
synth))))
```

4.146 (Atan2 x y)

Returns the arctangent of y/x.

See also hypot.

add a pan to the hypot doppler examples by using atan2 to find the azimuth, or direction angle, of the sound source. Assume speakers at +/- 45 degrees and clip the direction to between those.

```
(let* ((x 10)
       (y (mul (lf-saw kr 1/6 0) 100))
       (distance (hypot x y))
       (amplitude (fdiv 40 (squared distance)))
       (sound (rlpf (mul (f-sin-osc ar 200 0) (lf-
pulse ar 31.3 0 0.4)) 400 0.3))
       (azimuth (atan2 y x))
       (loc (clip2 (fdiv azimuth (/ pi 2)) 1)))
  (audition
   (out 0 (pan2 (delay-l sound 55/172 (fdiv distance 344))
                loc
                amplitude))))
```

4.147 (trunc a b)

Truncate a to a multiple of b.

```
(let* ((x (mouse-x kr 60 4000 0 0.1))
       (f (trunc x 100)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.148 (sub a b)

subtraction, written '-' in sclang.

Silence.

```
(let ((z (f-sin-osc ar 800 0)))
  (audition
   (out 0 (sub z z))))
```

4.149 (round-up a b)

Rounds a up to the nearest multiple of b.

```
(let* ((x (mouse-x kr 60 4000 0 0.1))
       (f (round-up x 100)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))


(let ((n (line kr 24 108 6 remove-synth)))
  (audition
   (out 0 (mul (saw ar (midi-cps (round-up n 1))) 0.2))))
```

4.150 (ring4 a b)

Ring modulation variant. Return the value of ((a*a *b) - (a*b*b)). This is more efficient than using separate unit generators for the multiplies.

See also mul, ring1, ring2, ring3.

```
(audition
 (out 0 (mul (ring4 (f-sin-osc ar 800 0)
                    (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))


(let ((a (f-sin-osc ar 800 0))
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul (sub (mul3 a a b) (mul3 a b b))
               0.125))))
```

4.151 (pow a b)

Exponentiation, written ** in sclang. When the signal is negative this function extends the usual definition of exponentiation and returns neg(neg(a) ** b). This allows exponentiation of negative signal values by noninteger exponents.

```
(audition
 (out 0 (let ((a (mul (f-sin-osc ar 100 0) 0.1)))
          (mce2 a (pow a 10)))))
```

http://create.ucsb.edu/pipermail/sc-users/2006-December/029998.html

```
(let* ((n0 (mul-add (lf-noise2 kr 8) 200 300))
       (n1 (mul-add (lf-noise2 kr 3) 10 20))
```

```
      (s  (blip ar n0 n1))
      (x  (mouse-x kr 1000 (mul sample-rate 0.5) 1 0.1))
      (y  (mouse-y kr 1 24 1 0.1))
      (d  (latch s (impulse ar x 0)))
      (b  (u:round d (pow 0.5 y))))
  (audition
   (out 0 (mce2 d b))))
```

4.152 (ring1 a b)

Ring modulation plus first source. Return the value of ((a*b) + a). This is more efficient than using separate unit generators for the multiply and add.

See also mul, Ring1, Ring2, Ring3, Ring4.

```
(audition
 (out 0 (mul (ring1 (f-sin-osc ar 800 0)
                    (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))


(let ((a (f-sin-osc ar 800 0))
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul (add (mul a b) a) 0.125))))
```

4.153 (gt a b)

Greater than, written '>' in sclang. Signal is 1.0 if a > b, otherwise it is 0.0. Similarly LT is <, GE >=, LE <= and EQ ==. These can be useful for triggering purposes, among other things.

```
(let* ((o (sin-osc kr 1 0))
       (t (list (gt o 0)
                (ge o 0)
                (lt o 0)
                (le o 0)
                (eq o 0)
                (mul (lt o 0.001) (gt o -0.001))))
       (f (list 220
                330
                440
                550
```

```
              660
              770))
      (p (env-perc 0.01 1 1 (list -4 -4)))
      (e (env-gen kr (make-mce t) 0.1 0 1 do-nothing p)))
 (audition (out 0 (mix (mul (sin-osc ar (make-mce f) 0) e)))))
```

4.154 (add a b)

addition, written '+' in sclang.

```
(audition
 (out 0 (add (mul (f-sin-osc ar 800 0) 0.1)
             (mul (pink-noise ar) 0.1))))
```

DC offset.

```
(audition
 (out 0 (add (f-sin-osc ar 440 0) 0.1)))
```

4.155 (abs-dif a b)

Calculates the value of (abs (- a b). Finding the magnitude of the difference of two values is a common operation.

```
(audition
 (out 0 (mul (f-sin-osc ar 440 0)
             (abs-dif 0.2 (mul (f-sin-osc ar 2 0) 0.5)))))
```

4.156 (am-clip a b)

Two quadrant multiply, 0 when b <= 0, a*b when b > 0

```
(audition
 (out 0 (am-clip (white-noise ar)
                 (mul (f-sin-osc kr 1 0) 0.2))))
```

4.157 (ge a b)

See gt

4.158 (max a b)

Maximum.

```
(audition
   (out 0 (let ((z (f-sin-osc ar 500 0)))
            (u:max z (f-sin-osc ar 0.1 0)))))
```

4.159 (ring3 a b)

Ring modulation variant. Return the value of (a * a * b). This is more efficient than using separate unit generators for the multiplies.

See also mul, ring1, ring2, ring4.

```
(audition
 (out 0 (mul (ring3 (f-sin-osc ar 800 0)
                    (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))

(let ((a (f-sin-osc ar 800 0))
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul4 a a b 0.125))))
```

4.160 (thresh a b)

Signal thresholding. 0 when a < b, otherwise a.

```
(audition
 (out 0 (thresh (mul (lf-noise0 ar 50) 0.5) 0.45)))
```

4.161 (dif-sqr a b)

Difference of squares. Return the value of (a*a) - (b*b). This is more efficient than using separate unit generators for each operation.

```
(audition
   (out 0 (mul (dif-sqr (f-sin-osc ar 800 0)
                        (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
               0.125)))
```

```
(audition
   (out 0 (let ((a (f-sin-osc ar 800 0))
                (b (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0)))
            (mul (sub (mul a a) (mul b b)) 0.125))))
```

4.162 (excess a b)

clipping residual. Returns the difference of the original signal and its clipped form: (a - clip2(a,b)).

```
(audition
   (out 0 (excess (f-sin-osc ar 1000 0) (line kr 0 1 8 do-
nothing))))
```

```
(audition
 (out 0 (let ((a (f-sin-osc ar 1000 0))
              (b (line kr 0 1 8 do-nothing)))
          (sub a (clip2 a b)))))
```

4.163 (fold2 a b)

Bilateral folding. folds a to +/- b.

```
(audition
   (out 0 (fold2 (f-sin-osc ar 1000 0)
                 (line kr 0 1 8 do-nothing))))
```

4.164 (sqr-dif a b)

Square of the difference. Return the value of (a - b)**2. This is more efficient than using separate unit generators for each operation.

```
(audition
 (out 0 (mul (sqr-dif (f-sin-osc ar 800 0)
                      (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))
```

```
(let ((a (f-sin-osc ar 800 0))
```

```
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul (mul (sub a b) (sub a b))
               0.125))))
```

4.165 (hypot x y)

Returns the square root of the sum of the squares of a and b. Or equivalently, the distance from the origin to the point (x, y).

```
(audition
   (out 0 (mul (sin-osc ar 440 0)
               (hypot (mouse-x kr 0 0.1 0 0.1)
                      (mouse-y kr 0 0.1 0 0.1)))))
```

Object travels 200 meters in 6 secs (=120kph) passing 10 meters from the listener. The speed of sound is 344 meters/sec.

```
(let* ((x 10)
       (y (mul (lf-saw kr 1/6 0) 100))
       (distance (hypot x y))
       (velocity (slope distance))
       (pitch-ratio (fdiv (sub 344 velocity) 344))
       (amplitude (fdiv 10 (squared distance))))
  (audition
   (out 0 (mul (f-sin-osc ar (mul 1000 pitch-ratio) 0)
               amplitude))))


(let* ((x 10)
       (y (mul (lf-saw kr 1/6 0) 100))
       (distance (hypot x y))
       (amplitude (fdiv 40 (squared distance)))
       (sound (rlpf (mul (f-sin-osc ar 200 0) (lf-
pulse ar 31.3 0 0.4)) 400 0.3)))
  (audition
   (out 0 (mul (delay-l sound 55/172 (fdiv distance 344))
               amplitude))))
```

4.166 (sqr-sum a b)

Square of the difference. Return the value of (a + b)**2. This is more efficient than using separate unit generators for each operation.

```
(audition
 (out 0 (mul (sqr-sum (f-sin-osc ar 800 0)
                      (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))

(let ((a (f-sin-osc ar 800 0))
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul (mul (add a b) (add a b))
               0.125))))
```

4.167 (sum-sqr a b)

Return the value of (a*a) + (b*b). This is more efficient than using separate unit generators for each operation.

```
(audition
 (out 0 (mul (sum-sqr (f-sin-osc ar 800 0)
                      (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))

(let ((a (f-sin-osc ar 800 0))
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul (add (mul a a) (mul b b)) 0.125))))
```

4.168 (le a b)

See gt

4.169 eq

See gt

4.170 (scale-neg a b)

Scale negative part of input wave. a * b when a < 0, otherwise a.

```
(audition
 (out 0 (scale-neg (f-sin-osc ar 500 0) (line ar 1 -1 4 remove-
synth))))
```

4.171 (ring2 a b)

Ring modulation plus both sources. Return the value of ((a*b) + a + b). This is more efficient than using separate unit generators for the multiply and add.

See also mul, Ring1, Ring3, Ring4.

```
(audition
 (out 0 (mul (ring2 (f-sin-osc ar 800 0)
                    (f-sin-osc ar (x-line kr 200 500 5 do-
nothing) 0))
             0.125)))

(let ((a (f-sin-osc ar 800 0))
      (b (f-sin-osc ar (x-line kr 200 500 5 do-nothing) 0)))
  (audition
   (out 0 (mul (add3 (mul a b) a b) 0.125))))
```

4.172 (Mod a b)

Modulo, written % in sclang. outputs a modulo b.

```
(audition
   (out 0 (u:mod (f-sin-osc ar 100 4) 1)))
```

4.173 (fdiv a b)

Division, written '/' in sclang.

Division can be tricky with signals because of division by zero.

```
(audition
   (out 0 (fdiv (mul (pink-noise ar) 0.1)
                (mul (f-sin-osc kr 10 0.5) 0.75))))
```

4.174 (mul a b)

multiplication, written '*' in sclang.

```
(audition
 (out 0 (mul (sin-osc ar 440 0) 0.5)))
```

Creates a beating effect (subaudio rate).

```
(audition
 (out 0 (mul3 (f-sin-osc kr 10 0)
              (pink-noise ar)
              0.5)))
```

Ring modulation.

```
(audition
 (out 0 (mul3 (sin-osc ar (x-line kr 100 1001 10 do-nothing) 0)
              (sync-saw ar 100 200)
              0.25)))
```

4.175 (min a b)

Minimum.

```
(audition
 (out 0 (let ((z (f-sin-osc ar 500 0)))
          (u:min z (f-sin-osc ar 0.1 0)))))
```

4.176 (lt a b)

See gt

4.177 (wrap2 a b)

Bilateral wrapping. wraps input wave to +/- b.

```
(audition
 (out 0 (wrap2 (f-sin-osc ar 1000 0)
               (line kr 0 1.01 8 do-nothing))))
```

4.178 (round a b)

Rounds a to the nearest multiple of b.

```
(let* ((x (mouse-x kr 60 4000 0 0.1))
       (f (u:round x 100)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))


(let ((n (line kr 24 108 6 remove-synth)))
  (audition
   (out 0 (mul (saw ar (midi-cps (u:round n 1))) 0.2))))
```

4.179 (hasher in)

Returns a unique output value from zero to one for each input value according to a hash function. The same input value will always produce the same output value. The input need not be from zero to one.

```
(audition (out 0 (mul (hasher (line ar 0 1 1 2)) 0.2)))
```

4.180 (rand-seed rate trig seed)

When the trigger signal changes from nonpositive to positve, the synth's random generator seed is reset to the given value. All other synths that use the same random number generator reproduce the same sequence of numbers again.

See also: randID.

Start a noise patch

```
(let ((n (add (mul (white-noise ar) (mce2 0.05 0.05)) (dust2 ar (mce2 70 70))))
      (f (mul-add (lf-noise1 kr 3) 5500 6000)))
  (audition (out 0 (add (resonz (mul n 5) f 0.5) (mul n 0.5)))))
```

Reset the seed at a variable rate.

```
(audition (mrg2 (rand-seed kr (impulse kr (mouse-
x kr 0.1 100 0 0.1) 0) 1956)
                0))
```

4.181 (lfd-noise0 rate freq)

(lfd-noise1 rate freq) (lfd-noise3 rate freq)

lfd-noise0: Dynamic step noise. Like lf-noise0, it generates random values at a rate given by the freq argument, with two differences: no time quantization, and fast recovery from low freq values.

lfd-noise1: Dynamic ramp noise. Like lf-noise1, it generates linearly interpolated random values at a rate given by the freq argument, with two differences: no time quantization, and fast recovery from low freq values.

lfd-noise3: Dynamic cubic noise. Like Lf-Noise3, it generates linearly interpolated random values at a rate given by the freq argument, with two differences: no time quantization, and fast recovery from low freq values.

lf-noise0,1,3 quantize to the nearest integer division of the samplerate, and they poll the freq argument only when scheduled, and thus seem to hang when freqs get very low).

If you don't need very high or very low freqs, or use fixed freqs, lf-noise0,1,3 is more efficient.

Try wiggling mouse quickly; Lf-Noise frequently seems stuck, LFDNoise changes smoothly.

```
(audition
 (out 0 (mul (lf-noise0 ar (mouse-x kr 0.1 1000 1 0.1)) 0.1)))


(audition
 (out 0 (mul (lfd-noise0 ar (mouse-x kr 0.1 1000 1 0.1)) 0.1)))
```

silent for 2 secs before going up in freq

```
(audition
 (out 0 (mul (lf-noise0 ar (x-line kr 0.5 10000 3 remove-
synth)) 0.1)))


(audition
 (out 0 (mul (lfd-noise0 ar (x-line kr 0.5 10000 3 remove-
synth)) 0.1)))
```

lf-noise quantizes time steps at high freqs, lfd-noise does not:

```
(audition
 (out 0 (mul (lf-noise0 ar (x-line kr 1000 20000 10 remove-
synth)) 0.1)))


(audition
 (out 0 (mul (lfd-noise0 ar (x-line kr 1000 20000 10 remove-
synth)) 0.1)))
```

4.182 (lfclip-noise rate freq)

randomly generates the values -1 or +1 at a rate given by the nearest integer division of the sample rate by the freq argument. It is probably pretty hard on your speakers. The freq argument is the approximate rate at which to generate random values.

```
(audition (out 0 (mul (lfclip-noise ar 1000) 0.1)))
```

Modulate frequency

```
(let ((f (x-line kr 1000 10000 10 remove-synth)))
  (audition (out 0 (mul (lfclip-noise ar f) 0.1))))
```

Use as frequency control

```
(let ((f (mul-add (lfclip-noise kr 4) 200 600)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.183 See lf-noise0

4.184 (clip-noise rate)

Generates noise whose values are either -1 or 1. This produces the maximum energy for the least peak to peak amplitude.

```
(audition (out 0 (mul (clip-noise ar) 0.2)))
```

4.185 (ti-rand lo hi trig)

Generates a random integer value in uniform distribution from lo to hi each time the trig signal changes from nonpositive to positive values

```
(let ((p (ti-rand -1 1 (dust kr 10))))
  (audition (out 0 (pan2 (pink-noise ar) p 0.2))))


(let ((f (mul-add (ti-rand 4 12 (dust kr 10)) 150 (mce2 0 1))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.186 (lf-noise0 rate freq)

(lf-noise1 rate freq) (lf-noise2 rate freq)

lf-noise0 is step noise. Generates random values at a rate given by the nearest integer division of the sample rate by the freq argument.

lf-noise1 is ramp noise. Generates linearly interpolated random values at a rate given by the nearest integer division of the sample rate by the freq argument.

lf-noise2 is quadratic noise. Generates quadratically interpolated random values at a rate given by the nearest integer division of the sample rate by the freq argument.

```
(audition (out 0 (mul (lf-noise0 ar 1000) 0.25)))


(audition (out 0 (mul (lf-noise1 ar 1000) 0.25)))


(audition (out 0 (mul (lf-noise2 ar 1000) 0.25)))
```

Modulate frequency.

```
(audition (out 0 (mul (lf-noise0 ar (x-line kr 1000 10000 10 remove-
synth)) 0.25)))


(audition (out 0 (mul (lf-noise1 ar (x-line kr 1000 10000 10 remove-
synth)) 0.25)))


(audition (out 0 (mul (lf-noise2 ar (x-line kr 1000 10000 10 remove-
synth)) 0.25)))
```

Use as frequency control.

```
(audition (out 0 (mul (sin-osc ar (mul-add (lf-
noise0 kr 4) 400 450) 0) 0.2)))


(audition (out 0 (mul (sin-osc ar (mul-add (lf-
noise1 kr 4) 400 450) 0) 0.2)))


(audition (out 0 (mul (sin-osc ar (mul-add (lf-
noise2 kr 4) 400 450) 0) 0.2)))
```

4.187 (pink-noise rate)

Generates noise whose spectrum falls off in power by 3 dB per octave. This gives equal power over the span of each octave. This version gives 8 octaves of pink noise.

```
(audition (out 0 (mul (pink-noise ar) 0.25)))
```

4.188 (rand lo hi)

Generates a single random value in uniform distribution from lo to hi. It generates this when the SynthDef first starts playing, and remains fixed for the duration of the synth's existence.

```
(let* ((a (line kr 0.2 0 0.1 2))
       (p (rand -1 1))
       (s (mul (f-sin-osc ar (rand 200 1200) 0) a)))
  (audition (out 0 (pan2 s p 1))))
```

4.189 (gray-noise rate)

Generates noise which results from flipping random bits in a word. This type of noise has a high RMS level relative to its peak to peak level. The spectrum is emphasized towards lower frequencies.

```
(audition (out 0 (mul (gray-noise ar) 0.1)))
```

4.190 See lfd-noise0

4.191 (i-rand lo hi)

Generates a single random integer value in uniform distribution from 'lo' to 'hi'.

```
(let ((f (i-rand 200 1200))
      (a (line kr 0.2 0 0.1 remove-synth)))
  (audition (out 0 (mul (f-sin-osc ar f 0) a))))
```

4.192 (n-rand lo hi n)

Generates a single random float value in a sum of 'n' uniform distributions from 'lo' to 'hi'.

n = 1 : uniform distribution - same as rand n = 2 : triangular distribution n = 3 : smooth hump as n increases, distribution converges towards gaussian

```
(let ((f (mul (n-rand 1200 4000 2) (mce2 2 5)))
      (a (line kr 0.2 0 0.01 remove-synth)))
  (audition (out 0 (mul (f-sin-osc ar f 0) a))))
```

4.193 (lfdclip-noise rate freq)

Like lfclip-noise, it generates the values -1 or +1 at a rate given by the freq argument, with two differences: no time quantization, and fast recovery from low freq values.

(lfclip-noise, as well as lf-noise0,1,2 quantize to the nearest integer division of the sampler-ate, and they poll the freq argument only when scheduled, and thus seem to hang when freqs get very low).

If you don't need very high or very low freqs, or use fixed freqs, lfclip-noise is more efficient.

Try wiggling mouse quickly; LFNoise frequently seems stuck, LFDNoise changes smoothly.

```
(let ((f (mul-add (lfclip-noise ar (mouse-
x kr 0.1 1000 1 0.1)) 200 500)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))


(let ((f (mul-add (lfdclip-noise ar (mouse-
x kr 0.1 1000 1 0.1)) 200 500)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

LFNoise quantizes time steps at high freqs, LFDNoise does not:

```
(let ((f (x-line kr 1000 20000 10 remove-synth)))
  (audition (out 0 (mul (lfclip-noise ar f) 0.1))))


(let ((f (x-line kr 1000 20000 10 remove-synth)))
  (audition (out 0 (mul (lfdclip-noise ar f) 0.1))))
```

4.194 (coin-gate prob in)

When it receives a trigger, it tosses a coin, and either passes the trigger or doesn't.

```
(let ((f (t-rand 300 400 (Coingate 0.8 (impulse kr 10 0)))))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))


(let* ((p 0.2)
```

```
      (t (mul (impulse ar 20 0) (add (sin-osc kr 0.5 0) 1)))
      (t* (t-exp-rand (Mce 1000 1000) 12000 t))
      (i (lambda () (Coingate (+ p (rand 0 0.1)) (mul t 0.5))))
      (s (lambda () (ringz (i) t* 0.01)))
      (ignore (lambda (f) (lambda (_) (f)))))
  (audition (out 0 (mix/fill 3 (ignore s)))))
```

4.195 (t-exp-rand lo hi trig)

Generates a random float value in exponential distribution from lo to hi each time the trig signal changes from nonpositive to positive values lo and hi must both have the same sign and be non-zero.

```
(let* ((t (dust kr 10))
       (f (t-exp-rand 300 3000 t)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.196 (t-rand lo hi trig)

Generates a random float value in uniform distribution from lo each time the trig signal changes from nonpositive to positive values

```
(let* ((t (dust kr (mce2 5 12)))
       (f (t-rand (mce2 200 1600) (mce2 500 3000) t)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.2))))
```

4.197 (white-noise rate)

Generates noise whose spectrum has equal power at all frequencies.

```
(audition (out 0 (mul (white-noise ar) 0.15)))
```

Noise generators constructors are unique, to share noise UGens values must be explictly stored and reused.

```
(audition (out 0 (mul (sub (white-noise ar) (white-
noise ar)) 0.15)))


(let ((n (white-noise ar)))
  (audition (out 0 (sub n n))))
```

4.198 (dust2 rate density)

Generates random impulses from -1 to +1. The 'density' is in impulses per second.

```
(audition (out 0 (mul (dust2 ar 200) 0.5)))

(let ((r (x-line kr 20000 2 10 remove-synth)))
  (audition (out 0 (mul (dust2 ar r) 0.5))))
```

4.199 See lfd-noise0

4.200 (rand-id rate id)

Choose which random number generator to use for this synth. All synths that use the same generator reproduce the same sequence of numbers when the same seed is set again.

See also: rand-seed.

Graphs to generate noise in the context of a given RNG and to reset a specified RNG.

```
(with-sc3
 (lambda (fd)
   (send-synth
    fd "r"
    (letc ((bus 0)
           (id 1))
      (mrg2 (rand-id ir id)
            (out bus (add (mul (white-noise ar) 0.05)
                          (dust2 ar 70))))))
   (send-synth
    fd "s"
    (letc ((seed 1910) (id 1))
      (mrg2 (rand-id kr id)
            (rand-seed kr
                       (impulse kr (mul-add (f-sin-
osc kr 0.2 0) 10 11) 0)
                       seed))))))
```

Start two noise synths on left and right channel with a different randgen id

```
(with-sc3
 (lambda (fd)
   (send fd (s-new2 "r" 1001 1 1 "bus" 0 "id" 1))
   (send fd (s-new2 "r" 1002 1 1 "bus" 1 "id" 2))))
```

Reset the seed of randgen 1

```
(with-sc3 (lambda (fd) (send fd (s-new1 "s" 1003 1 1 "id" 1))))
```

Change the target RNG with ID 2, ie. effect right channel.

```
(with-sc3 (lambda (fd) (send fd (n-set1 1003 "id" 2))))
```

free noise nodes.

```
(with-sc3
 (lambda (fd)
   (send fd (n-free1 1001))
   (send fd (n-free1 1002))
   (send fd (n-free1 1003))))
```

4.201 See lf-noise0

4.202 (mantissa-mask in bits)

Masks off bits in the mantissa of the floating point sample value. This introduces a quantization noise, but is less severe than linearly quantizing the signal.

in - input signal bits - the number of mantissa bits to preserve. a number from 0 to 23.

```
(let ((s (mul (sin-osc ar (mul-add (sin-osc kr 0.2 0) 400 500) 0) 0.4)))
  (audition (out 0 (mantissa-mask s 3))))
```

4.203 (dust rate density)

Generates random impulses from 0 to +1 at a rate determined by the density argument.

```
(audition (out 0 (mul (dust ar 200) 0.5)))
```

```
(let ((r (x-line kr 20000 2 10 remove-synth)))
  (audition (out 0 (mul (dust ar r) 0.5))))
```

4.204 (lin-rand lo hi minmax)

Generates a single random float value in linear distribution from lo to hi, skewed towards lo if minmax < 0, otherwise skewed towards hi.

```
(let ((f (lin-rand 200 10000 (mce2 -1 1)))
      (a (line kr 0.4 0 0.01 remove-synth)))
  (audition (out 0 (mul (f-sin-osc ar f 0) a))))
```

4.205 (exp-rand lo hi)

Generates a single random float value in an exponential distributions from ‘lo’ to ‘hi’.

```
(let ((f (exp-rand 100 8000))
      (a (line kr 0.5 0 0.01 remove-synth)))
  (audition (out 0 (mul (f-sin-osc ar f 0) a))))
```

4.206 brown-noise

```
(brown-noise r) → noise?
  r : rate
```

Allocate buffer space.

Generates noise whose spectrum falls off in power by 6 dB per octave.

```
(audition (out 0 (mul (brown-noise ar) 0.1)))
```

4.207 (u:log a)

Reciprocal.

```
(let* ((a (line ar -2 2 2 remove-synth))
       (b (u:log a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.208 (frac a)

Fractional part.

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (frac a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.209 (arc-tan a)

Arc tan.

```
(let* ((a (line kr -1 1 1 remove-synth))
       (b (sub (fdiv (arc-tan a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.210 (distort a)

Nonlinear distortion.

```
(audition
   (out 0 (mul (distort (mul (f-sin-osc ar 500 0.0)
                             (x-line kr 0.1 10 10 do-nothing)))
               0.25)))
```

4.211 (tan-h a)

Tangent.

```
(let* ((a (line kr (- pi) pi 1 remove-synth))
       (b (sub (fdiv (tan-h a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.212 (u:floor a)

Round signal down.

```
(let* ((x (mouse-x kr 65 95 0 0.1))
       (f (midi-cps (u:floor x))))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.213 (cps-oct a)

Convert cycles per second to decimal octaves.

```
(audition
 (let ((f (oct-cps (cps-oct (x-line kr 600 900 6 remove-synth)))))
   (out 0 (mul (saw ar f) 0.2))))
```

4.214 (db-amp a)

Convert decibels to linear amplitude.

```
(audition
   (out 0 (mul (f-sin-osc ar 800 0.0)
               (db-amp (line kr -3 -40 10 remove-synth)))))
```

4.215 (u:sqrt a)

Square root. The definition of square root is extended for signals so that sqrt(a) when a<0 returns -sqrt(-a).

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (u:sqrt a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.216 (soft-clip a)

Nonlinear distortion. Distortion with a perfectly linear region from -0.5 to +0.5.

```
(audition
 (out 0 (mul (soft-clip (mul (f-sin-osc ar 500 0.0)
                             (x-line kr 0.1 10 10 do-nothing)))
             0.25)))
```

4.217 (cps-midi a)

Convert cycles per second to MIDI note.

```
(let ((f (line kr 600 900 5 remove-synth)))
  (audition
   (out 0 (mul (saw ar (midi-cps (cps-midi f))) 0.1))))
```

4.218 (is-strictly-positive a)

Predicate to determine if a value is strictly positive.

```
(define (is-strictly-positive a)
  (gt a 0.0))


(let* ((a (line ar -1 1 1 remove-synth))
       (b (is-strictly-positive a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.219 (u:tan a)

Tangent.

```
(let* ((a (line kr (- pi) pi 1 remove-synth))
       (b (sub (fdiv (u:tan a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.220 (cos-h a)

Cosine.

```
(let* ((a (line kr (- pi) pi 1 remove-synth))
       (b (sub (fdiv (cos-h a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.221 (amp-db a)

Convert linear amplitude to decibels.

```
(audition
 (out 0 (mul (f-sin-osc ar 800 0.0)
             (db-amp (amp-db (line kr 0.5 0.0 5 remove-synth))))))
```

```
(let* ((x (mouse-x kr -60 0 0 0.1))
       (f (mul-add (db-amp x) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.222 (abs a)

Absolute value.

```
(audition (out 0 (u:abs (mul (sync-saw ar 100 440) 0.1))))
```

4.223 (log10 a)

Base ten logarithm.

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (log10 a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.224 (midi-cps a)

Convert MIDI note to cycles per second.

```
(audition
 (out 0 (mul (saw ar (midi-cps (line kr 24 108 10 remove-
synth))) 0.2)))
```

4.225 (is-positive a)

Predicate to determine if a value is positive.

```
(define (is-positive a)
  (ge a 0.0))

(let* ((a (line ar -1 1 1 remove-synth))
       (b (is-positive a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.226 (sign a)

Sign function. -1 when a < 0, +1 when a > 0, 0 when a is 0

```
(let* ((a (line ar -1 1 1 remove-synth))
       (b (sign a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.227 (neg a)

Negation.

```
(let ((s (sin-osc ar 440 0)))
  (audition
   (out 0 (mce2 (mul s 0.1)
                (add s (neg s))))))
```

4.228 (log2 a)

Base two logarithm.

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (log2 a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.229 (ceil a)

Round signal up.

```
(let* ((x (mouse-x kr 65 95 0 0.1))
       (f (midi-cps (mce2 (u:floor x) (ceil x)))))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.230 (arc-cos a)

Arc cosine.

```
(let* ((a (line kr -1 1 1 remove-synth))
       (b (sub (fdiv (arc-cos a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.231 (exp a)

Exponential.

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (u:exp a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.232 (squared a)

Square.

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (squared a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.233 (arc-sin a)

Arc sine.

```
(let* ((a (line kr -1 1 1 remove-synth))
       (b (sub (fdiv (arc-sin a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.234 (cubed a)

Cube.

```
(let* ((a (line ar -2 2 3 remove-synth))
       (b (cubed a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.235 (is-negative a)

Predicate to determine if a value is negative.

```
(define (is-negative a)
  (lt a 0.0))


(let* ((a (line ar -1 1 1 remove-synth))
       (b (is-negative a))
       (f (mul-add (mce2 a b) 600 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.236 (oct-cps a)

Convert decimal octaves to cycles per second.

```
(audition
 (let ((f (oct-cps (line kr 2 9 6 remove-synth))))
   (out 0 (mul (saw ar f) 0.2))))


(audition
 (let ((f (oct-cps (u:round (line kr 2 9 6 remove-
synth) (/ 1 12)))))
   (out 0 (mul (saw ar f) 0.2))))
```

4.237 (u:sin a)

Sine.

```
(let* ((a (line kr (- pi) pi 1 remove-synth))
       (b (sub (fdiv (u:sin a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.238 (sin-h a)

Sine.

```
(let* ((a (line kr (- pi) pi 1 remove-synth))
       (b (sub (fdiv (sin-h a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.239 (cos a)

Cosine.

```
(let* ((a (line kr (- pi) pi 1 remove-synth))
       (b (sub (fdiv (u:cos a) (/ pi 2)) 1))
       (f (mul-add b 900 1600)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.240 (pv-mul bufferA bufferB)

4.241 (pv-mag-squared buffer)

4.242 (pv-min bufferA bufferB)

4.243 (pv-mag-noise buffer)

Magnitudes are multiplied with noise.

buffer - fft buffer.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (h (pv-mag-noise f)))
  (audition (out 0 (mul (ifft* h) 0.5))))
```

4.244 (pv-mag-below buffer threshold)

Pass bins below a threshold. Pass only bands where the magnitude is below 'threshold'. This value is not normalized and is therefore dependant on the buffer size.

See pv-mag-above

4.245 (convolution in kernel framesize)

Strict convolution of two continuously changing inputs. Also see [convolution2] for a cheaper CPU cost alternative for the case of a fixed kernel which can be changed with a trigger message.

in - processing target kernel - processing kernel. framesize - size of fft frame, must be a power of two

```
(audition
 (let ((input (sound-in (mce2 0 1)))
       (kernel (white-noise ar)))
   (out 0 (mul (convolution input kernel 2048) 0.1))))

(let ((a 2048)
      (b 0))
  (with-sc3
   (lambda (fd)
     (async fd (b-alloc b a 1))
     (send fd (b-set1 b 0 1.0))
     (replicate-m 100 (send fd (b-set1 b (random-
integer a) (random 0.0 1.0))))
     (play fd (out 0 (mul (convolution
                           (sound-in (mce2 0 1))
                           (play-buf 1 b (buf-rate-
scale kr b) 1 0 1)
                           (* 2 a))
                          0.2))))))
```

4.246 (pv-jensen-andersen buffer propsc prophfe prophfc propsf threshold waittime)

fft feature detector for onset detection based on work described in Jensen,K. & Andersen, T. H. (2003). Real-time Beat Estimation Using Feature Extraction. in Proceedings of the Computer Music Modeling and Retrieval Symposium, Lecture Notes in Computer Science. springer Verlag.

First order derivatives of the features are taken. Threshold may need to be set low to pick up on changes.

buffer - fft buffer to read from. propsc - Proportion of spectral centroid feature. prophfe - Proportion of high frequency energy feature. prophfc - Proportion of high frequency content feature. propsf - Proportion of spectral flux feature. threshold - Threshold level for allowing a detection waittime - If triggered, minimum wait until a further frame can cause another spot (useful to stop multiple detects on heavy signals)

Default values in sclang are: propsc=0.25, prophfe=0.25, prophfc=0.25, propsf=0.25, threshold=1.0, waittime=0.04.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 2048 1))))

(let* ((source (sound-in 0))
       (detect (pv-jensen-andersen (fft* 0 source)
                                   0.25 0.25 0.25 0.25
                                   (mouse-x kr 0.01 1.0 1 0.1)
                                   0.04)))
  (audition
   (out 0 (mul (sin-osc ar (mce2 440 445) 0)
               (decay (mul 0.1 detect) 0.1)))))
```

4.247 (pv-phase-shift270 buffer)

Swap the real and imaginary components of every bin at 'buffer' and swap the sign of the real components.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))

(let ((n (mul (white-noise ar) 0.1)))
  (audition (out 0 (ifft* (pv-phase-shift270 (fft* 10 n))))))
```

4.248 (pv-hainsworth-foote buffer proph propf threshold waittime)

fft onset detector based on work described in

Hainsworth, S. (2003) Techniques for the Automated Analysis of Musical Audio. PhD, University of Cambridge engineering dept. See especially p128. The Hainsworth metric is a modification of the Kullback Liebler distance.

The onset detector has general ability to spot spectral change, so may have some ability to track chord changes aside from obvious transient jolts, but there's no guarantee it won't be confused by frequency modulation artifacts.

Hainsworth metric on it's own gives good results but Foote might be useful in some situations: experimental.

buffer - fft buffer to read from

proph - What strength of detection signal from Hainsworth metric to use.

propf - What strength of detection signal from Foote metric to use. The Foote metric is normalised to [0.0,1.0]

threshold - Threshold hold level for allowing a detection

waittime - If triggered, minimum wait until a further frame can cause another spot (useful to stop multiple detects on heavy signals)

Default values in sclang are: proph=0.0, propf=0.0, threshold=1.0, waittime=0.04.

–

Just Hainsworth metric with low threshold

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 2048 1))))


(let* ((source (sound-in 0))
       (detect (pv-hainsworth-foote (fft* 0 source)
                                    1.0
                                    0.0
                                    (mouse-x kr 0.01 1.0 1 0.1)
                                    0.04)))
  (audition
   (out 0 (mul3 (sin-osc ar (mce2 440 445) 0)
                (decay (mul 0.1 detect) 0.1)
                0.1))))
```

Just Hainsworth metric, spot note transitions.

```
(let* ((src (mul (lf-saw ar (mul-add (lf-noise0 kr 1) 90 400) 0) 0.5))
       (dtc (pv-hainsworth-foote (fft* 0 src)
                                 1.0
                                 0.0
                                 0.9
```

```
                              0.5))
        (cmp (mul (sin-osc ar 440 0)
                  (decay (mul 0.1 dtc) 0.1))))
  (audition
   (out 0 (mul (mce2 src cmp) 0.1))))
```

Just Foote metric. Foote never triggers with threshold over 1.0, threshold under mouse control.

```
(let* ((src (sound-in 0))
       (dtc (pv-hainsworth-foote (fft* 0 src)
                                 0.0
                                 1.0
                                 (mouse-x kr 0.0 1.1 0 0.1)
                                 0.02))
       (cmp (mul (sin-osc ar 440 0)
                 (decay (mul 0.1 dtc) 0.1))))
  (audition
   (out 0 (mul (mce2 src cmp) 0.1))))
```

4.249 (pv-phase-shift buffer shift)

add 'shift' to the phase component of every bin at 'buffer'.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))


(let ((n (mul (white-noise ar) 0.1))
      (x (mouse-x kr 0 1 0 0.1)))
  (audition (out 0 (ifft* (pv-phase-shift (fft* 10 n) x)))))
```

4.250 (pv-copy bufferA bufferB)

Copies the spectral frame in bufferA to bufferB at that point in the chain of PV UGens. This allows for parallel processing of spectral data without the need for multiple fft UGens, and to copy out data at that point in the chain for other purposes. bufferA and bufferB must be the same size.

bufferA - source buffer. bufferB - destination buffer.

```
(with-sc3
```

```
 (lambda (fd)
   (async fd (b-alloc 0 2048 1))
   (async fd (b-alloc 1 2048 1))))
```

Proof of concept, silence

```
(let* ((in (lfclip-noise ar 100))
       (c0 (fft* 0 in))
       (c1 (pv-copy c0 1)))
  (audition (out 0 (sub (ifft* c0) (ifft* c1)))))
```

4.251 (pv-brick-wall buffer wipe)

Clears bins above or below a cutoff point. 'wipe' = a unit signal, from -1 to 0 the UGen acts as a low-pass filter, from 0 to 1 it acts as a high pass filter.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))

(let ((x (mouse-x kr -1 1 0 0.1))
      (c (fft* 10 (mul (white-noise ar) 0.2))))
  (audition (out 0 (ifft* (pv-brick-wall c x)))))
```

4.252 (pv-mag-smear buffer bins)

Average a bin's magnitude with its neighbors.

buffer - fft buffer.

bins - number of bins to average on each side of bin. As this number rises, so will CPU usage.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))

(let* ((dup (lambda (a) (mce2 a a)))
       (in (mul (lf-saw ar 500 0) (decay2 (mul (impulse ar 2 0) 0.2) 0.01 2)))
       (c0 (fft* 10 in))
       (c1 (pv-mag-smear c0 (mouse-x kr 0 100 0 0.1))))
  (audition (out 0 (mul 0.5 (dup (ifft* c1))))))
```

```
(let* ((dup (lambda (a) (mce2 a a)))
       (s (play-buf 1 12 (buf-rate-scale kr 12) 1 0 1))
       (x (mouse-x kr 0 100 0 0.1)))
  (audition (out 0 (mul 0.5 (dup (ifft* (pv-mag-
smear (fft* 10 s) x)))))))
```

4.253 (pv-mag-above buffer threshold)

Pass only bands where the magnitude is above 'threshold'. This value is not normalized and is therefore dependant on the buffer size.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc 11 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))
```

```
(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (x (mouse-x kr 1 100 0 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (c1 (fft* 10 a))
       (c2 (pv-copy c1 11))
       (c3 (pv-mag-below c1 x))
       (c4 (pv-mag-above c2 x)))
  (audition (out 0 (mul (mce2 (ifft* c3) (ifft* c4)) (mce2 y (sub 1 y))))))
```

```
(let* ((f1 (squared (mul-add (sin-osc kr 0.08 0) 6 6.2)))
       (f2 (mul-add (sin-osc kr f1 0) 100 800))
       (s (sin-osc ar f2 0))
       (x (mouse-x kr 1 1024 0 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (c1 (fft* 10 s))
       (c2 (pv-copy c1 11))
       (c3 (pv-mag-below c1 x))
       (c4 (pv-mag-above c2 x)))
  (audition (out 0 (mul (mce2 (ifft* c3) (ifft* c4)) (mce2 y (sub 1 y))))))
```

4.254 (pv-bin-shift buffer stretch shift)

Shift and scale the positions of the bins. Can be used as a very crude frequency shifter/scaler. Shifts the leftmost bin at 'buffer' by 'shift' places, the distance between subsequent bins is 'stretch'.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))

(define snd
  (let* ((f1 (squared (mul-add (sin-osc kr 0.08 0) 6 6.2)))
         (f2 (sin-osc kr f1 0)))
    (sin-osc ar (mul-add f2 100 800) 0)))

(audition (out 0 snd))

(audition
 (out 0 (mul
         (ifft*
          (pv-bin-shift
           (fft* 10 snd)
           (mouse-y kr 1 4 0 0.1)
           (mouse-x kr -10 100 0 0.1)))
         1/2)))
```

4.255 (fft buffer in hop wintype active)

(fft* b i) => (fft b i 0.5 0 1)

Fast fourier transform. The fast fourier transform analyzes the frequency content of a signal. fft uses a local buffer for holding the buffered audio. The inverse transform, Ifft, reconstructs an audio signal.

Note that the UGens the SC3 language provides do not use rate extensions, since only a single rate is valid for each UGen class. The fft and PV_ UGens must run at control rate, the ifft UGen at audio rate.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))

(let* ((s (mul (white-noise ar) 0.05))
       (c (fft* 10 s)))
  (audition (out 0 (ifft* c))))

(let* ((f1 (Squared (mul-add (sin-osc kr 0.08 0) 6 6.2)))
       (f2 (mul-add (sin-osc kr f1 0) 100 800))
       (s (sin-osc ar f2 0)))
  (audition (out 0 (ifft* (fft* 10 s)))))
```

4.256 (pv-bin-wipe bufferA bufferB wipe)

Combine low and high bins from two inputs

Copies low bins from one input and the high bins of the other.

bufferA - fft buffer A. bufferB - fft buffer B. wipe - can range between -1 and +1.

if wipe == 0 then the output is the same as inA. if wipe $>$ 0 then it begins replacing with bins from inB from the bottom up. if wipe $<$ 0 then it begins replacing with bins from inB from the top down.

```
(with-sc3
 (lambda (fd)
  (async fd (b-alloc 10 2048 1))
  (async fd (b-alloc 11 2048 1))
  (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((a (mul (white-noise ar) 0.2))
       (b (mul (sin-osc ar 100 0) 0.2))
       (f (fft* 10 a))
       (g (fft* 11 b))
       (h (pv-bin-wipe f g (mouse-x kr -1 1 0 0.1))))
  (audition (out 0 (mul (ifft* h) 0.5))))


(let* ((a (mul (white-noise ar) 0.2))
       (b (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (g (fft* 11 b))
       (h (pv-bin-wipe f g (mouse-x kr -1 1 0 0.1))))
  (audition (out 0 (mul (ifft* h) 0.5))))
```

4.257 (pv-copyPhase bufferA bufferB)

Combines magnitudes of first input and phases of the second input.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc 11 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((a (mul (white-noise ar) 0.2))
```

```
        (b (mul (sin-osc ar 100 0) 0.2))
        (f (fft* 10 a))
        (g (fft* 11 b))
        (h (pv-copy-phase f g)))
  (audition (out 0 (mul (ifft* h) 0.5))))


(let* ((a (mul (white-noise ar) 0.2))
       (b (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (g (fft* 11 b))
       (h (pv-copy-phase f g)))
  (audition (out 0 (mul (ifft* h) 0.5))))
```

4.258 (pv-phase-shift90 buffer)

Swap the real and imaginary components of every bin at 'buffer' and swap the of the imaginary components.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))


(let ((n (mul (white-noise ar) 0.1)))
  (audition (out 0 (ifft* (pv-phase-shift90 (fft* 10 n))))))
```

4.259 (convolution2 in bufnum trigger framesize)

Strict convolution with fixed kernel which can be updated using a trigger signal.

in - processing target bufnum - buffer index for the fixed kernel, may be modulated in combination with the trigger trigger - update the kernel on a change from <=0 to >0 framesize - size of fft frame, must be a power of two. convolution uses twice this number internally, maximum value you can give this argument is 2^16=65536. Note that it gets progressively more expensive to run for higher powers! 512, 1024, 2048, 4096 standard.

```
(with-sc3
 (lambda (fd)
   (for-each
    (lambda (b)
      (async fd (b-alloc b 2048 1)))
    (list 10 11 12))
   (for-each
```

```
    (lambda (n)
      (send fd (b-set1 10 (+ (* 400 n) 100) 1)))
    (enum-from-to 0 2))
   (for-each
    (lambda (n)
      (send fd (b-set1 11 (+ (* 20 n) 10) (random 0 1))))
    (enum-from-to 0 49))
   (for-each
    (lambda (n)
      (send fd (b-set1 12 (+ (* 40 n) 20) 1)))
    (enum-from-to 0 19))
   (send-synth
    fd "c"
    (letc ((k 0) (t 0))
      (let ((i (impulse ar 1 0)))
        (out 0 (mul (convolution2 i k t 2048) 0.5)))))))

(define send-to
  (lambda (m)
    (with-sc3
     (lambda (fd)
       (send fd m)))))

(define async-to
  (lambda (m)
    (with-sc3
     (lambda (fd)
       (async fd m)))))

(send-to (s-new1 "c" 1001 1 1 "k" 10))

(send-to (n-set1 1001 "k" 11))

(send-to (n-set1 1001 "t" 0))

(send-to (n-set1 1001 "t" 1))

(send-to (n-set1 1001 "k" 12))

(send-to (n-set1 1001 "t" 0))
```

```
(send-to (n-set1 1001 "t" 1))

(async-to (b-zero 12))

(for-each
 (lambda (n)
   (send-to (b-set1 12 (+ (* 20  n) 10)  1)))
 (enum-from-to 0 39))

(send-to (n-set1 1001 "t" 0))

(send-to (n-set1 1001 "t" 1))
```

With soundfile.

```
(async-to (b-alloc-read 10 "/home/rohan/audio/metal.wav" 0 0))

(let ((i (sound-in 0)))
  (audition (out 0 (mul (convolution2 i 10 0 512) 0.5))))
```

4.260 (pv-rect-comb2 bufferA bufferB numTeeth phase width)

4.261 (pv-add bufferA bufferB)

Complex addition: RealA + RealB, ImagA + ImagB

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc 11 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))

(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (g (fft* 11 (mul (sin-osc ar 440 0) 0.2)))
       (h (pv-add f g)))
  (audition (out 0 (mul (ifft* h) 0.5))))

(audition
 (out 0 (mul (add (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1)
                  (mul (sin-osc ar 440 0) 0.2))
             0.5)))
```

4.262 (pv-rand-comb buffer wipe trig)

randomly clear bins.

buffer = fft buffer. wipe = clear bins from input in a random order (0, 1). trig = select new random ordering.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))


(let ((dup (lambda (a) (mce2 a a)))
      (n (mul (white-noise ar) 0.5))
      (x (mouse-x kr 0.6 0.95 0 0.1))
      (t (impulse kr 0.4 0)))
  (audition (out 0 (dup (ifft* (pv-rand-comb (fft* 10 n) x t))))))
```

4.263 (pv-local-max buffer threshold)

Pass bins which are a local maximum

Passes only bins whose magnitude is above a threshold and above their nearest neighbors.

buffer - fft buffer. threshold - magnitude threshold.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (h (pv-local-max f (mouse-x kr 0 100 0 0.1))))
  (audition (out 0 (mul (ifft* h) 0.5))))
```

4.264 (pv-mag-mul bufferA bufferB)

4.265 (pv-conformal-map buffer real imag)

Applies the conformal mapping z send (z-a)/(1-za*) to the phase vocoder bins z with a given by the real and imag imputs to the UGen.

See http://mathworld.wolfram.com/ConformalMapping.html

buffer - buffer number of buffer to act on, passed in through a chain real - real part of a. imag - imaginary part of a.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 1024 1))
   (async fd (b-alloc 0 2048 1))))


(audition
 (out 0 (pan2
         (ifft*
          (pv-conformal-map
           (fft* 10 (mul (sound-in 0) 0.5)) (mouse-
x kr -1 1 0 0.1)
           (mouse-y kr -1 1 0 0.1)))
         0
         1)))


(let* ((signal (lambda (n)
                 (let* ((o (sin-osc kr (mix-
fill n (lambda (_) (rand 0.1 0.5))) 0))
                        (a (mul (make-mce (list 1 1.1 1.5 1.78 2.45 6.7)) 220))
                        (f (mul-add o 10 a)))
                   (mix (mul (lf-saw ar f 0) 0.3)))))
       (mapped (lambda (n)
                 (let* ((c0 (fft* 0 (signal n)))
                        (x (mouse-x kr 0.01 2.0  1.0 0.1))
                        (y (mouse-y kr 0.01 10.0 1.0 0.1))
                        (c1 (pv-conformal-map c0 x y)))
                   (ifft* c1))))
       (s (mapped 3))
       (t (mul-add (comb-n s 0.1 0.1 10) 0.5 s)))
  (audition (out 0 (pan2 t 0 1))))
```

4.266 (pv-diffuser buffer trig)

adds a different constant random phase shift to each bin. The trigger will select a new set of random phases.

buffer - fft buffer. trig - a trigger selects a new set of random values.

```
(with-sc3
```

```
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (h (pv-diffuser f (gt (mouse-x kr 0 1 0 0.1) 0.5))))
  (audition (out 0 (mul (ifft* h) 0.5))))
```

4.267 (pv-max bufferA bufferB)

4.268 (Ifft buffer wintype)

(ifft* b) => (ifft b 0)

inverse Fast Fourier Transform. The inverse fast fourier transform converts from frequency content to a signal.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 2048 1))))


(let* ((s (mul (white-noise ar) 0.05))
       (c (fft 0 s 0.5 0 1)))
  (audition (out 0 (Ifft c 0))))
```

4.269 (pv-bin-scramble buffer wipe width trig)

randomizes the order of the bins. The trigger will select a new random ordering.

buffer - fft buffer. wipe - scrambles more bins as wipe moves from zero to one. width - a value from zero to one, indicating the maximum randomized distance of a bin from its original location in the spectrum. trig - a trigger selects a new random ordering.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 1 0 1))
       (f (fft* 10 a))
```

```
       (g (pv-bin-scramble f
                           (mouse-x kr 0.0 1.0 0 0.1)
                           (mouse-y kr 0.0 1.0 0 0.1)
                           (impulse kr 4 0)))
       (h (ifft* g)))
  (audition (out 0 (mul 0.5 (mce2 h h)))))
```

careful - reads adc!

```
(let* ((a (mul (sound-in (mce2 0 1)) 4.0))
       (f (fft* 10 a))
       (g (pv-bin-scramble f
                           (mouse-x kr 0.25 1.0 0 0.1)
                           (mouse-y kr 0.25 1.0 0 0.1)
                           (impulse kr (mul-add (lf-
noise0 kr 2) 8 10) 0)))
       (h (ifft* g)))
  (audition (out 0 (pan2 h 0 0.5))))
```

4.270 (pv-rand-wipe bufferA bufferB wipe trig)

Cross fades between two sounds by copying bins in a random order.

bufferA = fft buffer A. bufferB = fft buffer B. wipe = copies bins from bufferB in a random order (0, 1). trig = select new random ordering.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc 11 2048 1))))


(define-syntax n-of
  (syntax-rules ()
    ((_ n f) (mix-fill n (lambda (_) f)))))


(let* ((n 6)
       (a (n-of n (mul (lf-saw ar (exp-rand 400.0 1000.0) 0.0) 0.1)))
       (b (n-of n (mul (lf-pulse ar (exp-rand 80.0 400.0) 0.0 0.2)
                       (u:max (mul (sin-osc kr (rand 0.0 8.0) 0.0) 0.2) 0.0))))
       (f (fft* 10 a))
       (g (fft* 11 b))
       (y (mouse-y kr 0 1 0 0.1))
       (x (mouse-x kr 0 1 0 0.1))
```

```
      (h (pv-rand-wipe f g x (gt y 0.5)))
      (i (ifft* h)))
  (audition (out 0 (mul 0.5 (mce2 i i)))))
```

4.271 (Packfft chain bufsize frombin tobin zeroothers magsphases)

Pack separate demand-rate fft bin streams into an fft chain buffer

Takes a length-prefixed array of magnitudes and phases, and packs them into an fft buffer ready for transforming back into time-domain audio using Ifft.

Most people won't need to use this directly - instead, use pvcollect, pvcalc, or pvcalc2.

The input data is magsphases, which should be a flat array containing magnitude and phase of all bins in ascending order. e.g. [mag0, phase0, mag1, phase1, mag2, phase2, ... magN, phaseN] This input is typically demand-rate.

This is technically similar to demand or duty in that it calls demand-rate UGens further up the graph to process the values, eventually calling Unpackfft. These two ends of the process must in most cases see the same chain...! Otherwise behaviour is undefined and, who knows, possibly unpleasant.

frombin and tobin allow you to fill the supplied data only into a subset of the fft bins (i.e. a single delimited frequency band), set zeroothers to 1 to zero all the magnitudes outside this band (otherwise they stay intact).

For usage examples, see Unpackfft, but also pvcollect, pvcalc, pvcalc2.

Here's an unusual example which uses Packfft without using Unpackfft first - essentially creating our fft data from scratch.

```
(with-sc3
 (lambda (fd)
   (send fd (b-alloc 10 512 1))))


(let* ((n 100)
       (n* (enum-from-to 1 n))
       (m1 (map (lambda (_) (range (f-sin-osc kr (exp-
rand 0.1 1) 0) 0 1)) n*))
       (square (lambda (a) (* a a)))
       (m2 (map mul m1 (map square (iota n 1.0 (- (/ 1.0 n))))))
       (i (map (lambda (_) (lf-pulse kr (pow 2 (i-
rand -3 5)) 0 0.3)) n*))
       (m3 (map mul m2 i))
       (p (replicate n 0.0))
```

```
      (c1 (fft* 10 (f-sin-osc ar 440 0)))
      (c2 (pack-fft c1 512 0 (- n 1) 1 (packfft-data m3 p)))
      (s (ifft* c2)))
  (audition (out 0 (mce2 s s))))
```

4.272 (pvcollect chain numframes func frombin tobin zeroothers)

Process each bin of an fft chain separately.

pvcollect applies function func to each bin of an fft chain. func should be a function that takes magnitude, phase, index as inputs and returns a resulting [magnitude, phase].

The "index" is the integer bin number, starting at 0 for DC. You can optionally ignore the phase and only return a single (magnitude) value, in which case the phase is assumed to be left unchanged.

frombin, tobin, and zeroothers are optional arguments which limit the processing to a specified integer range of fft bins. If zeroothers is set to 1 then bins outside of the range being processed are silenced.

Note that this procedure can be relatively CPU-heavy, depending on how you use it.

```
(define no-op
  (lambda (m p _)
    (list m p)))


(define rand-phase
  (lambda (m p _)
    (list m (rand 0 3.14))))


(define noise-phase
  (lambda (m p _)
    (list m (lin-lin (lf-noise0 kr 3) -1 1 0 3.14))))


(define combf
  (lambda (m p i)
    (list (if (= (modulo i 7) 0) m 0) p)))


(define noise-mag
  (lambda (m p _)
    (list (mul (gt (lf-noise0 kr 10) 0) m) p)))
```

```
(define spectral-delay
  (lambda (m p _)
    (let ((v (lin-lin (lf-par kr 0.5 0) -1 1 0.1 1)))
      (list (add m (delay-n m 1 v)) p))))

(define (bpf-sweep nf)
  (lambda (m p i)
    (let ((e (u:abs (sub i (lin-lin (lf-par kr 0.1 0) -1 1 2 (/ nf 20))))))
      (list (mul (lt e 10) m) p))))

(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 1024 1))
   (async fd (b-alloc-read 11 "/home/rohan/audio/metal.wav" 0 0))))

(let* ((nf 1024)
       (i (play-buf 1 11 (buf-rate-scale kr 11) 1 0 1))
       (c1 (fft* 10 i))
       (c2 (pvcollect c1 nf spectral-delay 0 250 0)))
  (audition (out 0 (mul 0.1 (ifft* c2)))))
```

4.273 (PV_Magclip buffer threshold)

clip bins to a threshold. clips bin magnitudes to a maximum threshold.

buffer - fft buffer. threshold - magnitude threshold.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))

(let* ((a (play-buf 1 12 (buf-rate-scale kr 12) 0 0 1))
       (f (fft* 10 a))
       (h (pv-mag-clip f (mouse-x kr 0 5 0 0.1))))
  (audition (out 0 (mul (ifft* h) 0.5))))
```

4.274 (PV_Magfreeze buffer freeze)

freeze magnitudes. freezes magnitudes at current levels when freeze > 0.

buffer - fft buffer. freeze - if > 0 then magnitudes are frozen at current levels.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))
   (async fd (b-alloc-read 12 "/home/rohan/audio/metal.wav" 0 0))))


(let ((dup (lambda (a) (mce2 a a)))
      (s (sin-osc ar (mul-add (lf-noise1 kr 5.2) 250 400) 0))
      (f (sin-osc kr 0.2 0)))
  (audition (out 0 (dup (mul 0.25 (ifft* (pv-mag-
freeze (fft* 10 s) f)))))))


(let ((dup (lambda (a) (mce2 a a)))
      (s (play-buf 1 12 (buf-rate-scale kr 12) 1 0 1))
      (f (gt (mouse-y kr 0 1 0 0.1) 0.5)))
  (audition (out 0 (dup (mul 0.25 (ifft* (pv-mag-
freeze (fft* 10 s) f)))))))
```

4.275 (pv-rect-comb buffer numTeeth phase width)

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 2048 1))))


(let* ((dup (lambda (a) (mce2 a a)))
       (x (mouse-x kr 0 0.5 0 0.1))
       (y (mouse-y kr 0 0.5 0 0.1))
       (n (dup (mul (white-noise ar) 0.3)))
       (c (pv-rect-comb (fft* 10 n) 8 x y)))
  (audition (out 0 (ifft* c))))


(let* ((dup (lambda (a) (mce2 a a)))
       (p (mul-add (lf-tri kr 0.097 0) 0.4 0.5))
       (w (mul-add (lf-tri kr 0.24 0) -0.5 0.5))
       (n (dup (mul (white-noise ar) 0.3)))
       (c (pv-rect-comb (fft* 10 n) 8 p w)))
  (audition (out 0 (ifft* c))))
```

4.276 (pv-mag-shift buffer stretch shift)

4.277 See allpass-n

4.278 See comb-n

4.279 See Bufallpass-c

4.280 (free-verb in mix room damp)

(free-verb2 in1 in2 mix room damp)

A simple reverb.

in, in1, in2 - input signal mix - dry/wet balance (0,1) room - room size (0,1) damp - reverb high frequency damping (0,1)

```
(let* ((i (impulse ar 1 0))
       (c (lf-cub ar 1200 0))
       (s (mul3 (decay i 0.25) c 0.1))
       (x (mouse-x kr 0 1 0 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (r (free-verb s y x 0.5)))
  (audition (out 0 r)))

(let* ((i (sound-in (mce2 0 1)))
       (c (lambda (u n) (mce-channel u n)))
       (x (mouse-x kr 0 1 0 0.1))
       (y (mouse-y kr 0 1 0 0.1))
       (r (free-verb2 (c i 0) (c i 1) y x 0.5)))
  (audition (out 0 r)))
```

4.281 See freeVerb

4.282 (play-buf numChannels bufnum rate trigger startPos loop)

Sample playback oscillator. Plays back a memory resident sample.

numChannels - number of channels that the buffer will be. This must be a fixed integer. The architechture of the SynthDef cannot change after it is compiled. Warning: if you supply a bufnum of a buffer that has a different numChannels then you have specified to the play-buf, it will fail silently.

bufnum - the index of the buffer to use

rate - 1.0 is the server's sample rate, 2.0 is one octave up, 0.5 is one octave down -1.0 is backwards normal rate etc. interpolation is cubic. Note: If the buffer's sample rate is different from the server's, you will need to multiply the desired playback rate by (file's rate / server's rate). The UGen buf-rate-scale.kr(bufnum) returns this factor. See examples below. buf-rate-scale should be used in virtually every case.

trigger - a trigger causes a jump to the startPos. A trigger occurs when a signal changes from <= 0 to > 0.

startPos - sample frame to start playback.

loop - 1 means true, 0 means false. This is modulate-able.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 10 "/home/rohan/audio/metal.wav" 0 0))))
```

Play once only.

```
(audition (out 0 (play-buf 1 10 (buf-rate-scale kr 10) 1 0 0)))
```

Play in infinite loop.

```
(audition (out 0 (play-buf 1 10 (buf-rate-scale kr 10) 1 0 1)))
```

trigger playback at each pulse.

```
(audition (out 0 (play-buf 1 10 (buf-rate-
scale kr 10) (impulse kr 2 0) 0 0)))
```

trigger playback at each pulse (diminishing intervals).

```
(let ((t (impulse kr (x-line kr 0.1 100 10 remove-synth) 0)))
  (audition (out 0 (play-buf 1 10 (buf-rate-scale kr 10) t 0 0))))
```

Loop playback, accelerating pitch.

```
(let ((rate (x-line kr 0.1 100 60 remove-synth)))
  (audition (out 0 (play-buf 1 10 rate 1 0 1))))
```

Sine wave control of playback rate, negative rate plays backwards.

```
(let ((r (mul-add (f-sin-osc kr (x-line kr 0.2 8 30 remove-
synth) 0) 3 0.6)))
  (audition (out 0 (play-buf 1 10 (mul (buf-rate-
scale kr 10) r) 1 0 1))))
```

Release buffer.

```
(with-sc3
 (lambda (fd)
   (async fd (b-free 10))))
```

4.283 See buf-delay-c

4.284 (delay2 in)

Fixed two sample delay.

```
(let ((s (impulse ar 1 0)))
  (audition
   (out 0 (add s (delay2 s)))))
```

4.285 (comb-n in maxDelayTime delayTime decayTime)

(comb-l in maxDelayTime delayTime decayTime) (comb-c in maxDelayTime delayTime decayTime)

Comb delay line. comb-n uses no interpolation, comb-l uses linear interpolation, comb-c uses all pass interpolation. All times are in seconds. The decay time is the time for the echoes to decay by 60 decibels. If this time is negative then the feedback coefficient will be negative, thus emphasizing only odd harmonics at an octave lower.

Comb used as a resonator. The resonant fundamental is equal to reciprocal of the delay time.

```
(define src (mul (white-noise ar) 0.01))


(define ctl (x-line kr 0.0001 0.01 20 remove-synth))


(define hear (lambda (u) (audition (out 0 u))))


(hear (comb-n src 0.01 ctl 0.2))
```

```
(hear (comb-l src 0.01 ctl 0.2))

(hear (comb-c src 0.01 ctl 0.2))
```

With negative feedback:

```
(hear (comb-n src 0.01 ctl -0.2))

(hear (comb-l src 0.01 ctl -0.2))

(hear (comb-c src 0.01 ctl -0.2))
```

Used as an echo.

```
(hear (comb-n (mul (decay (mul (dust ar 1) 0.5) 0.2) (white-
noise ar))
       0.2 0.2 3))
```

4.286 See Bufallpass-c

4.287 (allpass-n in maxDelayTime delayTime decayTime)

All pass delay line. allpass-n uses no interpolation, allpass-l uses linear interpolation, allpass-c uses all pass interpolation. All time values are in seconds. The decay time is the time for the echoes to decay by 60 decibels. If this time is negative then the feedback coefficient will be negative, thus emphasizing only odd harmonics at an octave lower.

Since the allpass delay has no audible effect as a resonator on steady state sound ...

```
(define z (mul (white-noise ar) 0.1))

(audition (out 0 (allpass-c z 0.01 (x-line kr 0.0001 0.01 20 do-
nothing) 0.2)))
```

...these examples add the input to the effected sound so that you can hear the effect of the phase comb.

```
(audition
 (out 0 (add z (allpass-n z 0.01 (x-line kr 0.0001 0.01 20 do-
nothing) 0.2))))
```

```
(audition
 (out 0 (add z (allpass-l z 0.01 (x-line kr 0.0001 0.01 20 do-
nothing) 0.2))))


(audition
 (out 0 (add z (allpass-c z 0.01 (x-line kr 0.0001 0.01 20 do-
nothing) 0.2))))
```

Used as an echo - doesn't really sound different than Comb, but it outputs the input signal immediately (inverted) and the echoes are lower in amplitude.

```
(audition
 (out 0 (allpass-n (mul (decay (dust ar 1) 0.2) z) 0.2 0.2 3)))
```

4.288 See comb-n

4.289 (buf-allpass-c buf in delaytime decaytime)

Buffer based all pass delay line with cubic interpolation

All pass delay line with cubic interpolation which uses a buffer for its internal memory. See also Bufallpass-n which uses no interpolation, and Bufallpass-l which uses linear interpolation. Cubic interpolation is more computationally expensive than linear, but more accurate.

See also allpass-c.

buf - buffer number.

in - the input signal.

delaytime - delay time in seconds.

decaytime - time for the echoes to decay by 60 decibels. If this time is negative then the feedback coefficient will be negative, thus emphasizing only odd harmonics at an octave lower.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 44100 1))))


(let ((x (mul3 (decay (dust ar 1) 0.2) (white-noise ar) 0.5)))
  (audition (out 0 (buf-allpass-n 0 x 0.25 6))))
```

4.290 See allpass-n

4.291 See buf-delay-c

4.292 (buf-delay-c buf in delaytime)

Buffer based simple delay line with cubic interpolation.

Simple delay line with cubic interpolation which uses a buffer for its internal memory. See also buf-delay-n which uses no interpolation, and buf-delay-l which uses linear interpolation. Cubic interpolation is more computationally expensive than linear, but more accurate.

See also delay-c.

buf - buffer number. in - the input signal. delaytime - delay time in seconds.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 44100 1))))


(let ((z (mul3 (decay (dust ar 1) 0.5) 0.3 (white-noise ar))))
  (audition (out 0 (add (buf-delay-c 0 z 0.2) z))))
```

4.293 (pluck in tr maxdelaytime delaytime decaytime coef)

Karplus-Strong synthesis.

in - an excitation signal

tr - upon a negative to positive transition, the excitation signal will be fed into the delay line

maxdelaytime - the max delay time in seconds (initializes the internal delay buffer).

delaytime - delay time in seconds.

decaytime - time for the echoes to decay by 60 decibels. Negative times emphasize odd partials.

coef - the coef of the internal one-pole filter. Values should be between -1 and +1 (larger values will be unstable... so be careful!).

Excitation signal is white-noise, triggered twice a second with varying one-pole coef.

```
(let ((n (mul (white-noise ar) 0.1))
```

```
      (t (impulse kr 2 0))
      (x (mouse-x kr -0.999 0.999 0 0.1))
      (dl (/ 1 440)))
  (audition (out 0 (pluck n t dl dl 10 x))))


(let* ((n 25)
       (gen (lambda (n f) (mce-fill n (lambda (_) (f)))))
       (f (gen n (lambda () (rand 0.05 0.2))))
       (p (gen n (lambda () (rand 0 1))))
       (x (mouse-x kr 60 1000 1 0.1))
       (o (lin-lin (sin-osc kr f p) -1 1 x 3000))
       (w (clone n (mul (white-noise ar) 0.1)))
       (i (impulse kr (gen n (lambda () (rand 10 12))) 0))
       (ks (pluck w i 0.01 (fdiv 1 o) 2 (rand 0.01 0.2)))
       (l (gen n (lambda () (rand -1 1)))))
  (audition (out 0 (leak-dc (mix (pan2 ks l 1)) 0.995))))
```

4.294 (pitch-shift in winSize pchRatio pchDispersion timeDispersion)

A simple time domain pitch shifter.

```
(audition
 (out 0 (pitch-shift (sin-osc ar 440 0)
                     0.2
                     (mouse-x kr 0.5 2 0 0.1)
                     (mouse-y kr 0 0.1 0 0.1)
                     0)))
```

4.295 See buf-comb-c

4.296 (buf-comb-c buf in delaytime decaytime)

Buffer based comb delay line with cubic interpolation

All pass delay line with cubic interpolation which uses a buffer for its internal memory. See also buf-comb-n which uses no interpolation, and buf-comb-l which uses linear interpolation. Cubic interpolation is more computationally expensive than linear, but more accurate. See also comb-c.

buf - buffer number.

in - the input signal.

delaytime - delay time in seconds.

decaytime - time for the echoes to decay by 60 decibels. If this time is negative then the feedback coefficient will be negative, thus emphasizing only odd harmonics at an octave lower.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 0 44100 1))))


(let ((x (mul3 (decay (dust ar 1) 0.2) (white-noise ar) 0.5)))
  (audition (out 0 (buf-comb-n 0 x 0.25 6))))
```

4.297 (buf-rd numChannels rate bufnum phase loop interpolation)

Plays the content of a buffer.

The number of channels must be a fixed integer. The architechture of the SynthDef cannot change after it is compiled. NOTE: if you supply a bufnum of a buffer that has a different numChannels then you have specified to the buf-rd, it will fail silently.

The interpolation type is an integer: 1 no interpolation, 2 linear interpolation, 4 cubic interpolation.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc-read 0 "/home/rohan/audio/metal.wav" 0 0))))


(audition (out 0 (buf-rd 1 ar 0 (mul (sin-osc ar 0.1 0) (buf-
frames ir 0)) 0 2)))


(let ((phase (mul (lf-noise1 ar (mouse-x kr (mce2 5 10) 100 0 0.1))
                  (buf-frames ir 0))))
  (audition (out 0 (buf-rd-l 1 ar 0 phase 0))))


(let ((phase (add (lf-tri ar 0.1 0)
                  (mul (lf-tri ar 0.23 0) (buf-frames ir 0)))))
  (audition (out 0 (buf-rd-l 1 ar 0 phase 0))))
```

Use a phasor index into the file

```
(let ((phase (phasor ar
```

```
                    0
                    (mul (mouse-x kr 0.5 2 0 0.1)
                         (buf-rate-scale kr 0))
                    0
                    (buf-frames kr 0)
                    0)))
  (audition (out 0 (buf-rd 1 ar 0 phase 1 (mouse-
y kr 0 5 0 0.1)))))
```

4.298 (delay1 in)

Fixed Single sample delay.

```
(let ((s (impulse ar 1 0)))
  (audition
   (out 0 (add s (delay1 s)))))
```

4.299 (record-buf bufnum offset reclevel prelevel run loop trigger inputs)

Records input into a Buffer.

If recLevel is 1.0 and preLevel is 0.0 then the new input overwrites the old data. If they are both 1.0 then the new data is added to the existing data. (Any other settings are also valid.)

bufnum - the index of the buffer to use offset - an offset into the buffer in samples, default 0 recLevel - value to multiply by input before mixing with existing data. Default is 1.0. preLevel - value to multiply to existing data in buffer before mixing with input. Default is 0.0. run - If zero, then recording stops, otherwise recording proceeds. Default is 1. loop - If zero then don't loop, otherwise do. This is modulate-able. Default is 1. trigger - a trigger causes a jump to the start of the Buffer. A trigger occurs when a signal changes from <=0 to >0. inputArray - an Array of input channels

```
(with-sc3
 (lambda (fd)
   (send-synth
    fd
    "recorder"
    (letc ((in 0)
           (bufnum 0)
           (offset 1)
           (recLevel 1)
           (preLevel 0)
```

```
            (run 1)
            (loop 1)
            (trigger 1))
       (let ((i (in 2 ar in)))
         (out 0 (record-buf bufnum offset recLevel preLevel run loop trigger i)))))
   (let ((b 10)
         (y 1001)
         (z 1002))
     (async fd (/b_alloc b 44100 2))
     (send fd (/s_new "recorder" y add-to-
tail 1 "bufnum" b "in" 8))
     (send fd (/n_trace y))
     (send-synth
      fd
      "player"
      (letc ((bufnum 0)
             (rate 1)
             (trigger 1)
             (startPos 0)
             (loop 1)
             (gain 1))
        (out 0 (mul (play-buf 2 bufnum rate trigger startPos loop) gain))))
     (send fd (/s_new "player" z add-to-tail 1 "bufnum" b)))))

(define do-send
  (lambda (m)
    (with-sc3
     (lambda (fd)
       (send fd m)))))

(do-send (/n_set 1001 "run" 1))

(do-send (/n_set 1002 "loop" 1))

(do-send (/n_set 1002 "gain" 2))

(do-send (/n_set 1002 "trigger" 1))

(do-send (/n_free 1001))

(do-send (/n_free 1002))

(with-sc3
 (lambda (fd)
   (async fd (/b_free 10))))
```

4.300 See delay-n

4.301 See buf-comb-c

4.302 See delay-n

4.303 (delay-n in maxDelayTime delayTime)

(delay-l in maxDelayTime delayTime) (delay-c in maxDelayTime delayTime)

Simple delay line. There are three forms, delay-n uses no interpolation, delay-l uses linear interpolation, delay-c uses cubic interpolation. The maximum delay length is set at initialization time and cannot be extended.

dust randomly triggers decay to create an exponential decay envelope for the white-noise input source.

```
(let ((z (mul (decay (dust ar 1) 0.3)
              (white-noise ar))))
  (audition
   (out 0 (add (delay-n z 0.2 0.2) z))))
```

4.304 (ball in g damp friction)

Physical model of bouncing object.

Models the path of a bouncing object that is reflected by a vibrating surface.

in - modulated surface level g - gravity damp - damping on impact friction - proximity from which on attraction to surface starts

4.305 (dswitch1 index array)

demand rate generator for switching between inputs

index - which of the inputs to return array - array of values or other ugens

```
(let* ((x (mouse-x kr 0 4 0 0.1))
       (y (mouse-y kr 1 15 0 0.1))
       (a (dswitch1 x (make-mce (list 1 3 y 2 (dwhite 2 0 3)))))
       (t (impulse kr 3 0))
       (f (mul-add (demand t 0 a) 30 340)))
```

```
(audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.306 (t-duty rate duration reset doneAction level gap)

demand results as trigger from demand rate ugens.

A value is demanded each ugen in the list and output as a trigger according to a stream of duration values. The unit generators in the list should be 'demand' rate. When there is a trigger at the reset input, the demand rate ugens in the list and the duration are reset. The reset input may also be a demand ugen, providing a stream of reset times.

NOTE: sclang reorders the inputs to be 'duration reset level doneAction', rsc does not.

duration - time values. Can be a demand ugen or any signal. The next trigger value is acquired after the duration provided by the last time value.

reset - trigger or reset time values. Resets the list of ugens and the duration ugen when triggered. The reset input may also be a demand ugen, providing a stream of reset times.

doneAction - a doneAction that is evaluated when the duration stream ends.

level - demand ugen providing the output values.

Play a little rhythm

```
(let ((s (dseq dinf (make-mce (list 0.1 0.2 0.4 0.3)))))
  (audition (out 0 (t-duty ar s 0 0 1 0))))
```

Amplitude changes

```
(let ((t (t-duty ar
                 (dseq dinf (make-mce (list 0.1 0.2 0.4 0.3)))
                 0
                 0
                 (dseq dinf (make-mce (list 0.1 0.4 0.01 0.5 1.0)))
                 0)))
  (audition (out 0 (ringz t 1000 0.1))))


(let ((t (t-duty ar
                 (mouse-x kr 0.001 2 1 0.1)
                 0
                 0
                 (dseq dinf (make-mce (list 0.1 0.4 0.01 0.5 1.0)))
                 0)))
  (audition (out 0 (ringz t 1000 0.1))))
```

4.307 See dwhite

4.308 (dwhite length lo hi)

(diwhite length lo hi)

demand rate white noise random generators.

length number of values to create lo minimum value hi maximum value

dwhite returns numbers in the continuous range between lo and hi, diwhite returns integer values. The arguments can be a number or any other ugen

```
(let* ((a (dwhite dinf 0 15))
       (t (impulse kr (mouse-x kr 1 40 1 0.1) 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))

(let* ((a (diwhite dinf 0 15))
       (t (impulse kr (mouse-x kr 1 40 1 0.1) 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.309 (dbufrd bufnum phase loop)

Buffer demand ugen.

bufnum - buffer number to read from phase - index into the buffer (demand ugen or any other ugen) loop - loop when phase exceeds number of frames in buffer

Example

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 10 24 1))
   (send fd (b-setn1 10 0 (replicate-m 24 (exp-
random 200 500))))))

(let* ((q (dseq 3 (make-mce (list 0 3 5 0 3 7 0 5 9))))
       (p (dseq dinf (mce2 q (dbrown 5 0 23 1))))
       (t (dust kr 10)))
  (audition (out 0 (mul (sin-osc ar (demand t 0 (dbufrd 10 p 1)) 0) 0.1))))
```

Buffer as a time pattern.

```
(with-sc3
 (lambda (fd)
   (async fd (b-alloc 11 24 1))
   (send fd (b-setn1 11 0 (replicate-m 24 (choose (list 1 0.5 0.25)))))))

(let* ((p (dseq dinf (mce2 (dseq 3 (make-mce (list 0 3 5 0 3 7 0 5 9)))
                           (dbrown 5 0 23 1))))
       (d (mul (dbufrd 11 (dseries dinf 0 1) 1) 0.5))
       (l (dbufrd 10 p 1)))
  (audition (out 0 (mul (sin-osc ar (duty kr d 0 do-
nothing l) 0) 0.1))))
```

free buffers

```
(with-sc3
 (lambda (fd)
   (async fd (b-free 10))
   (async fd (b-free 11))))
```

4.310 (demand-env-gen rate levels times shapes curves gate reset

levelScale levelOffset timeScale doneAction)

levels - a demand ugen or any other ugen

times - a demand ugen or any other ugen if one of these ends, the doneAction is evaluated

shapes - a demand ugen or any other ugen, the number given is the shape number according to Env

curves - a demand ugen or any other ugen, if shape is 5, this is the curve factor some curves/shapes don't work if the duration is too short. have to see how to improve this. also some depend on the levels obviously, like exponential cannot cross zero.

gate - if gate is x >= 1, the ugen runs, if gate is 0 > x > 1, the ugen is released at the next level (doneAction), if gate is x < 0, the ugen is sampled end held

reset - if reset crosses from nonpositive to positive, the ugen is reset at the next level, if it is > 1, it is reset immediately.

Frequency envelope with random times.

```
(let* ((l (dseq dinf (make-mce (list 204 400 201 502 300 200))))
       (t (drand dinf (make-mce (list 1.01 0.2 0.1 2.0))))
       (y (mouse-y kr 0.01 3 1 0.1))
```

```
       (f (demand-env-gen ar l (mul t y) 7 0 1 1 1 0 1 do-
nothing)))
  (audition (out 0 (mul (sin-osc ar (mul f (mce2 1 1.01)) 0) 0.1))))
```

Frequency modulation

```
(let* ((x (mouse-x kr -0.01 -4 0 0.1))
       (y (mouse-y kr 1 3000 1 0.1))
       (l (lambda () (dseq dinf (clone 32 (exp-rand 200 1000)))))
       (t (mul sample-dur y))
       (f (demand-env-gen ar (mce2 (l) (l)) t 5 x 1 1 1 0 1 do-
nothing)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

gate. Mouse x on right side of screen toggles gate.

```
(let* ((x (mouse-x kr 0 1 0 0.1))
       (l (u:round (dwhite dinf 300 1000) 100))
       (f (demand-env-gen kr l 0.1 5 0.3 (gt x 0.5) 1 1 0 1 do-
nothing)))
  (audition (out 0 (mul (sin-osc ar (mul f (mce2 1 1.21)) 0) 0.1))))
```

4.311 (demand trig reset ugens)

demand results from demand rate ugens.

When there is a trigger at the trig input, a value is demanded from each ugen in the list and output. The unit generators in the list should be 'demand' rate.

When there is a trigger at the reset input, the demand rate ugens in the list are reset.

trig - trigger. trigger can be any signal. A trigger happens when the signal changes from non-positive to positive.

reset - trigger. Resets the list of ugens when triggered.

```
(define (mirror1 l)
  (append l (cdr (reverse (cdr l)))))


(let* ((t (impulse kr 24 0))
       (s (drand dinf (mce2 (dseq 1 (make-mce (mirror1 (enum-from-
to 1 5))))
                            (drand 8 (make-mce (enum-from-
to 4 11))))))
```

```
      (f (demand t 0 (mul s 100)))
      (x (mouse-x kr -1 1 0 0.1))
      (o (sin-osc ar (mce2 f (add f 0.7)) 0)))
  (audition (out 0 (mul (scale-neg (cubed (cubed o)) x) 0.1))))


(let* ((t (impulse kr 10 0))
       (r (dust kr 1))
       (s (dgeom dinf (midi-cps 72) (midi-ratio 1)))
       (f (demand t r s))
       (o (sin-osc ar (mce2 f (add f 0.7)) 0)))
  (audition (out 0 (mul (u:max (cubed o) 0) 0.1))))


(let* ((t (impulse kr 10 0))
       (s (midi-cps (diwhite dinf 60 72)))
       (f (demand t 0 s))
       (o (sin-osc ar (mce2 f (add f 0.7)) 0)))
  (audition (out 0 (mul (cubed (cubed o)) 0.1))))
```

4.312 (duty rate duration reset doneAction level)

demand results from demand rate ugens

A value is demanded from each ugen in the list and output according to a stream of duration values. The unit generators in the list should be 'demand' rate. When there is a trigger at the reset input, the demand rate ugens in the list and the duration are reset. The reset input may also be a demand ugen, providing a stream of reset times.

duration: time values. Can be a demand ugen or any signal. The next value is acquired after the duration provided by the last time value.

reset: trigger or reset time values. Resets the list of ugens and the duration ugen when triggered. The reset input may also be a demand ugen, providing a stream of reset times.

doneAction: a doneAction that is evaluated when the duration stream ends.

level: demand ugen providing the output values.

```
(let* ((f (duty kr
                (drand dinf (mce3 0.01 0.2 0.4))
                0
                2
                (dseq dinf (make-mce (list 204 400 201 502 300 200)))))
       (o (sin-osc ar (mul f (mce2 1 1.01)) 0)))
  (audition (out 0 (mul o 0.1))))
```

```
(let* ((f (duty kr
                (mouse-x kr 0.001 2 1 0.1)
                0
                2
                (dseq dinf (make-mce (list 204 400 201 502 300 200)))))
       (o (sin-osc ar (mul f (mce2 1 1.0)) 0)))
  (audition (out 0 (mul o 0.1))))
```

4.313 (dser length array)

demand rate sequence generator.

array - array of values or other ugens length - number of values to return

```
(let* ((a (dser dinf (make-mce (list 1 3 2 7 8))))
       (x (mouse-x kr 1 40 1 0.1))
       (t (impulse kr x 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.314 (dgeom length start grow)

demand rate geometric series ugen.

start - start value grow - value by which to grow (x = x[-1] * grow) length - number of values to create

The arguments can be a number or any other ugen

```
(let* ((a (dgeom 15 1 1.2))
       (t (impulse kr (mouse-x kr 1 40 1 0.1) 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

demand rate UGens are not shared...

```
(let* ((a  (dgeom 15 1 1.2))
       (t  (impulse ar (mouse-x kr 1 40 1 0.1) 0))
       (f0 (mul-add (demand (delay1 t) 0 a) 30 340))
       (f1 (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar (mce2 f0 f1) 0) 0.1))))
```

```
(let* ((a0 (dgeom 15 1 1.2))
       (a1 (dgeom 15 1 1.2))
       (t  (impulse ar (mouse-x kr 1 40 1 0.1) 0))
       (f0 (mul-add (demand (delay1 t) 0 a0) 30 340))
       (f1 (mul-add (demand t 0 a1) 30 340)))
  (audition (out 0 (mul (sin-osc ar (mce2 f0 f1) 0) 0.1))))
```

4.315 (drand length array)

(dxrand length array)

demand rate random sequence generators.

length - number of values to return array - array of values or other ugens

dxrand never plays the same value twice, whereas drand chooses any value in the list.

```
(let ((f (lambda (u)
           (let* ((a (u dinf (make-mce (list 1 3 2 7 8))))
                  (t (impulse kr (mouse-x kr 1 400 1 0.1) 0))
                  (f (mul-add (demand t 0 a) 30 340)))
             (mul (sin-osc ar f 0) 0.1)))))
  (audition (out 0 (mce2 (f drand)
                         (f dxrand)))))
```

4.316 See drand

4.317 (dseries length start step)

demand rate arithmetic series ugen.

length - number of values to create start - start value step - step value

The arguments can be a number or any other ugen

```
(let* ((a (dseries 15 0 1))
       (t (impulse kr (mouse-x kr 1 40 1 0.1) 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.318 (dswitch index array)

demand rate generator for switching between inputs

index - which of the inputs to return array - array of values or other ugens

In difference to dswitch1, dswitch embeds all items of an input demand ugen first before looking up the next index.

```
(let* ((a0 (dwhite 2 3 4))
       (a1 (dwhite 2 0 1))
       (a2 (dseq 2 (make-mce (list 1 1 1 0))))
       (i (dseq 2 (make-mce (list 0 1 2 1 0))))
       (d (dswitch i (make-mce (list a0 a1 a2))))
       (t (impulse kr 4 0))
       (f (mul-add (demand t 0 d) 300 400))
       (o (mul (sin-osc ar f 0) 0.1)))
  (audition (out 0 o)))
```

compare with dswitch1

```
(let* ((a0 (dwhite 2 3 4))
       (a1 (dwhite 2 0 1))
       (a2 (dseq 2 (make-mce (list 1 1 1 0))))
       (i (dseq 2 (make-mce (list 0 1 2 1 0))))
       (d (dswitch1 i (make-mce (list a0 a1 a2))))
       (t (impulse kr 4 0))
       (f (mul-add (demand t 0 d) 300 400))
       (o (mul (sin-osc ar f 0) 0.1)))
  (audition (out 0 o)))
```

4.319 See dbrown

4.320 (dbrown length lo hi step)

(dibrown length lo hi step)

demand rate brownian movement generators.

lo - minimum value hi - maximum value step - maximum step for each new value length - number of values to create

dbrown returns numbers in the continuous range between lo and hi, dibrown returns integer values. The arguments can be a number or any other ugen.

```
(let ((f (lambda (u)
           (let* ((a (u dinf 0 15 1))
                  (t (impulse kr (mouse-x kr 1 40 1 0.1) 0))
                  (f (mul-add (demand t 0 a) 30 340)))
             (mul (sin-osc ar f 0) 0.1)))))
  (audition (out 0 (mce2 (f dbrown) (f dibrown)))))
```

4.321 (dseq length array)

demand rate sequence generator.

array - array of values or other ugens length - number of repeats

```
(let* ((a (dseq 3 (make-mce (list 1 3 2 7 8))))
       (t (impulse kr (mouse-x kr 1 40 1 0.1) 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

```
(let* ((a (dseq dinf (make-mce (replicate-m 32 (random 0 10)))))
       (t (impulse ar (mouse-x kr 1 10000 1 0.1) 0))
       (f (mul-add (demand t 0 a) 30 340)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.322 See latoocarfian-c.

4.323 (fb-sine-c rate freq im fb a c xi yi)

Feedback sine with chaotic phase indexing.

freq - iteration frequency in Hz - 22050 im - index multiplier amount - 1 fb - feedback amount - 0.1 a - phase multiplier amount - 1.1 c - phase increment amount - 0.5 xi - initial value of x - 0.1 yi - initial value of y - 0.1

A cubic-interpolating sound generator based on the difference equations:

xn+1 = sin(im*yn + fb*xn) yn+1 = (ayn + c) % 2pi

This uses a linear congruential function to drive the phase indexing of a sine wave. For im = 1, fb = 0, and a = 1 a normal sinewave results.

sclang default values

```
(audition
```

```
 (out 0 (mul (fb-sine-c ar (fdiv sample-rate 4) 1 0.1 1.1 0.5 0.1 0.1)
             0.2)))
```

increase feedback

```
(let ((fb (line kr 0.01 4 10 do-nothing)))
  (audition
   (out 0 (mul (fb-sine-c ar sample-rate 1 fb 1.1 0.5 0.1 0.1) 0.2))))
```

increase phase multiplier

```
(let ((a (line kr 1 2 10 do-nothing)))
  (audition
   (out 0 (mul (fb-sine-c ar sample-rate 1 0 a 0.5 0.1 0.1) 0.2))))
```

randomly modulate parameters

```
(let* ((x (mouse-x kr 1 12 0 0.1))
       (f (lambda (m a) (mul-add (lf-noise2 kr x) m a))))
  (audition
   (out 0 (mul (fb-sine-c ar
                          (f 10000.0 10000.0)
                          (f 32 33)
                          (f 0.5 0)
                          (f 0.05 1.05)
                          (f 0.3 0.3)
                          0.1
                          0.1)
               0.2))))
```

4.324 See fb-sine-c

4.325 (quad-n rate freq a b c xi)

(quad-l rate freq a b c xi) (quad-c rate freq a b c xi)

freq - iteration frequency in Hertz a, b, c - equation variables xi - initial value of x

General quadratic map chaotic generator. Non-, linear- and cubic- interpolating sound generators based on the difference equation: xn+1 = axn2 + bxn + c

```
(define quad_ quad-c)
```

```
(audition
 (out 0 (mul (quad_ ar 4000 1 -1 -0.75 0) 0.2)))


(let ((r (mouse-x kr 3.5441 4 0 0.1)))
  (audition
   (out 0 (mul (quad_ ar 4000.0 (neg r) r 0 0.1) 0.4))))


(let ((r (mouse-x kr 3.5441 4 0 0.1)))
  (audition
   (out 0 (mul (sin-osc ar (mul-add (quad_ ar 4 (neg r) r 0 0.1) 800 900) 0)
               0.4))))
```

4.326 (lorenz-l rate freq s r b h xi yi zi)

freq - iteration frequency in Hertz s, r, b - equation variables h - integration time step xi - initial value of x yi - initial value of y zi - initial value of z

Lorenz chaotic generator. A strange attractor discovered by Edward N. Lorenz while studying mathematical models of the atmosphere. The system is composed of three ordinary differential equations:

x' = s(y - x) y' = x(r - z) - y z' = xy - bz

The time step amount h determines the rate at which the ODE is evaluated. Higher values will increase the rate, but cause more instability. A safe choice is the default amount of 0.05.

vary frequency

```
(audition
 (out 0 (mul (lorenz-l ar (mouse-x kr 20 sample-rate 0 0.1)
                          10 28 2.667 0.05 0.1 0 0)
             0.3)))
```

randomly modulate params

```
(audition
 (out 0 (mul (lorenz-l ar sample-rate
                          (mul-add (lf-noise0 kr 1) 2 10)
                          (mul-add (lf-noise0 kr 1) 20 38)
                          (mul-add (lf-noise0 kr 1) 1.5 2)
                          0.05
                          0.1 0.0 0.0)
             0.2)))
```

as a frequency control

```
(audition
 (out 0 (mul
         (sin-osc ar (mul-add
                      (lag
                       (lorenz-l ar
                                 (mouse-x kr 1 200 0 0.1)
                                 10
                                 28
                                 2.667
                                 0.05
                                 0.1
                                 0
                                 0)
                       0.003)
                      800 900)
                     0)
         0.4)))
```

4.327 See standard-l.

4.328 See quad-n

4.329 See cusp-n

4.330 See latoocarfian-c.

4.331 (logistic rate chaosParam freq)

UNDOCUMENTED.

Implements the equation: y1 = param * y1 * (1.0 - y1)

```
(audition
 (out 0 (mul (logistic ar 2.9 1000) 0.2)))
```

4.332 See fb-sine-c

4.333 (latoocarfian-c rate freq a b c d xi yi)

(latoocarfian-l rate freq a b c d xi yi) (latoocarfian-n rate freq a b c d xi yi)

This is a function given in Clifford Pickover's book Chaos in Wonderland, pg 26. The function has four parameters a, b, c, and d. The function is:

xnew = sin(y * b) + c * sin(x * b); ynew = sin(x * a) + d * sin(y * a); x = xnew; y = ynew; output = x;

According to Pickover, parameters a and b should be in the range from -3 to +3, and parameters c and d should be in the range from 0.5 to 1.5. The function can, depending on the parameters given, give continuous chaotic output, converge to a single value (silence) or oscillate in a cycle (tone). This UGen is experimental and not optimized currently, so is rather hoggish of CPU.

Default initial parameters.

```
(let ((x (mouse-x kr 20 sample-rate 0 0.1)))
  (audition
   (out 0 (mul (latoocarfian-c ar x 1 3 0.5 0.5 0.5 0.5) 0.2))))
```

randomly modulate all parameters.

```
(audition
 (out 0 (mul (latoocarfian-c ar
                             (fdiv sample-rate 4)
                             (mul-add (lf-noise2 kr 2) 1.5 1.5)
                             (mul-add (lf-noise2 kr 2) 1.5 1.5)
                             (mul-add (lf-noise2 kr 2) 0.5 1.5)
                             (mul-add (lf-noise2 kr 2) 0.5 1.5)
                             0.5
                             0.5)
             0.2)))
```

4.334 (rossler rate chaosParam dt)

The Rossler attractor is a well known chaotic function. The chaosParam can be varied from 1.0 to 25.0 with a dt of 0.04. Valid ranges for chaosParam vary depending on dt.

```
(audition
 (out 0 (mul (rossler ar 4 0.08) 0.1)))
```

4.335 (standard-l rate freq k xi yi)

(standard-n rate freq k xi yi)

Standard map chaotic generator.

freq - iteration frequency in Hertz k - perturbation amount xi - initial value of x yi - initial value of y

A linear-interpolating sound generator based on the difference equations:

xn+1 = (xn + yn+1) % 2pi yn+1 = (yn + ksin(xn)) % 2pi

The standard map is an area preserving map of a cylinder discovered by the plasma physicist Boris Chirikov.

Vary frequency

```
(audition
 (out 0 (mul (standard-l ar (mouse-x kr 20 sample-
rate 0 0.1) 1 0.5 0) 0.3)))
```

Mouse-controlled parameter.

```
(let ((f (fdiv sample-rate 2))
      (x (mouse-x kr 0.9 4 0 0.1)))
  (audition
   (out 0 (mul (standard-l ar f x 0.5 0) 0.3))))
```

As a frequency control

```
(let* ((x (mouse-x kr 0.9 4 0 0.1))
       (f (mul-add (standard-l ar 40 x 0.5 0) 800 900)))
  (audition
   (out 0 (mul (sin-osc ar f 0) 0.4))))
```

4.336 See quad-n

4.337 (cusp-n rate freq a b xi)

(cusp-l rate freq a b xi)

freq - iteration frequency in Hertz a, b - equation variables xi - initial value of x

Cusp map chaotic generator. Non- and linear- interpolating sound generator based on the difference equation:

xn+1 = a - b*sqrt(|xn|)

```
(define cusp_ cusp-l)
```

vary frequency

```
(let ((x (mouse-x kr 20 sample-rate 0 0.1)))
  (audition (out 0 (mul (cusp_ ar x 1.0 1.99 0) 0.3))))
```

mouse-controlled params

```
(let ((x (mouse-x kr 0.9 1.1 1 0.1))
      (y (mouse-y kr 1.8 2 1 0.1)))
  (audition (out 0 (mul (cusp_ ar (fdiv sample-
rate 4) x y 0) 0.3))))
```

as a frequency control

```
(let* ((x (mouse-x kr 0.9 1.1 1 0.1))
       (y (mouse-y kr 1.8 2 1 0.1))
       (f (mul-add (cusp_ ar 40 x y 0) 800 900)))
  (audition (out 0 (mul (sin-osc ar f 0.0) 0.4))))
```

4.338 (lin-cong-c rate freq a c m xi)

(lin-cong-l rate freq a c m xi) (lin-cong-n rate freq a c m xi)

linear congruential chaotic generator.

freq - iteration frequency in Hertz a - multiplier amount c - increment amount m - modulus amount xi - initial value of x

A cubic-interpolating sound generator based on the difference equation:

xn+1 = (axn + c) % m

The output signal is automatically scaled to a range of [-1, 1].

Default initial parameters.

```
(audition
 (out 0 (let ((x (mouse-x kr 20 sample-rate 0 0.1)))
          (mul (lin-cong-c ar x 1.1 0.13 1 0) 0.2))))
```

randomly modulate parameters.

```
(audition
 (out 0 (mul (lin-cong-c ar
                         (mul-add (lf-noise2 kr 1.0) 10000.0 10000.0)
                         (mul-add (lf-noise2 kr 0.1) 0.5 1.4)
                         (mul-add (lf-noise2 kr 0.1) 0.1 0.1)
                         (lf-noise2 kr 0.1)
                         0)
             0.2)))
```

As frequency control...

```
(audition
 (out 0 (mul (sin-osc ar (mul-add (lin-cong-c ar
                                              40
                                              (mul-add (lf-
noise2 kr 0.1) 0.1 1.0)
                                              (mul-add (lf-
noise2 kr 0.1) 0.1 0.1)
                                              (lf-noise2 kr 0.1)
                                              0)
                                  500 600) 0)
             0.4)))
```

4.339 (crackle rate chaosParam)

A noise generator based on a chaotic function. The parameter of the chaotic function has useful values from just below 1.0 to just above 2.0. Towards 2.0 the sound crackles.

The equation implemented is: y0 = fabs(y1 * param - y2 - 0.05f)

```
(audition (out 0 (mul (crackle ar 1.95) 0.5)))
```

Modulate chaos parameter

```
(let ((p (line kr 1.0 2.0 3 remove-synth)))
  (audition (out 0 (mul (crackle ar p) 0.5))))
```

4.340 (henon-n rate freq a b x0 x1)

(henon-l rate freq a b x0 x1) (henon-c rate freq a b x0 x1)

Henon map chaotic generator.

freq - iteration frequency in Hertz – 22050 a, b - equation variables – 1.4, 0.3 x0, x1 - initial and second values of x – 0, 0

A non-interpolating sound generator based on the difference equation:

xn + 2 = 1 - axn + 12 + bxn

This equation was discovered by French astronomer Michel Henon while studying the orbits of stars in globular clusters.

With default initial parameters.

```
(audition
 (out 0 (mul (henon-n ar (mouse-x kr 20 sample-
rate 0 0.1) 1.4 0.3 0 0)
             0.1)))
```

With mouse-control of parameters.

```
(audition
 (out 0 (mul (henon-n ar
                      (fdiv sample-rate 4)
                      (mouse-x kr 1 1.4 0 0.1)
                      (mouse-y kr 0 0.3 0 0.1)
                      0
                      0)
             0.1)))
```

With randomly modulate parameters.

```
(audition
 (out 0 (mul (henon-n ar
                      (fdiv sample-rate 8)
                      (mul-add (lf-noise2 kr 1) 0.2 1.2)
                      (mul-add (lf-noise2 kr 1) 0.15 0.15)
                      0
                      0)
             0.1)))
```

As a frequency control.

```
(let ((x (mouse-x kr 1 1.4 0 0.1))
      (y (mouse-y kr 0 0.3 0 0.1))
      (f 40))
```

```
  (audition
   (out 0 (mul (sin-osc ar (mul-add (henon-
n ar f x y 0 0) 800 900) 0)
               0.4))))
```

4.341 See henon-n

4.342 See lin-cong-c.

4.343 See GbmanL.

4.344 See GbmanL.

4.345 See henon-n

4.346 See lin-cong-c.

4.347 (gbman-c rate freq xi yi)

(gbman-l rate freq xi yi) (gbman-n rate freq xi yi)

Gingerbreadman map chaotic generator. Cubic, linear and non-interpolating variants.

freq - iteration frequency in Hertz xi - initial value of x yi - initial value of y

A linear-interpolating sound generator based on the difference equations:

xn+1 = 1 - yn + |xn| yn+1 = xn

The behavior of the system is dependent only on its initial conditions qand cannot be changed once it's started.

Reference: Devaney, R. L. "The Gingerbreadman." Algorithm 3, 15-16, Jan. 1992.

sclang default initial parameters.

```
(audition
 (out 0 (mul (gbman-l ar (mouse-x kr 20 sample-
rate 0 0.1) 1.2 2.1) 0.1)))
```

Different initial parameters.

```
(audition
 (out 0 (mul (gbman-l ar (mouse-x kr 20 sample-
rate 0 0.1) -0.7 -2.7) 0.1)))
```

Wait for it...

```
(audition
 (out 0 (mul (gbman-l ar (mouse-x kr 20 sample-
rate 0 0.1) 1.2 2.0002) 0.1)))
```

As a frequency control

```
(audition
 (out 0 (mul (sin-osc ar (mul-add (gbman-l ar 40 1.2 2.1) 400 500) 0) 0.4)))
```

4.348 (lin-pan2 in pos level)

Two channel linear pan. See pan2.

```
(audition (out 0 (lin-pan2 (pink-noise ar) (f-sin-
osc kr 2 0) 0.1)))
```

```
(audition (out 0 (lin-pan2 (f-sin-osc ar 800 0.1) (f-sin-
osc kr 3 0) 0.1)))
```

4.349 (rotate2 x y pos)

Rotate a sound field. rotate2 can be used for rotating an ambisonic B-format sound field around an axis. rotate2 does an equal power rotation so it also works well on stereo sounds. It takes two audio inputs (x, y) and an angle control (pos). It outputs two channels (x, y).

It computes:

xout = cos(angle) * xin + sin(angle) * yin; yout = cos(angle) * yin - sin(angle) * xin;

where angle = pos * pi, so that -1 becomes -pi and +1 becomes +pi. This allows you to use an lf-saw to do continuous rotation around a circle.

The control pos is the angle to rotate around the circle from -1 to +1. -1 is 180 degrees, -0.5 is left, 0 is forward, +0.5 is right, +1 is behind.

```
(let* ((p (mul (white-noise ar) 0.05))
```

```
        (q (mul (mix (lf-saw ar (make-mce (list 200 200.37 201)) 0)) 0.03))
        (encoded (add (pan-b2 p -0.5 1) (pan-b2 q -0.5 1)))
        (rotated (rotate2 (mce-channel encoded 1)
                          (mce-channel encoded 2)
                          (mouse-x kr -1 1 0 0.1)))
        (decoded (decode-b2 4
                            (mce-channel encoded 0)
                            (mce-channel rotated 0)
                            (mce-channel rotated 1)
                            0.5)))
  (audition (out 0 decoded)))
```

Rotation of stereo sound, via LFO.

```
(let ((x (mul (pink-noise ar) 0.4))
      (y (mul (lf-tri ar 800 0) (mul (lf-pulse kr 3 0 0.3) 0.2))))
  (audition (out 0 (rotate2 x y (lf-saw kr 0.1 0)))))
```

Rotation of stereo sound, via mouse.

```
(let ((x (mix-fill 4 (lambda (_) (mul (lf-
saw ar (rand 198 202) 0) 0.1))))
      (y (mul (sin-osc ar 900 0) (mul (lf-
pulse kr 3 0 0.3) 0.2))))
  (audition (out 0 (rotate2 x y (mouse-x kr 0 2 0 0.1)))))
```

4.350 (decode-b2 numChannels w x y orientation)

2D Ambisonic B-format decoder.

Decode a two dimensional ambisonic B-format signal to a set of speakers in a regular polygon. The outputs will be in clockwise order. The position of the first speaker is either center or left of center.

The number of output speakers is typically 4 to 8.

The parameters w, x and y are the B-format signals.

The parameter orientation should be zero if the front is a vertex of the polygon. The first speaker will be directly in front. Should be 0.5 if the front bisects a side of the polygon. Then the first speaker will be the one left of center. Default is 0.5.

```
(let* ((p (pink-noise ar))
       (encoded (pan-b2 p (mouse-x kr -1 1 0 0.1) 0.1))
```

```
      (decoded (decode-b2 4
                          (mce-channel encoded 0)
                          (mce-channel encoded 1)
                          (mce-channel encoded 2)
                          0)))
  (audition (out 0 decoded)))
```

4.351 (pan2 in pos level)

Two channel equal power panner. The pan position is bipolar, -1 is left, +1 is right. The level is a control rate input.

```
(let ((p (f-sin-osc kr 2 0)))
  (audition (out 0 (pan2 (pink-noise ar) p 0.3))))

(let ((x (mouse-x kr -1 1 0 0.1))
      (y (mouse-y kr 0 1 0 0.1)))
  (audition (out 0 (pan2 (pink-noise ar) x y))))
```

4.352 (pan-b2 in azimuth gain)

2D Ambisonic B-format panner. Encode a mono signal to two dimensional ambisonic B-format. The azimuth parameter is the position around the circle from -1 to +1. -1 is behind, -0.5 is left, 0 is forward, +0.5 is right, +1 is behind.

```
(let* ((p (pink-noise ar))
       (encoded (pan-b2 p (mouse-x kr -1 1 0 0.1) 0.1))
       (decoded (decode-b2 4
                           (mce-channel encoded 0)
                           (mce-channel encoded 1)
                           (mce-channel encoded 2)
                           0.5)))
  (audition (out 0 decoded)))
```

4.353 (detect-silence in amp time doneAction)

If the signal at ‘in’ falls below ‘amp’ for ‘time’ seconds then ‘doneAction’ is raised.

```
(let ((s (mul (sin-osc ar 440 0) (mouse-y kr 0 0.4 0 0.1))))
  (audition (mrg2 (detect-silence s 0.1 0.2 remove-synth)
                  (out 0 s))))

(with-sc3 display-server-status)
```

4.354 (line rate start end dur doneAction)

Generates a line from the start value to the end value.

start - starting value end - ending value dur - duration in seconds

Note: The SC3 UGen reorders the mul and add inputs to precede the doneAction input.

```
(let ((f (line kr 200 17000 5 remove-synth)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.355 (free trig nodeID)

When triggered frees a node.

trig - when triggered, frees node nodeID - node to be freed

```
(with-sc3
 (lambda (fd)
   (send-synth fd "a" (out 0 (mul (sin-osc ar 800 0) 0.1)))
   (send-synth fd "b" (mrg2 (out 1 (mul (pink-noise ar) 0.1))
                            (free (dust ar 6) 1001)))
   (send fd (s-new0 "a" 1001 0 0))
   (send fd (s-new0 "b" -1 0 0))))


(with-sc3 reset)
```

4.356 (pause-self-when-done src)

pause the synth when the 'done' flag of the unit at 'src' is set.

```
(let* ((x (mouse-x kr -1 1 0 0.1))
       (e (linen x 1 0.1 1 pause-synth)))
  (audition (out 0 (mul (sin-osc ar 440 0) e))))


(let* ((x (mouse-x kr -1 1 0 0.1))
       (e (linen x 2 0.1 2 do-nothing)))
  (audition (mrg2 (pause-self-when-done e)
                  (out 0 (mul (sin-osc ar 440 0) e)))))
```

4.357 (pause-self src)

pause enclosing synth when input signal crosses from non-positive to positive. If the synth is restarted and the gate reset the synthesis *not* paused a second time.

```
(audition
 (mrg2 (pause-self (mouse-x kr -1 1 0 0.1))
       (out 0 (mul (sin-osc ar 440 0) 0.1))))
```

4.358 (env-gen rate gate levelScale levelBias timeScale doneAction envelope)

A segment based envelope generator. Note that the SC3 language reorders the inputs to this UGen so that the envelope is the first argument.

There are utilities for contructing the envelope argument.

The arguments for levelScale, levelBias, and timeScale are polled when the env-gen is triggered and remain constant for the duration of the envelope.

envelope - an breakpoint set

gate - this triggers the envelope and holds it open while > 0. If the Env is fixed-length (e.g. Env.linen, Env.perc), the gate argument is used as a simple trigger. If it is an sustaining envelope (e.g. Env.adsr, Env.asr), the envelope is held open until the gate becomes 0, at which point is released.

levelScale - scales the levels of the breakpoints.

levelBias - offsets the levels of the breakpoints.

timeScale - scales the durations of the segments.

doneAction - an integer representing an action to be executed when the env is finished playing. This can be used to free the enclosing synth, etc.

```
(import (rhs) (rsc3))
```

Percussive envelope

```
(let* ((d (env-perc 0.01 1 1 (list -4 -4)))
       (e (env-gen kr 1 0.1 0 1 remove-synth d)))
  (audition (out 0 (mul e (sin-osc ar 440 0)))))
```

The break-point assistant makes a static envelope from a co-ordinate list. There is a duration and amplitude scalar.

```
(let* ((d (env-coord (list (cons 0 0)
                           (cons 0.75 1)
                           (cons 1 0))
                     1 1 (replicate 3 "linear")))
       (e (env-gen kr 1 0.1 0 1 remove-synth d)))
  (audition (out 0 (mul e (sin-osc ar 440 0)))))
```

Trapezoidal

```
(let* ((d (env-trapezoid 0 0.25 2 0.1))
       (e (env-gen kr 1 0.1 0 1 remove-synth d)))
  (audition (out 0 (mul e (sin-osc ar 440 0)))))
```

0.0 3 -1 -1 0.1 0.5 1.0 0.0 0.1 0.0 1.0 0.0 0.0 1.5 1.0 0.0

```
(env-trapezoid 0 0.25 2 0.1)
```

4.359 (free-self-when-done src)

free the synth when the 'done' flag of the unit at 'src' is set.

```
(let* ((x (mouse-x kr -1 1 0 0.1))
       (e (linen x 1 0.1 1 remove-synth)))
  (audition (out 0 (mul (sin-osc ar 440 0) e))))
```

```
(let* ((x (mouse-x kr -1 1 0 0.1))
       (e (linen x 2 0.1 2 do-nothing)))
  (audition (mrg2 (free-self-when-done e)
                  (out 0 (mul (sin-osc ar 440 0) e)))))
```

4.360 (pause gate nodeID)

When triggered pauses a node.

gate - when gate is 0, node is paused, when 1 it runs nodeID - node to be paused

```
(with-sc3
 (lambda (fd)
   (send-synth fd "a" (out 0 (mul (sin-osc ar 800 0) 0.1)))
   (send-synth fd "b" (letc ((g 1))
                        (Mrg (out 1 (mul (pink-noise ar) 0.05))
                             (pause g 1001))))
```

```
(send fd (/s_new "a" 1001 0 0))
(send fd (/s_new "b" 1002 0 0))
(sleep 1)
(send fd (/n_set 1002 "g" 0))
(sleep 1)
(send fd (/n_set 1002 "g" 1))
(sleep 1)
(reset fd)))
```

4.361 (x-line rate start end dur doneAction)

Exponential line generator. Generates an exponential curve from the start value to the end value. Both the start and end values must be non-zero and have the same sign.

start - starting value end - ending value dur - duration in seconds doneAction - a doneAction to be evaluated when the x-line is completed. See env-gen for details.

```
(let ((f (x-line kr 200 17000 10 remove-synth)))
  (audition (out 0 (mul (sin-osc ar f 0) 0.1))))
```

4.362 (done src)

outputs a unit signal if the 'done' flag of the unit at 'src' is set, else output zero.

```
(let* ((x (mouse-x kr -1 1 0 0.1))
       (e (linen x 0.1 0.1 0.5 do-nothing))
       (l (mul (sin-osc ar 880 0) 0.1))
       (r (sin-osc ar 440 0)))
  (audition (out 0 (mce2 (mul (done e) l)
                         (mul e r)))))
```

4.363 (linen gate attackTime susLevel releaseTime doneAction)

A linear envelope generator. The done flag is set when the envelope reaches zero.

Note that the sustain level input is consulted only at the instant when the gate is opened.

```
(let ((e (linen (impulse kr 2 0) 0.01 0.1 0.4 do-nothing)))
  (audition (out 0 (mul (sin-osc ar 440 0) e))))


(let* ((y (mouse-y kr 0.1 0.5 0 0.1))
```

```
      (x (mouse-x kr -1 1 0 0.1))
      (e (linen x 1 y 1.0 do-nothing))
      (o (sin-osc ar 440 0)))
  (audition (out 0 (mul o e))))
```

Open gate for a random interval.

```
(let* ((r (rand 0.05 0.4))
       (u (letc ((gate 0))
            (let ((e (linen gate 0.1 0.2 0.1 do-nothing)))
              (out 0 (mul (sin-osc ar 440 0) e)))))
       (g (encode-graphdef (synthdef "linen" u))))
  (with-sc3
   (lambda (fd)
     (async fd (d-recv g))
     (send fd (s-new0 "linen" 1001 1 1))
     (send fd (bundle (utc) (n-set1 1001 "gate" 1)))
     (send fd (bundle (+ (utc) r)
                      (n-set1 1001 "gate" 0)))
     (sleep (* r 4))
     (send fd (n-free1 1001)))))
```

4.364 (free-self src)

free enclosing synth when the input signal 'src' crosses from non-positive to positive.

```
(audition
 (mrg2 (free-self (mouse-x kr -1 1 0 0.1))
       (out 0 (mul (sin-osc ar 440 0) 0.1))))
```

5 rsc3: tutorials

Contents

A Gentle Introduction to SuperCollider (2nd edition).pdf

SuperCollider tutorial - Nick Collins

rsc3-for-schemers.md

rsc3-tutorial.md

Index

(abs a), 113
(abs-dif a b), 93
(add a b), 93
(allpass-n in maxDelayTime delayTime decayTime), 140
(am-clip a b), 93
(amp-comp freq root exp), 25
(amp-compA freq root minAmp rootAmp), 31
(amp-db a), 112
(amplitude rate in attackTime releaseTime), 51
(arc-cos a), 114
(arc-sin a), 115
(arc-tan a), 110
(Atan2 x y), 89
(ball in g damp friction), 147
(bpf in freq rq), 24
(bpz2 in), 25
(brf in freq rq), 36
(brz2 in), 35
(buf-allpass-c buf in delaytime decaytime), 141
(buf-channels rate bufnum), 46
(buf-comb-c buf in delaytime decaytime), 143
(buf-delay-c buf in delaytime), 142
(buf-dur rate bufnum), 45
(buf-frames rate bufnum), 44
(buf-rate-scale rate bufnum), 45
(buf-rd numChannels rate bufnum phase loop interpolation), 144
(buf-sample-rate rate bufnum), 46
(ceil a), 114
(clip in lo hi), 30
(clip-noise rate), 102
(clip2 a b), 89
(coin-gate prob in), 105
(comb-n in maxDelayTime delayTime decayTime), 139
(compander input control thresh slopeBelow slopeAbove clampTime relaxTime), 49
(convolution in kernel framesize), 118
(convolution2 in bufnum trigger framesize), 126
(cos a), 117
(cos-h a), 112
(cps-midi a), 111
(cps-oct a), 110
(crackle rate chaosParam), 163
(cubed a), 115
(cusp-n rate freq a b xi), 161
(db-amp a), 111
(dbrown length lo hi step), 155
(dbufrd bufnum phase loop), 149
(decay in decayTime), 84
(decay2 in attackTime decayTime), 88
(decode-b2 numChannels w x y orientation), 167
(degree-to-key bufnum in octave), 85
(delay-n in maxDelayTime delayTime), 147
(delay1 in), 145
(delay2 in), 139
(demand trig reset ugens), 151
(demand-env-gen rate levels times shapes curves gate reset, 150
(detect-silence in amp time doneAction), 168
(dgeom length start grow), 153
(dif-sqr a b), 94
(disk-in num-channels rate bufnum), 42
(disk-out bufnum channels), 43
(distort a), 110
(done src), 172
(drand length array), 154
(dseq length array), 156
(dser length array), 153
(dseries length start step), 154
(dswitch index array), 155
(dswitch1 index array), 147
(dust rate density), 108
(dust2 rate density), 107
(duty rate duration reset doneAction level), 152

(dwhite length lo hi), 149
(dyn-klank in freqScale freqOffset decayScale spec), 37
(env-gen rate gate levelScale levelBias timeScale doneAction envelope), 170
(excess a b), 95
(exp a), 115
(exp-rand lo hi), 109
(fb-sine-c rate freq im fb a c xi yi), 156
(fdiv a b), 98
(fft buffer in hop wintype active), 124
(fold in lo hi), 23
(fold2 a b), 95
(formlet in freq attackTime decayTime), 23
(fos in a0 a1 b1), 24
(frac a), 109
(free trig nodeID), 169
(free-self src), 173
(free-self-when-done src), 171
(free-verb in mix room damp), 137
(freq-shift input shift phase), 33
(gate in trig), 51
(gbman-c rate freq xi yi), 165
(ge a b), 93
(grain-buf nc tr dur sndbuf rate pos interp pan envbuf), 39
(grain-fm nc tr dur carfreq modfreq index pan envbuf), 41
(grain-sin nc tr dur freq pan envbuf), 40
(Grainin nc tr dur in pan envbuf), 39
(gray-noise rate), 104
(gt a b), 92
(hasher in), 100
(henon-n rate freq a b x0 x1), 163
(hpf in freq), 29
(hpz1 in), 32
(hpz2 in), 28
(hypot x y), 96
(i-rand lo hi), 104
(Ifft buffer wintype), 131
(in num-channels rate bus), 80
(in-feedback num-channels bus), 82
(in-range in lo hi), 59
(in-trig num-channels bus), 80
(is-negative a), 116
(is-positive a), 113
(is-strictly-positive a), 112
(k2a in), 88
(key-state rate keynum minval maxval lag), 86
(klang rate freqScale freqOffset spec), 79
(klank in freqScale freqOffset decayScale spec), 27
(lag in lagTime), 36
(lag-in num-channels bus lag), 79
(lag2 in lagTime), 26
(lag3 in lagTime), 31
(last-value in diff), 55
(latch in trig), 84
(latoocarfian-c rate freq a b c d xi yi), 160
(le a b), 97
(leak-dc in coef), 28
(lf-cub rate freq iphase), 76
(lf-noise0 rate freq), 103
(lf-pulse rate freq iphase width), 77
(lf-saw rate freq iphase), 65
(lf-tri rate freq iphase), 62
(lfclip-noise rate freq), 102
(lfd-noise0 rate freq), 100
(lfdclip-noise rate freq), 105
(limiter input level lookAheadTime), 30
(lin-cong-c rate freq a c m xi), 162
(lin-exp in srclo srchi dstlo dsthi), 28
(lin-lin in srclo srchi dstlo dsthi), 24
(lin-pan2 in pos level), 166
(lin-rand lo hi minmax), 108
(line rate start end dur doneAction), 169
(linen gate attackTime susLevel releaseTime doneAction), 172
(local-in num-channels rate), 81
(log10 a), 113
(log2 a), 114
(logistic rate chaosParam freq), 159
(lorenz-l rate freq s r b h xi yi zi), 158

(lpf in freq), 32
(lpz1 ar in), 33
(lpz2 ar in), 28
(lt a b), 99
(mantissa-mask in bits), 108
(max a b), 94
(median length in), 30
(midi-cps a), 113
(min a b), 99
(mix UGen), 83
(mix-fill n f), 84
(Mod a b), 98
(moog-ff in freq gain reset), 34
(most-change a b), 52
(mouse-button rate minval maxval lag), 87
(mouse-x rate minval maxval warp lag), 87
(mouse-y rate minval maxval warp lag), 85
(mrg2 left right), 86
(mul a b), 98
(mul-add a b c), 89
(n-rand lo hi n), 104
(neg a), 114
(normalizer in level dur), 29
(oct-cps a), 116
(offset-out bufferindex inputs), 81
(one-pole in coef), 36
(one-zero in coef), 26
(osc rate bufnum freq phase), 64
(osc-n rate bufnum freq phase), 64
(out bufferindex inputs), 83
(Packfft chain bufsize frombin tobin zeroothers magsphases), 133
(pan-b2 in azimuth gain), 168
(pan2 in pos level), 168
(pause gate nodeID), 171
(pause-self src), 170
(pause-self-when-done src), 169
(peak trig reset), 57
(peak-follower in decay), 55
(phasor trig rate start end resetpos), 57
(pink-noise rate), 104
(pitch in initFreq minFreq maxFreq execFreq maxBinsPerOctave, 49
(pitch-shift in winSize pchRatio pchDispersion timeDispersion), 143
(play-buf numChannels bufnum rate trigger startPos loop), 137
(pluck in tr maxdelaytime delaytime decaytime coef), 142
(pm-osc rate carfreq modfreq index modphase), 61
(poll trig in trigid label), 51
(pow a b), 91
(pulse rate freq width), 69
(pulse-count trig reset), 53
(pulse-divider trig div start), 51
(pv-add bufferA bufferB), 128
(pv-bin-scramble buffer wipe width trig), 131
(pv-bin-shift buffer stretch shift), 123
(pv-bin-wipe bufferA bufferB wipe), 125
(pv-brick-wall buffer wipe), 122
(pv-conformal-map buffer real imag), 129
(pv-copy bufferA bufferB), 121
(pv-copyPhase bufferA bufferB), 125
(pv-diffuser buffer trig), 130
(pv-hainsworth-foote buffer proph propf threshold waittime), 119
(pv-jensen-andersen buffer propsc prophfe prophfc propsf threshold waittime), 118
(pv-local-max buffer threshold), 129
(pv-mag-above buffer threshold), 123
(pv-mag-below buffer threshold), 118
(pv-mag-mul bufferA bufferB), 129
(pv-mag-noise buffer), 117
(pv-mag-shift buffer stretch shift), 137
(pv-mag-smear buffer bins), 122
(pv-mag-squared buffer), 117
(pv-max bufferA bufferB), 131
(pv-min bufferA bufferB), 117
(pv-mul bufferA bufferB), 117
(pv-phase-shift buffer shift), 121
(pv-phase-shift270 buffer), 119
(pv-phase-shift90 buffer), 126
(pv-rand-comb buffer wipe trig), 129

(pv-rand-wipe bufferA bufferB wipe trig), 132
(pv-rect-comb buffer numTeeth phase width), 136
(pv-rect-comb2 bufferA bufferB numTeeth phase width), 128
(PV_Magclip buffer threshold), 135
(PV_Magfreeze buffer freeze), 135
(pvcollect chain numframes func frombin to-bin zeroothers), 134
(quad-n rate freq a b c xi), 157
(rand lo hi), 104
(rand-id rate id), 107
(rand-seed rate trig seed), 100
(record-buf bufnum offset reclevel prelevel run loop trigger inputs), 145
(replace-out bufferindex inputs), 81
(resonz in freq bwr), 34
(Rhpf in freq rq), 35
(ring1 a b), 92
(ring2 a b), 98
(ring3 a b), 94
(ring4 a b), 91
(ringz in freq decayTime), 36
(Rlpf in freq rq), 22
(rossler rate chaosParam dt), 160
(rotate2 x y pos), 166
(round a b), 99
(round-up a b), 90
(running-max in trig), 56
(running-min in trig), 60
(running-sum in numsamp), 48
(saw rate freq), 61
(scale-neg a b), 97
(schmidt in lo hi), 57
(select which array), 73
(send-trig in id value), 59
(set-reset-ff trig reset), 60
(shaper bufnum in), 69
(sign a), 114
(sin-h a), 117
(sin-osc rate freq phase), 78
(slew in up dn), 87
(slope in), 48
(soft-clip a), 111
(sos in a0 a1 a2 b1 b2), 29
(sound-in channel), 79
(sqr-dif a b), 95
(sqr-sum a b), 96
(squared a), 115
(standard-l rate freq k xi yi), 161
(stepper trig reset min max step resetval), 53
(sub a b), 90
(sum-sqr a b), 97
(sweep trig rate), 58
(sync-saw rate syncFreq sawFreq), 77
(t-choose trig array), 78
(t-delay trigger delayTime), 60
(t-duty rate duration reset doneAction level gap), 148
(t-exp-rand lo hi trig), 106
(t-grains numChannels trigger bufnum rate centerPos dur pan amp interp), 62
(t-rand lo hi trig), 106
(tan-h a), 110
(thresh a b), 94
(ti-rand lo hi trig), 102
(timer trig), 60
(toggle-ff trig), 58
(trig in dur), 53
(trig1 in dur), 57
(trunc a b), 90
(tw-choose trig array weights normalize), 66
(tw-index in normalize array), 63
(two-pole in freq radius), 32
(two-zero in freq radius), 29
(u:floor a), 110
(u:log a), 109
(u:sin a), 116
(u:sqrt a), 111
(u:tan a), 112
(v-osc rate bufpos freq phase), 70
(v-osc3 rate bufpos freq1 freq2 freq3), 74
(var-saw rate freq iphasewidth), 71

(warp1 nc buf ptr freqScale windowSize envbuf overlaps windowrandRatio interp), 41
(white-noise rate), 106
(wrap in lo hi), 25
(wrap-index bufnum in), 85
(wrap2 a b), 99
(x-line rate start end dur doneAction), 172
(x-out buffer-index xfade inputs), 83
(zero-crossing in), 47
/b_alloc Allocate buffer space., 9
/b_allocRead Allocate buffer space and read a sound file., 15
/b_close, 10
/b_fill Fill ranges of sample value(s), 14
/b_free Free buffer data., 18
/b_gen Call a command to fill a buffer, 8
/b_get Get sample value(s), 13
/b_getn Get ranges of sample value(s), 14
/b_query, 18
/b_read Read sound file data into an existing buffer., 12
/b_set Set sample value(s), 16
/b_setn Set ranges of sample value(s), 11
/b_write Write sound file data., 14
/b_zero Zero sample data, 17
/c_fill Fill ranges of bus value(s), 19
/c_get Get bus value(s), 16
/c_getn Get ranges of bus value(s), 19
/c_set Set bus value(s), 11
/c_setn Set ranges of bus value(s), 7
/clearSched Clear all scheduled bundles., 18
/d_free delete synth definition, 13
/d_load Load synth definition, 10
/d_recv Receive a synth definition file, 15
/dumpOSC Display incoming OSC messages, 17
/g_deepFree Free all synths in this group and all its sub-groups., 7
/g_freeAll Delete all nodes in a group., 19
/g_head Add node to head of group, 19
/g_new Create a new group, 16
/g_tail Add node to tail of group, 18
/n_after Place a node after another, 13
/n_before Place a node before another, 10
/n_fill Fill ranges of a node's control value(s), 10
/n_free Delete a node., 12
/n_map Map a node's controls to read from a bus, 15
/n_mapn Map a node's controls to read from buses, 8
/n_query Get info about a node, 9
/n_run Turn node on or off, 12
/n_set Set a node's control value(s), 13
/n_setn Set ranges of a node's control value(s), 17
/n_trace Trace a node, 9
/notify Register to receive notifications from server, 18
/nrt_end end real time mode, close file, 12
/quit Quit program, 19
/s_get Get control value(s), 7
/s_getn Get ranges of control value(s), 7
/s_new Create a new synth, 10
/s_noid Auto-reassign synth's ID to a reserved value, 14
/status Query the status, 16
/sync Notify when async commands have completed., 7
/tr A trigger message, 17
/u_cmd send a command to a unit generator, 15
blip, 73
brown-noise, 109
brown-noise, 109
buf-wr, 72
c-osc, 74
eq, 97
formant, 73
Further, 4
gendy1, 66
impulse, 72
index, 77
num-audio-buses, 46
num-buffers, 44

num-control-buses, 44
num-input-buses, 44
num-output-buses, 44
num-running-synths, 46
radians-per-sample, 45
Reference & resuscitation, 2
rsc3, 1
rsc3: server commands, 5
rsc3: SuperCollider client , 1
rsc3: tutorials, 174
rsc3: ugens, 20
sample-dur, 45
sample-rate, 45
SC2: Note extra iphase argument., 70
See allpass-n, 142
See buf-comb-c, 147
See buf-delay-c, 142
See Bufallpass-c, 140
See comb-n, 141
See cusp-n, 159
See dbrown, 155
See delay-n, 147
See drand, 154
See dwhite, 149
See fb-sine-c, 160
See freeVerb, 137
see g-new, 19
See GbmanL., 165
See henon-n, 165
See latoocarfian-c., 159
See lf-cub., 77
See lf-noise0, 108
See lfd-noise0, 107
See lin-cong-c., 165
see n-set, 12
See quad-n, 161
see s-new, 10
See standard-l., 159
subsample-offset, 46