C.1.56. randomGauss (rg)

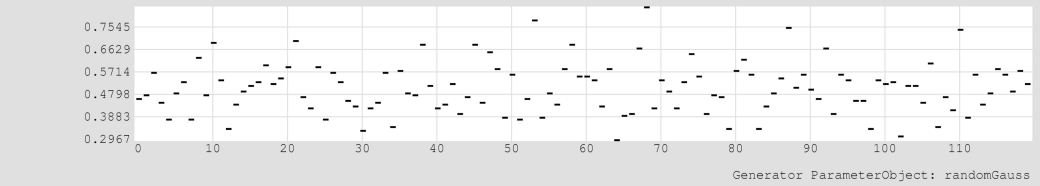
randomGauss, mu, sigma, min, max

Description: Provides random numbers between 0 and 1 within a Gaussian distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: suggested values: mu = 0.5, sigma = 0.1.

Arguments: (1) name, (2) mu, (3) sigma, (4) min, (5) max

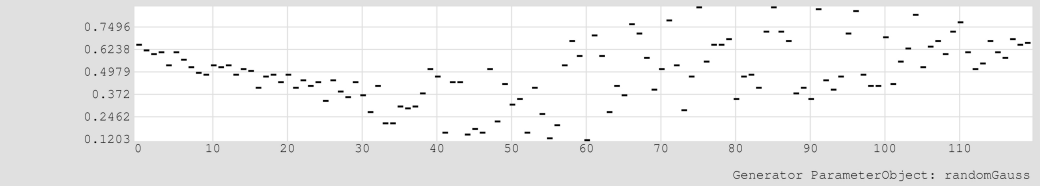
Sample Arguments: `rg, 0.5, 0.1, 0, 1`

Example C-109. randomGauss Demonstration 1




```
randomGauss, 0.5, 0.1, (constant, 0), (constant, 1)
```

Example C-110. randomGauss Demonstration 2




```
randomGauss, 0.5, 0.5, (waveSine, event, (constant, 120), 0.25, (constant, 0),
(constant, 0.5)), (waveSine, event, (constant, 120), 0.5, (constant, 1),
(constant, 0.5))
```

C.1.57. randomInverseExponential (rie)

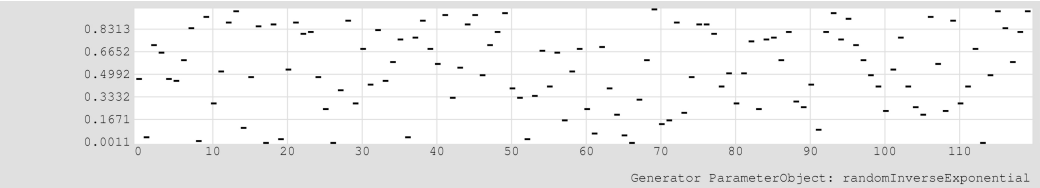
randomInverseExponential, lambda, min, max

Description: Provides random numbers between 0 and 1 within an inverse exponential distribution. Lambda values control the spread of values; larger values (such as 10) increase the probability of events near the maximum. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) lambda, (3) min, (4) max

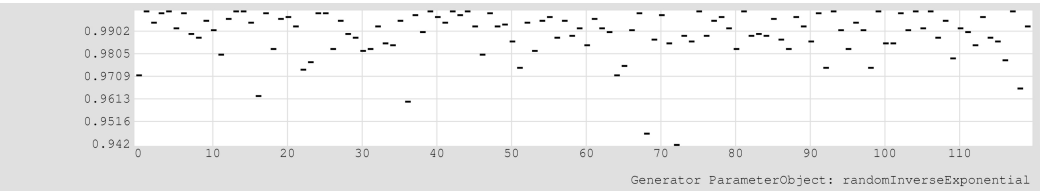
Sample Arguments: `rie, 0.5, 0, 1`

Example C-111. randomInverseExponential Demonstration 1



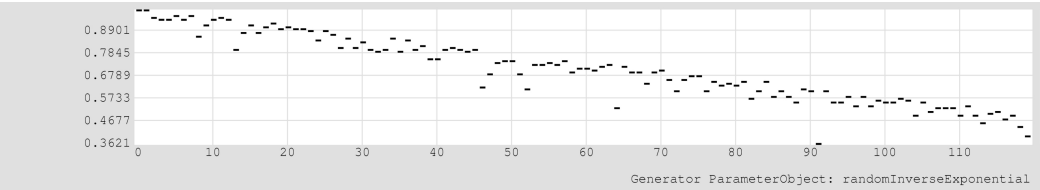

```
randomInverseExponential, 0.5, (constant, 0), (constant, 1)
```

Example C-112. randomInverseExponential Demonstration 2




```
randomInverseExponential, 100.0, (constant, 0), (constant, 1)
```

Example C-113. randomInverseExponential Demonstration 3




```
randomInverseExponential, 10.0, (breakPointLinear, event, loop,
((0,0.5),(120,0))), (breakPointLinear, event, loop, ((0,1),(120,0.5)))
```

C.1.58. randomInverseLinear (ril)

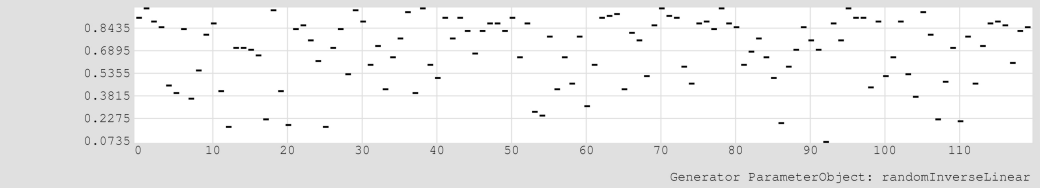
randomInverseLinear, min, max

Description: Provides random numbers between 0 and 1 within a linearly increasing distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: values are distributed more strongly toward max.

Arguments: (1) name, (2) min, (3) max

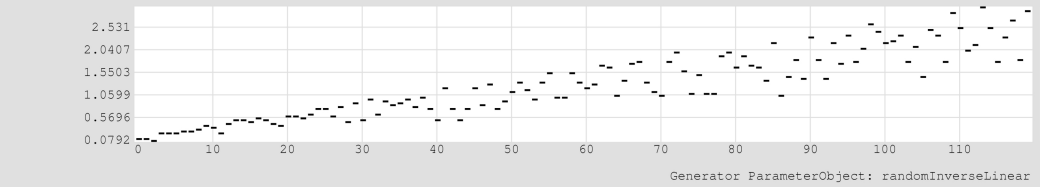
Sample Arguments: `ril, 0, 1`

Example C-114. randomInverseLinear Demonstration 1




```
randomInverseLinear, (constant, 0), (constant, 1)
```

Example C-115. randomInverseLinear Demonstration 2




```
randomInverseLinear, (accumulator, 0, (constant, 0.01)), (accumulator, 0.2,
(constant, 0.03))
```

C.1.59. randomInverseTriangular (rit)

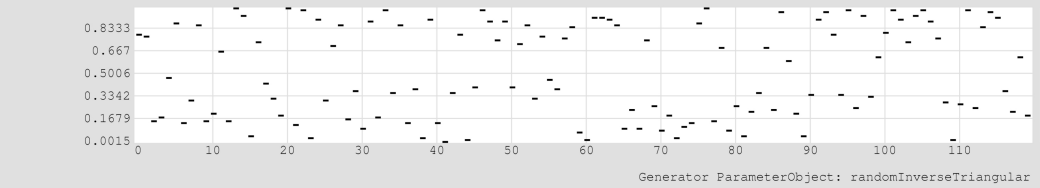
randomInverseTriangular, min, max

Description: Provides random numbers between 0 and 1 within an inverse triangular distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: values are distributed more strongly away from the mean of min and max.

Arguments: (1) name, (2) min, (3) max

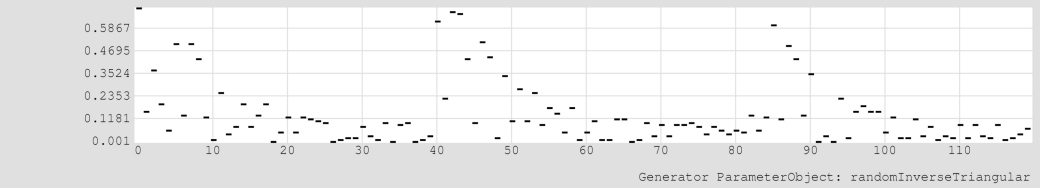
Sample Arguments: `rit, 0, 1`

Example C-116. randomInverseTriangular Demonstration 1




```
randomInverseTriangular, (constant, 0), (constant, 1)
```

Example C-117. randomInverseTriangular Demonstration 2




```
randomInverseTriangular, (constant, 0), (wavePowerDown, event, (constant, 40),
0, 2, (constant, 1), (constant, 0.1))
```

C.1.60. randomLinear (rl)

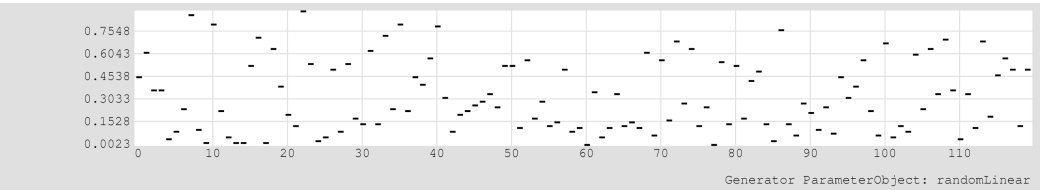
randomLinear, min, max

Description: Provides random numbers between 0 and 1 within a linearly decreasing distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: values are distributed more strongly toward min.

Arguments: (1) name, (2) min, (3) max

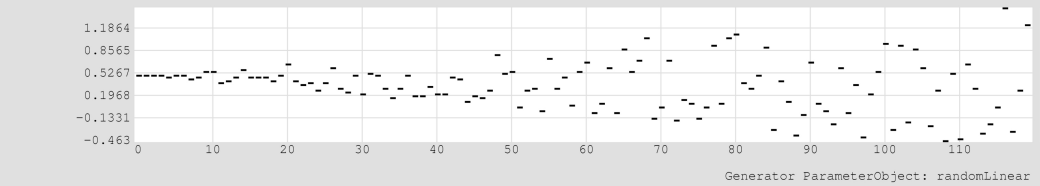
Sample Arguments: `rl, 0, 1`

Example C-118. randomLinear Demonstration 1



```
randomLinear, (constant, 0), (constant, 1)
```

Example C-119. randomLinear Demonstration 2




```
randomLinear, (accumulator, 0.5, (constant, -0.01)), (accumulator, 0.5,
(constant, 0.01))
```

C.1.61. randomTriangular (rt)

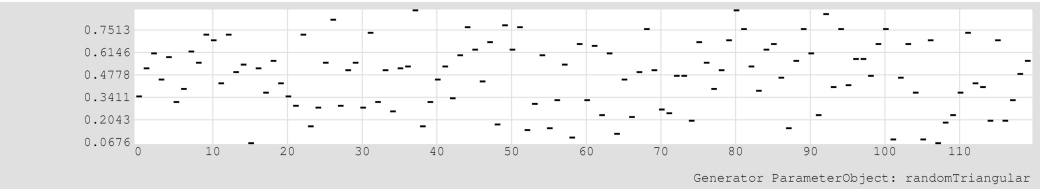
randomTriangular, min, max

Description: Provides random numbers between 0 and 1 within a triangular distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: values are distributed more strongly toward the mean of min and max.

Arguments: (1) name, (2) min, (3) max

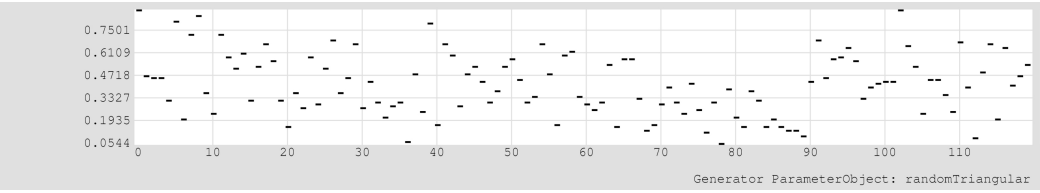
Sample Arguments: `rt, 0, 1`

Example C-120. randomTriangular Demonstration 1




```
randomTriangular, (constant, 0), (constant, 1)
```

Example C-121. randomTriangular Demonstration 2




```
randomTriangular, (constant, 0), (wavePowerDown, event, (constant, 90), 0,
-1.5, (constant, 1), (constant, 0))
```

C.1.62. randomUniform (ru)

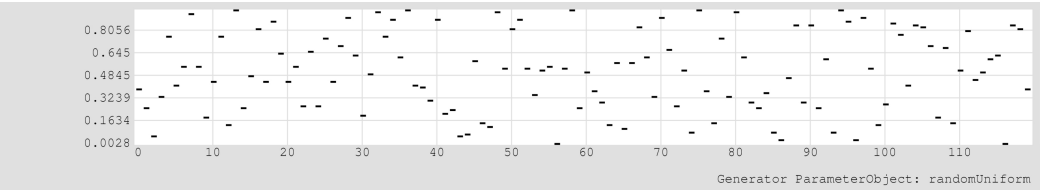
randomUniform, min, max

Description: Provides random numbers between 0 and 1 within an uniform distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: values are evenly distributed between min and max.

Arguments: (1) name, (2) min, (3) max

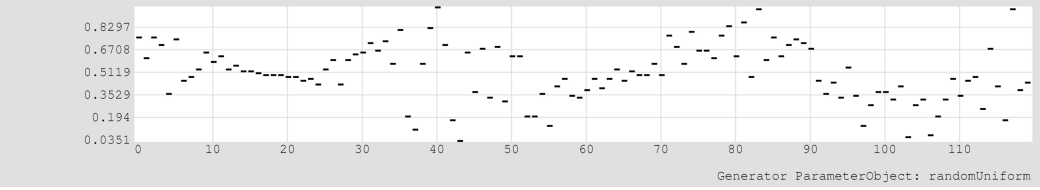
Sample Arguments: `ru, 0, 1`

Example C-122. randomUniform Demonstration 1




```
randomUniform, (constant, 0), (constant, 1)
```

Example C-123. randomUniform Demonstration 2



```
randomUniform, (waveSine, event, (constant, 60), 0, (constant, 0.5),
(constant, 0)), (waveSine, event, (constant, 40), 0.25, (constant, 1),
(constant, 0.5))
```

C.1.63. randomWeibull (rw)

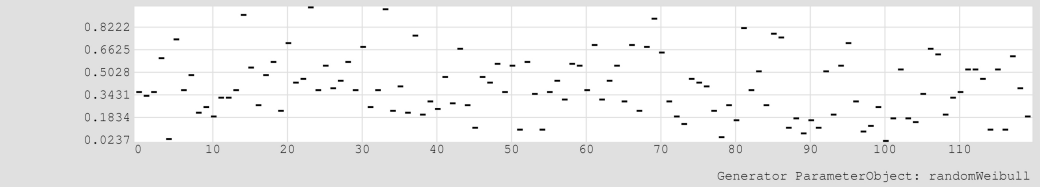
randomWeibull, alpha, beta, min, max

Description: Provides random numbers between 0 and 1 within a Weibull distribution. This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. Note: suggested values: alpha = 0.5, beta = 2.0.

Arguments: (1) name, (2) alpha, (3) beta, (4) min, (5) max

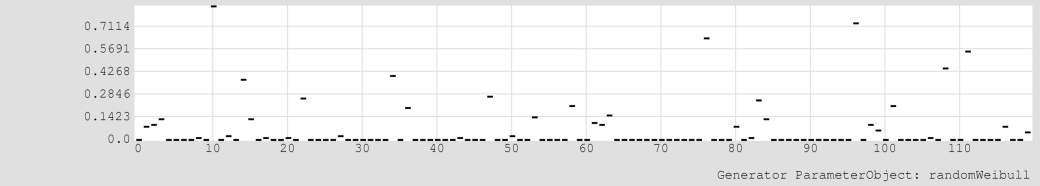
Sample Arguments: `rw, 0.5, 2.0, 0, 1`

Example C-124. randomWeibull Demonstration 1



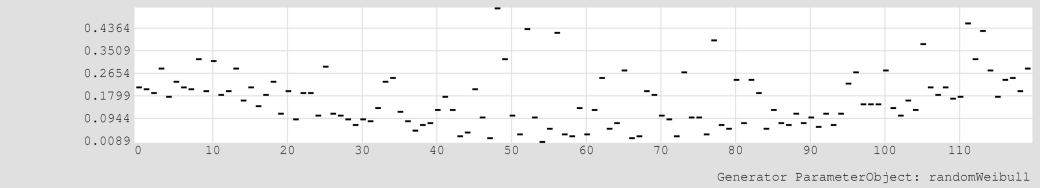

```
randomWeibull, 0.5, 2.0, (constant, 0), (constant, 1)
```

Example C-125. randomWeibull Demonstration 2




```
randomWeibull, 0.9, 0.1, (constant, 0), (constant, 1)
```

Example C-126. randomWeibull Demonstration 3




```
randomWeibull, 0.1, 0.9, (waveSine, event, (constant, 240), 0, (constant, 0),
(constant, 0.4)), (constant, 1)
```

C.1.64. sampleAndHold (sah)

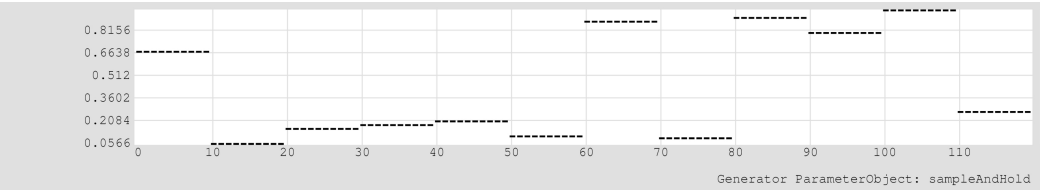
sampleAndHold, comparison, parameterObject, parameterObject, parameterObject

Description: A sample and hold generator. Produces, and continues to produce, a value drawn from the source Generator until the trigger Generator produces a value equal to the threshold Generator. All values are converted to floating-point values.

Arguments: (1) name, (2) comparison, (3) parameterObject {source Generator}, (4) parameterObject {trigger Generator}, (5) parameterObject {trigger threshold Generator}

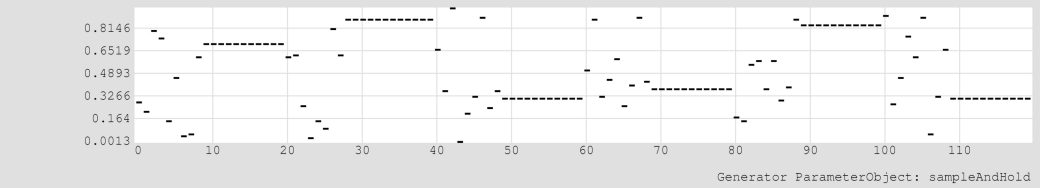
Sample Arguments: `sah, gt, (ru,0,1), (wsd,e,10,0,0,1), (c,0.5)`

Example C-127. sampleAndHold Demonstration 1



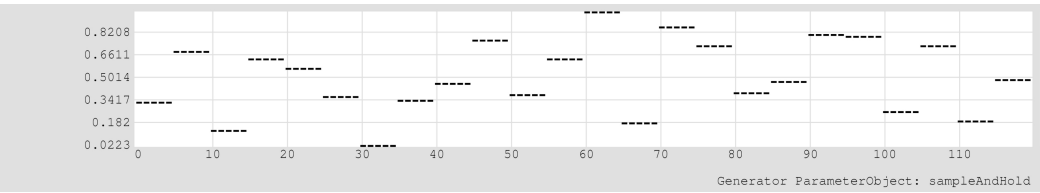

```
sampleAndHold, greaterThan, (randomUniform, (constant, 0), (constant, 1)),
(waveSawDown, event, (constant, 10), 0, (constant, 0), (constant, 1)),
(constant, 0.5)
```

Example C-128. sampleAndHold Demonstration 2




```
sampleAndHold, equal, (randomUniform, (constant, 0), (constant, 1)),
(wavePulse, event, (constant, 20), 0, (constant, 0), (constant, 1)),
(constant, 1)
```

Example C-129. sampleAndHold Demonstration 3




```
sampleAndHold, equal, (randomUniform, (constant, 0), (constant, 1)),
(waveSawDown, event, (constant, 5), 0, (constant, 0), (constant, 1)),
(constant, 1)
```

C.1.65. sieveFunnel (sf)

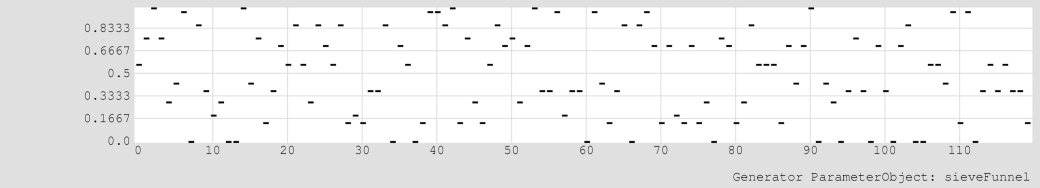
sieveFunnel, logicalString, length, min, max, parameterObject

Description: Using the user-supplied logical string, this Generator produces a Xenakis sieve segment within the z range of zero to one less than the supplied length. Values produced with the fill value Generator ParameterObject are funneled through this sieve: given a fill value, the nearest sieve value is selected and returned. Note: the fill value ParameterObject min and max should be set to 0 and 1.

Arguments: (1) name, (2) logicalString, (3) length, (4) min, (5) max, (6) parameterObject {fill value generator}

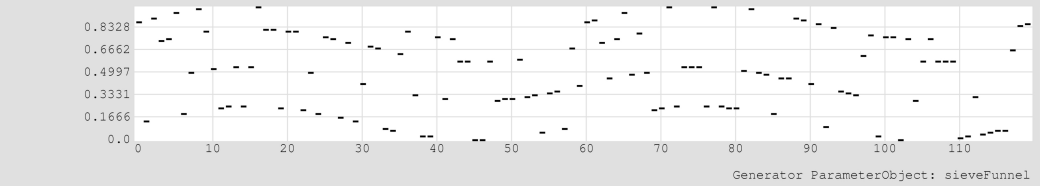
Sample Arguments: `sf, 3|4, 24, 0, 1, (ru,0,1)`

Example C-130. sieveFunnel Demonstration 1



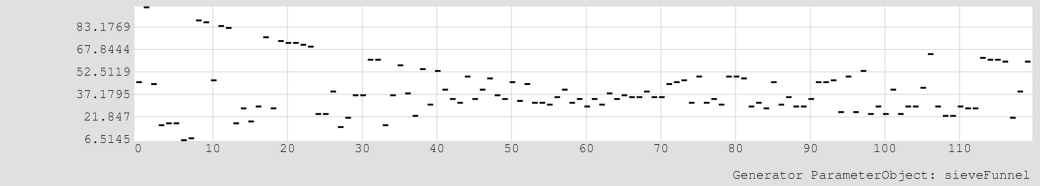

```
sieveFunnel, 3@0|4@0, 24, (constant, 0), (constant, 1), (randomUniform,
(constant, 0), (constant, 1))
```

Example C-131. sieveFunnel Demonstration 2




```
sieveFunnel, 5@0|13@0, 14, (waveSine, event, (constant, 60), 0, (constant, 0),
(constant, 0.25)), (waveSine, event, (constant, 60), 0, (constant, 0.75),
(constant, 1)), (randomUniform, (constant, 0), (constant, 1))
```

Example C-132. sieveFunnel Demonstration 3




```
sieveFunnel, 13@5|13@7|13@11, 20, (accumulator, 0, (waveSine, event,
(constant, 30), 1, (constant, -0.75), (constant, 1.75))), (breakPointPower,
event, loop, ((0,100),(160,20)), 2), (randomBeta, 0.4, 0.3, (constant, 0),
(constant, 1))
```

C.1.66. sieveList (sl)

sieveList, logicalString, zMin, zMax, format, selectionString

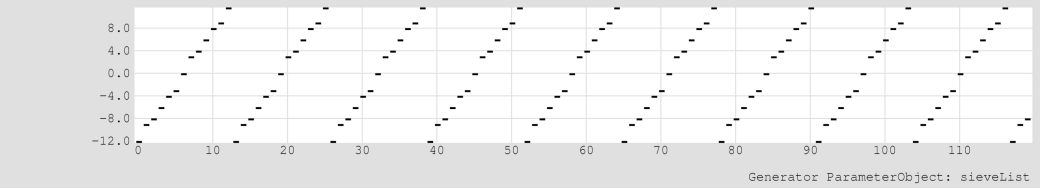
Description: Produces a Xenakis sieve as a raw, variable format sieve segment list. A z is defined by the range of integers from zMin to zMax. Depending on format type, the resulting segment can be

given as an integer, width, unit, or binary segment. Values are chosen from this list using the selector specified by the selectionString argument.

Arguments: (1) name, (2) logicalString, (3) zMin, (4) zMax, (5) format {'integer', 'width', 'unit', 'binary'}, (6) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `sl, 3|4, -12, 12, int, oc`

Example C-133. sieveList Demonstration 1




```
sieveList, 3@0|4@0, -12, 12, integer, orderedCyclic
```

C.1.67. sampleSelect (ss)

sampleSelect, fileNameList, selectionString

Description: Given a list of file names (fileNameList), this Generator provides a complete file path to the file found within either the athenaCL/audio or the user-selected audio directory. Values are chosen from this list using the selector specified by the selectionString argument.

Arguments: (1) name, (2) fileNameList, (3) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `ss, (), rc`

C.1.68. typeFormat (tf)

typeFormat, typeFormatString, parameterObject

Description: Convert the output of any ParameterObject into a different type or display format.

Arguments: (1) name, (2) typeFormatString {'string', 'stringQuote'}, (3) parameterObject {generator}

Sample Arguments: `tf, sq, (bg,rc,(1,3,4,7,-11))`

C.1.69. valuePrime (vp)

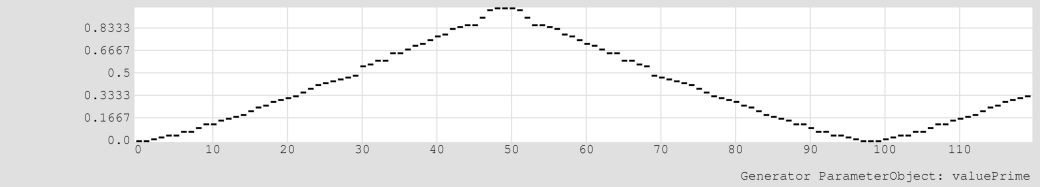
valuePrime, start, length, min, max, selectionString

Description: Produces a segment of prime (pseudoprime) integers defined by a positive or negative start value and a length. The resulting list of values is normalized within the unit interval. Values are chosen from this list using the selector specified by the selectionString argument. After selection, this value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) start, (3) length, (4) min, (5) max, (6) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

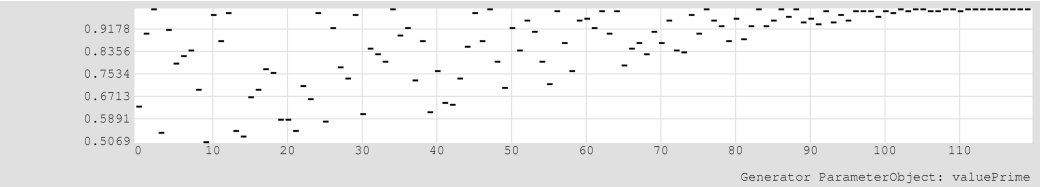
Sample Arguments: `vp, 2, 50, 0, 1, oo`

Example C-134. valuePrime Demonstration 1




```
valuePrime, 2, 50, (constant, 0), (constant, 1), orderedOscillate
```

Example C-135. valuePrime Demonstration 2




```
valuePrime, 100, 20, (breakPointHalfCosine, event, loop, ((0,0.5),(120,1))),
(constant, 1), randomPermutate
```

C.1.70. valueSieve (vs)

valueSieve, logicalString, length, min, max, selectionString

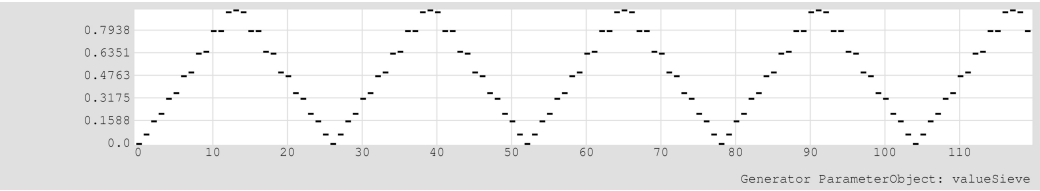
Description: Using the user-supplied logical string, this Generator produces a Xenakis sieve segment within the z range of zero to one less than the supplied length. The resulting list of values is normalized within the unit interval. Values are chosen from this list using the selector specified by

the selectionString argument. After selection, this value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) logicalString, (3) length, (4) min, (5) max, (6) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

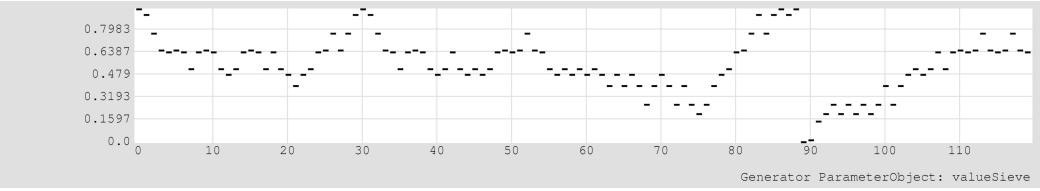
Sample Arguments: `vs, 3&19|4&13@11, 360, 0, 1, oo`

Example C-136. valueSieve Demonstration 1



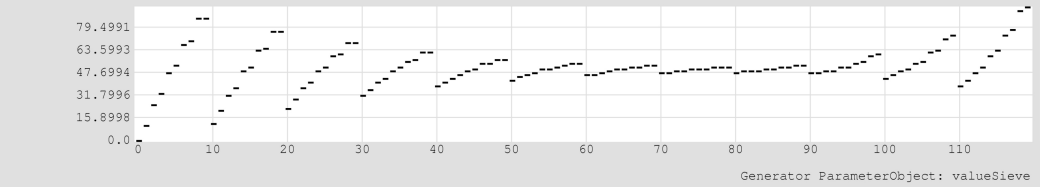

```
valueSieve, 3@0&19@0|4@0&13@11, 360, (constant, 0), (constant, 1),
orderedOscillate
```

Example C-137. valueSieve Demonstration 2



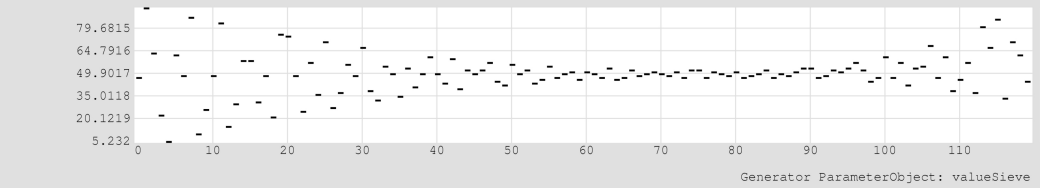

```
valueSieve, 3@0&19@0|4@0&13@11|5@2&15@2, 120, (constant, 0), (constant, 1),
randomWalk
```

Example C-138. valueSieve Demonstration 3




```
valueSieve, 3@0&19@0|4@0&13@11, 240, (breakPointPower, event, single,
((0,0),(80,48),(120,30)), -1.25), (breakPointPower, event, single,
((0,100),(80,52),(120,100)), 1.25), orderedCyclic
```

Example C-139. valueSieve Demonstration 4




```
valueSieve, 3@0&19@0|4@0&13@11, 120, (breakPointPower, event, single,
((0,0),(80,48),(120,30)), -1.25), (breakPointPower, event, single,
((0,100),(80,52),(120,100)), 1.25), randomPermutate
```

C.1.71. waveCosine (wc)

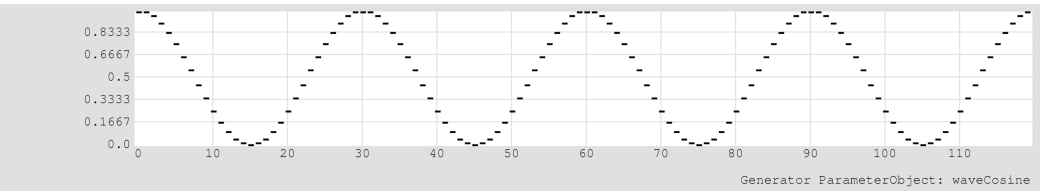
waveCosine, stepString, parameterObject, phase, min, max

Description: Provides cosinusoid oscillation between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

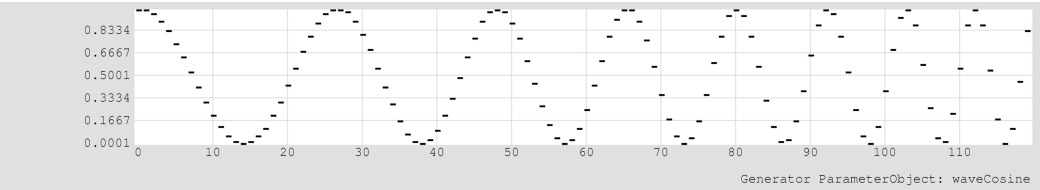
Sample Arguments: `wc, e, 30, 0, 0, 1`

Example C-140. waveCosine Demonstration 1



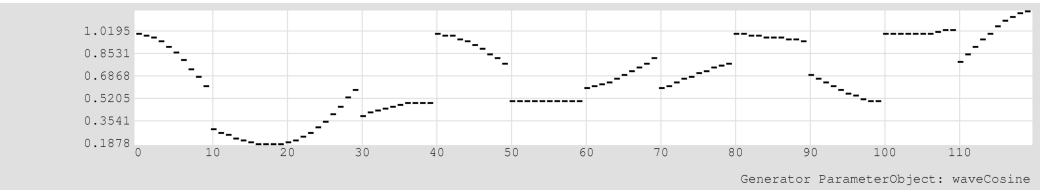

```
waveCosine, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-141. waveCosine Demonstration 2




```
waveCosine, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-142. waveCosine Demonstration 3




```
waveCosine, event, (constant, 40), 0, (wavePulse, event, (constant, 20), 0,
(constant, 1), (constant, 0.5)), (accumulator, 0, (constant, 0.01))
```

C.1.72. waveHalfPeriodCosine (whpc)

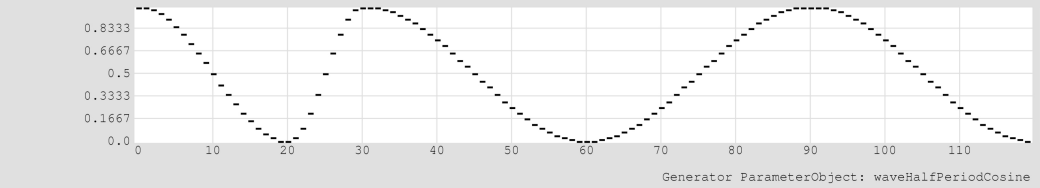
waveHalfPeriodCosine, stepString, parameterObject, phase, min, max

Description: Provides half-period cosinusoid oscillation between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the half-period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Half-period oscillators update seconds/event per cycle only once per half-period, permitting smooth transitons between diverse half-period segments. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

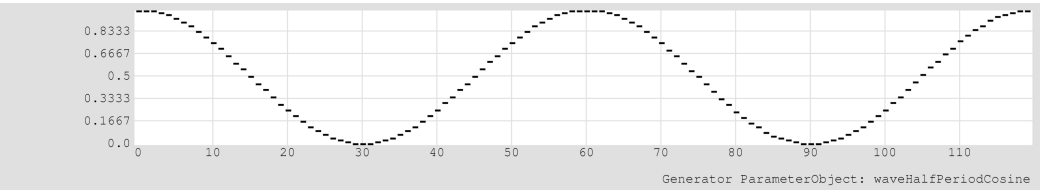
Sample Arguments: `whpc, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-143. waveHalfPeriodCosine Demonstration 1




```
waveHalfPeriodCosine, event, (basketGen, randomChoice, (10,20,30)), 0,
(constant, 0), (constant, 1)
```

Example C-144. waveHalfPeriodCosine Demonstration 2




```
waveHalfPeriodCosine, event, (breakPointLinear, event, loop,
((0,30),(120,15))), 0, (constant, 0), (constant, 1)
```

C.1.73. waveHalfPeriodPulse (whpp)

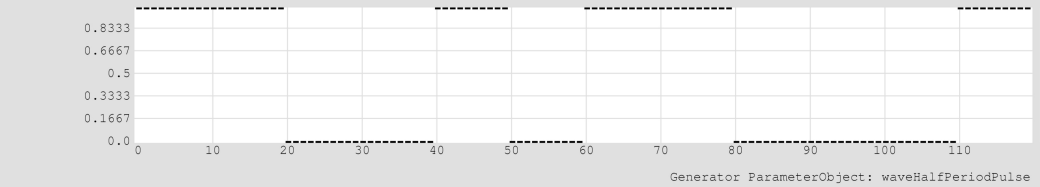
waveHalfPeriodPulse, stepString, parameterObject, phase, min, max

Description: Provides half-period pulse (square) wave oscillation between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the half-period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Half-period oscillators update seconds/event per cycle only once per half-period, permitting smooth transitons between diverse half-period segments. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

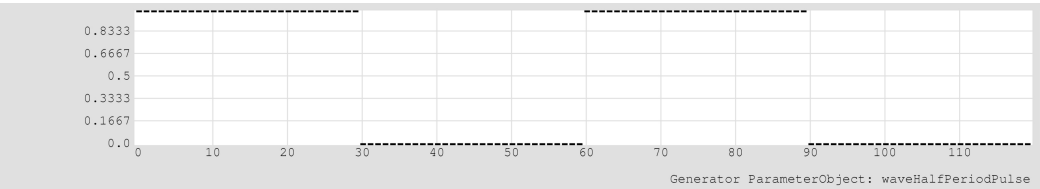
Sample Arguments: `whpp, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-145. waveHalfPeriodPulse Demonstration 1



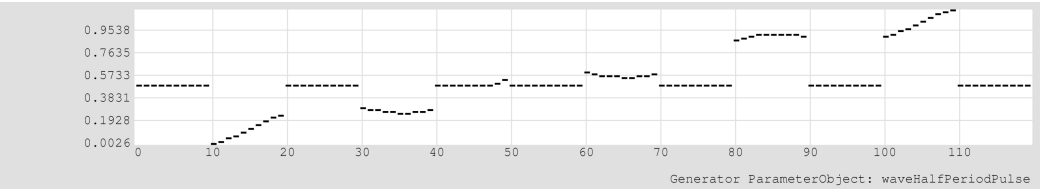

```
waveHalfPeriodPulse, event, (basketGen, randomChoice, (10,20,30)), 0,
(constant, 0), (constant, 1)
```

Example C-146. waveHalfPeriodPulse Demonstration 2



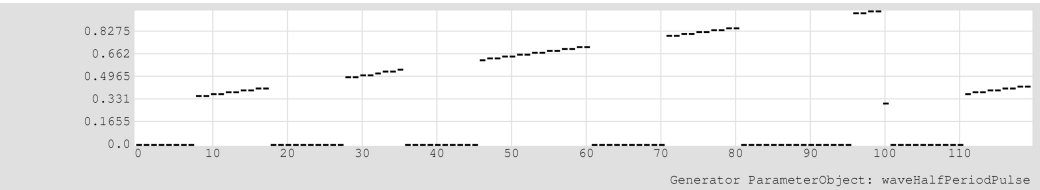

```
waveHalfPeriodPulse, event, (breakPointLinear, event, loop,
((0,30),(120,15))), 0, (constant, 0), (constant, 1)
```

Example C-147. waveHalfPeriodPulse Demonstration 3




```
waveHalfPeriodPulse, event, (constant, 10), 0, (accumulator, 0, (waveSine,
event, (constant, 30), 0.75, (constant, -0.01), (constant, 0.03))), (constant,
0.5)
```

Example C-148. waveHalfPeriodPulse Demonstration 4



```
waveHalfPeriodPulse, event, (basketGen, randomChoice, (15,10,5,8,2)), 0,
(constant, 0), (lineSegment, (constant, 100), (constant, 0.3), (constant, 1))
```

C.1.74. waveHalfPeriodSine (whps)

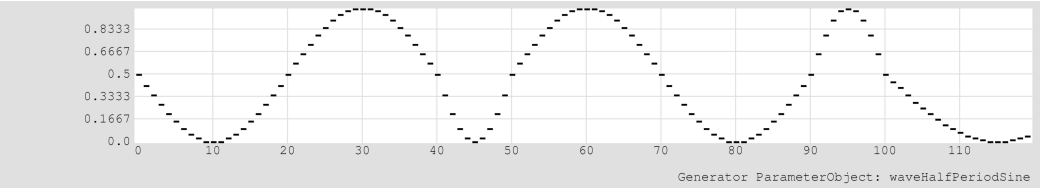
waveHalfPeriodSine, stepString, parameterObject, phase, min, max

Description: Provides half-period sinusoid oscillation between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the half-period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Half-period oscillators update seconds/event per cycle only once per half-period, permitting smooth transitons between diverse half-period segments. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

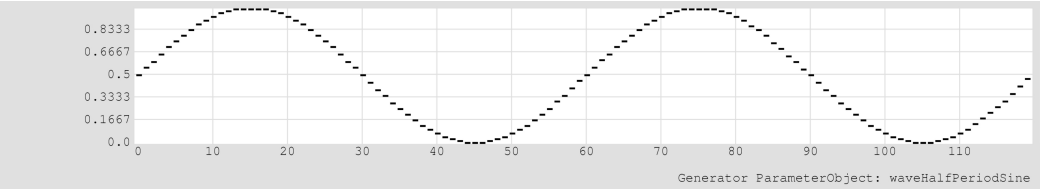
Sample Arguments: `whps, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-149. waveHalfPeriodSine Demonstration 1



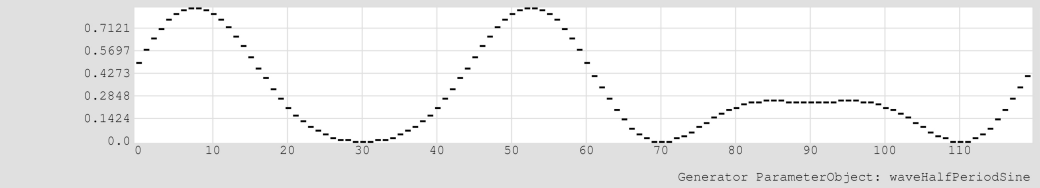

```
waveHalfPeriodSine, event, (basketGen, randomChoice, (10,20,30)), 0,
(constant, 0), (constant, 1)
```

Example C-150. waveHalfPeriodSine Demonstration 2



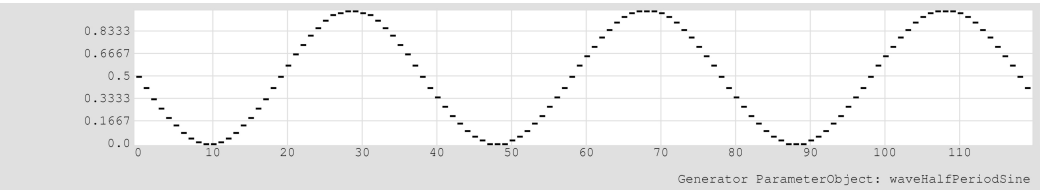

```
waveHalfPeriodSine, event, (breakPointLinear, event, loop, ((0,30),(120,15))),
0, (constant, 0), (constant, 1)
```

Example C-151. waveHalfPeriodSine Demonstration 3




```
waveHalfPeriodSine, event, (constant, 20), 0, (constant, 0), (waveSine, event,
(constant, 60), 0.25, (constant, 0.25), (constant, 1))
```

Example C-152. waveHalfPeriodSine Demonstration 4




```
waveHalfPeriodSine, event, (basketGen, orderedOscillate, (19,19,20,20,20)), 0,
(constant, 0), (constant, 1)
```

C.1.75. waveHalfPeriodTriangle (whpt)

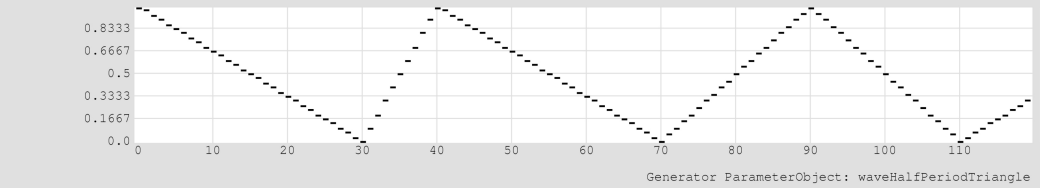
waveHalfPeriodTriangle, stepString, parameterObject, phase, min, max

Description: Provides half-period triangle wave oscillation between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the half-period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Half-period oscillators update seconds/event per cycle only once per half-period, permitting smooth transitons between diverse half-period segments. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

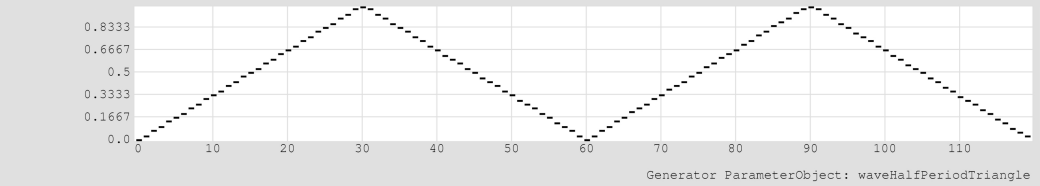
Sample Arguments: `whpt, e, (bg,rc,(10,20,30)), 0, 0, 1`

Example C-153. waveHalfPeriodTriangle Demonstration 1




```
waveHalfPeriodTriangle, event, (basketGen, randomChoice, (10,20,30)), 0,
(constant, 0), (constant, 1)
```

Example C-154. waveHalfPeriodTriangle Demonstration 2




```
waveHalfPeriodTriangle, event, (breakPointLinear, event, loop,
((0,30),(120,15))), 0, (constant, 0), (constant, 1)
```

C.1.76. wavePulse (wp)

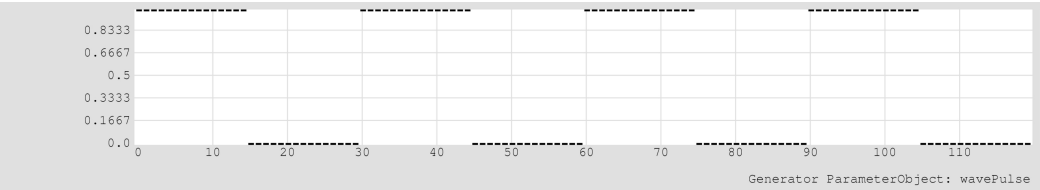
wavePulse, stepString, parameterObject, phase, min, max

Description: Provides a pulse (square) wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

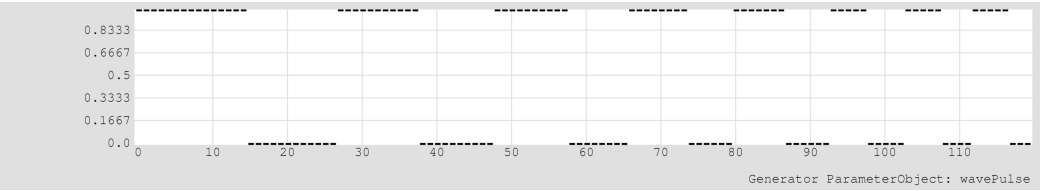
Sample Arguments: `wp, e, 30, 0, 0, 1`

Example C-155. wavePulse Demonstration 1



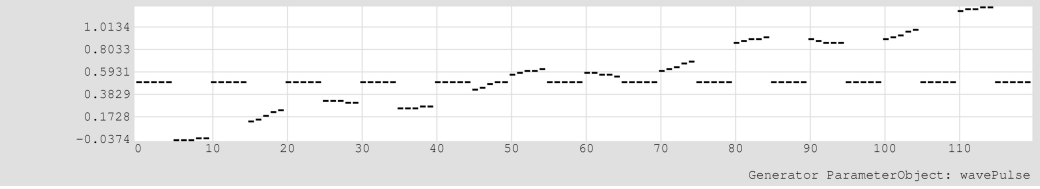

```
wavePulse, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-156. wavePulse Demonstration 2




```
wavePulse, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-157. wavePulse Demonstration 3




```
wavePulse, event, (constant, 10), 0, (accumulator, 0, (waveSine, event,
(constant, 30), 0.75, (constant, -0.01), (constant, 0.03))), (constant, 0.5)
```

C.1.77. wavePowerDown (wpd)

wavePowerDown, stepString, parameterObject, phase, exponent, min, max

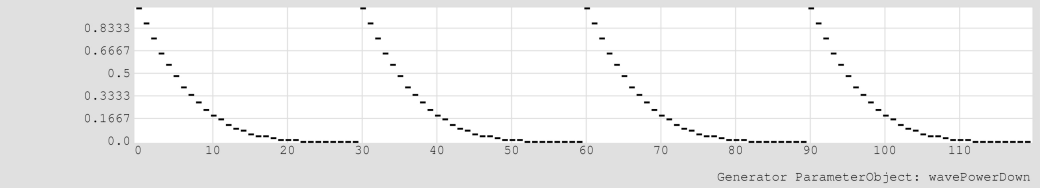
Description: Provides a power down wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is

specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) exponent, (6) min, (7) max

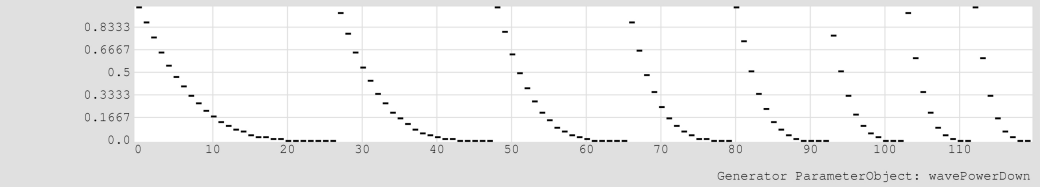
Sample Arguments: `wpd, e, 30, 0, 2, 0, 1`

Example C-158. wavePowerDown Demonstration 1



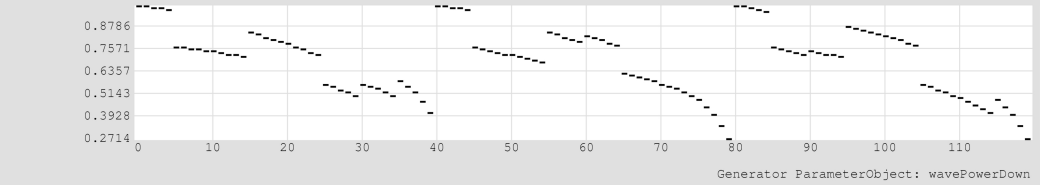

```
wavePowerDown, event, (constant, 30), 0, 2, (constant, 0), (constant, 1)
```

Example C-159. wavePowerDown Demonstration 2




```
wavePowerDown, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
2, (constant, 0), (constant, 1)
```

Example C-160. wavePowerDown Demonstration 3




```
wavePowerDown, event, (constant, 40), 0, -1.5, (wavePulse, event, (constant,
30), 0, (constant, 0), (constant, 0.2)), (wavePulse, event, (constant, 20),
0.25, (constant, 1), (constant, 0.8))
```

C.1.78. wavePowerUp (wpu)

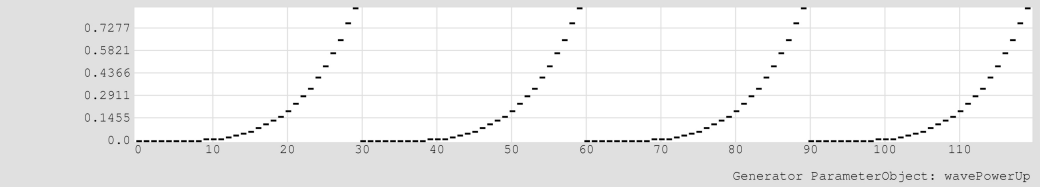
wavePowerUp, stepString, parameterObject, phase, exponent, min, max

Description: Provides a power up wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) exponent, (6) min, (7) max

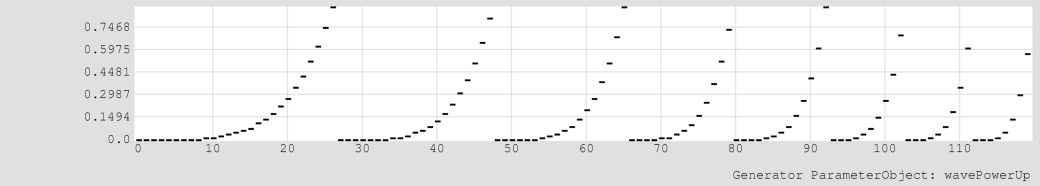
Sample Arguments: `wpu, e, 30, 0, 2, 0, 1`

Example C-161. wavePowerUp Demonstration 1



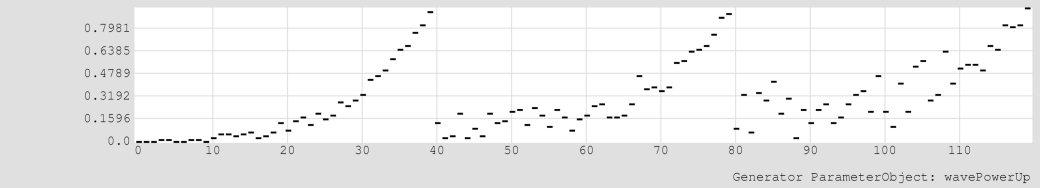

```
wavePowerUp, event, (constant, 30), 0, 2, (constant, 0), (constant, 1)
```

Example C-162. wavePowerUp Demonstration 2




```
wavePowerUp, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0, 2,
(constant, 0), (constant, 1)
```

Example C-163. wavePowerUp Demonstration 3




```
wavePowerUp, event, (constant, 40), 0, 2, (randomUniform, (constant, 0),
(accumulator, 0, (constant, 0.005))), (constant, 1)
```

C.1.79. waveSine (ws)

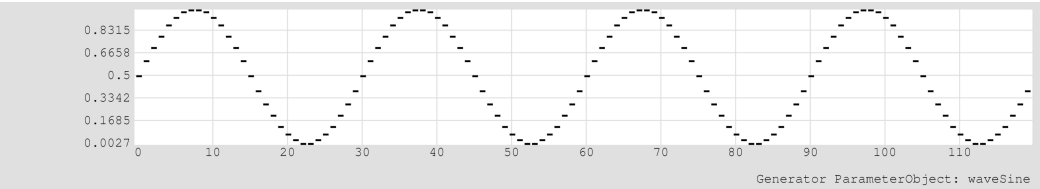
waveSine, stepString, parameterObject, phase, min, max

Description: Provides sinusoid oscillation between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

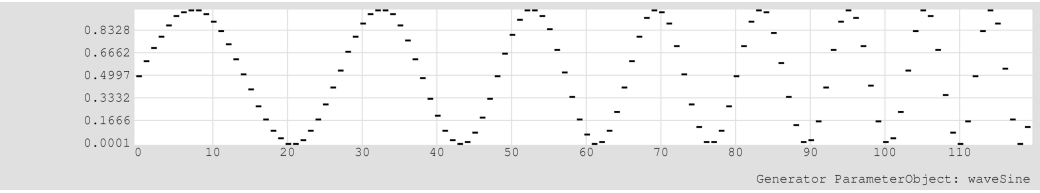
Sample Arguments: `ws, e, 30, 0, 0, 1`

Example C-164. waveSine Demonstration 1



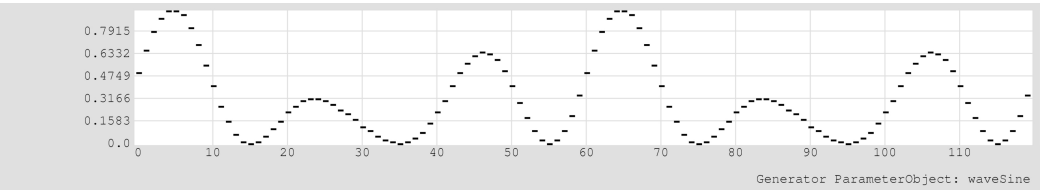

```
waveSine, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-165. waveSine Demonstration 2



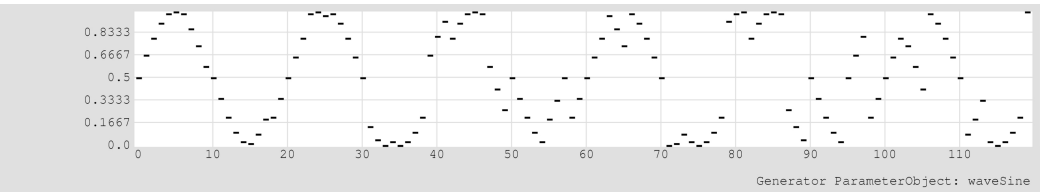

```
waveSine, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-166. waveSine Demonstration 3




```
waveSine, event, (constant, 20), 0, (constant, 0), (waveSine, event,
(constant, 60), 0.25, (constant, 0.25), (constant, 1))
```

Example C-167. waveSine Demonstration 4




```
waveSine, event, (basketGen, orderedOscillate, (19,19,20,20,20)), 0,
(constant, 0), (constant, 1)
```

C.1.80. waveSawDown (wsd)

waveSawDown, stepString, parameterObject, phase, min, max

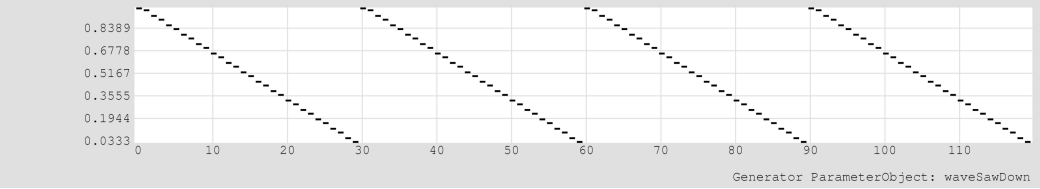
Description: Provides a saw-down wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as

a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

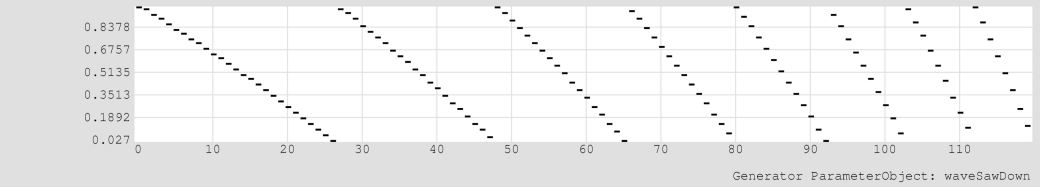
Sample Arguments: `wsd, e, 30, 0, 0, 1`

Example C-168. waveSawDown Demonstration 1



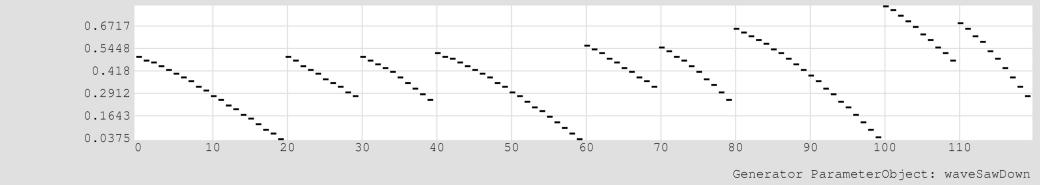

```
waveSawDown, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-169. waveSawDown Demonstration 2




```
waveSawDown, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-170. waveSawDown Demonstration 3




```
waveSawDown, event, (constant, 20), 0, (wavePowerUp, event, (constant, 120),
0, 1.5, (constant, 0.5), (constant, 1)), (wavePowerDown, event, (constant,
40), 0.25, 1.5, (constant, 0.5), (constant, 0))
```

C.1.81. waveSawUp (wsu)

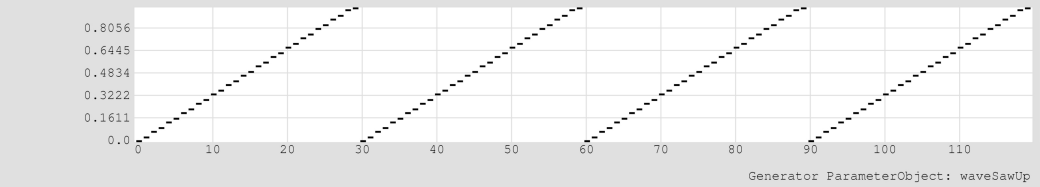
waveSawUp, stepString, parameterObject, phase, min, max

Description: Provides a saw-up wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

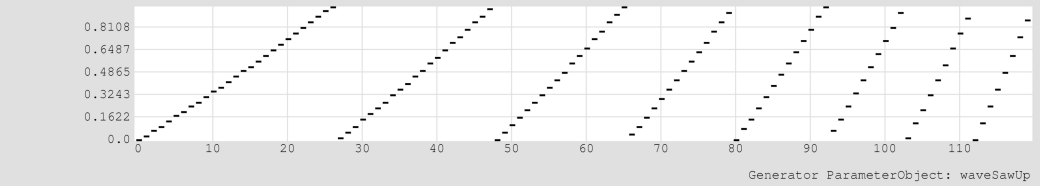
Sample Arguments: `wsu, e, 30, 0, 0, 1`

Example C-171. waveSawUp Demonstration 1



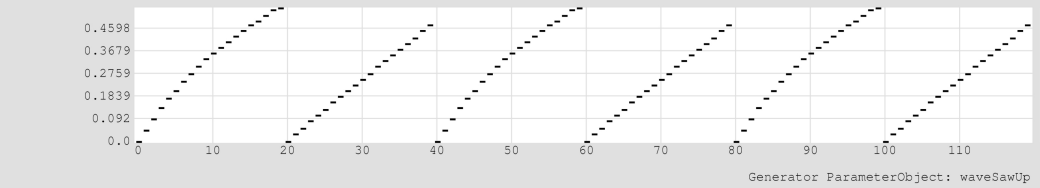

```
waveSawUp, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-172. waveSawUp Demonstration 2




```
waveSawUp, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-173. waveSawUp Demonstration 3




```
waveSawUp, event, (constant, 20), 0, (wavePowerDown, event, (constant, 40), 0,
1.5, (constant, 1), (constant, 0.5)), (constant, 0)
```

C.1.82. waveTriangle (wt)

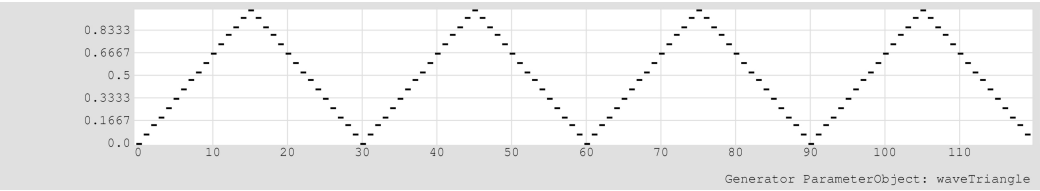
waveTriangle, stepString, parameterObject, phase, min, max

Description: Provides a triangle wave between 0 and 1 at a rate given in either time or events per period. Depending on the stepString argument, the period rate (frequency) may be specified in spc (seconds per cycle) or eps (events per cycle). This value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects. The phase argument is specified as a value between 0 and 1. Note: conventional cycles per second (cps or Hz) are not used for frequency.

Arguments: (1) name, (2) stepString {'event', 'time'}, (3) parameterObject {secPerCycle}, (4) phase, (5) min, (6) max

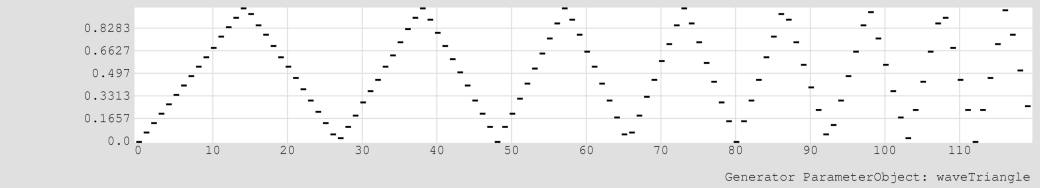
Sample Arguments: `wt, e, 30, 0, 0, 1`

Example C-174. waveTriangle Demonstration 1



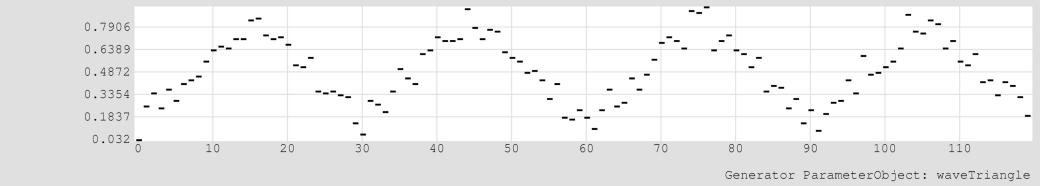

```
waveTriangle, event, (constant, 30), 0, (constant, 0), (constant, 1)
```

Example C-175. waveTriangle Demonstration 2




```
waveTriangle, event, (breakPointLinear, event, loop, ((0,30),(120,15))), 0,
(constant, 0), (constant, 1)
```

Example C-176. waveTriangle Demonstration 3




```
waveTriangle, event, (constant, 30), 0, (randomUniform, (constant, 0),
(constant, 0.3)), (randomUniform, (constant, 0.7), (constant, 1))
```

C.2. Rhythm ParameterObjects

C.2.1. binaryAccent (ba)

binaryAccent, pulseList

Description: Deploys two Pulses based on event pitch selection. Every instance of the first pitch in the current set of a Texture's Path is assigned the second Pulse; all other pitches are assigned the first Pulse. Amplitude values of events that have been assigned the second pulse are increased by a scaling function.

Arguments: (1) name, (2) pulseList {a list of Pulse notations}

Sample Arguments: `ba, ((3,1,1),(3,2,1))`

C.2.2. convertSecond (cs)

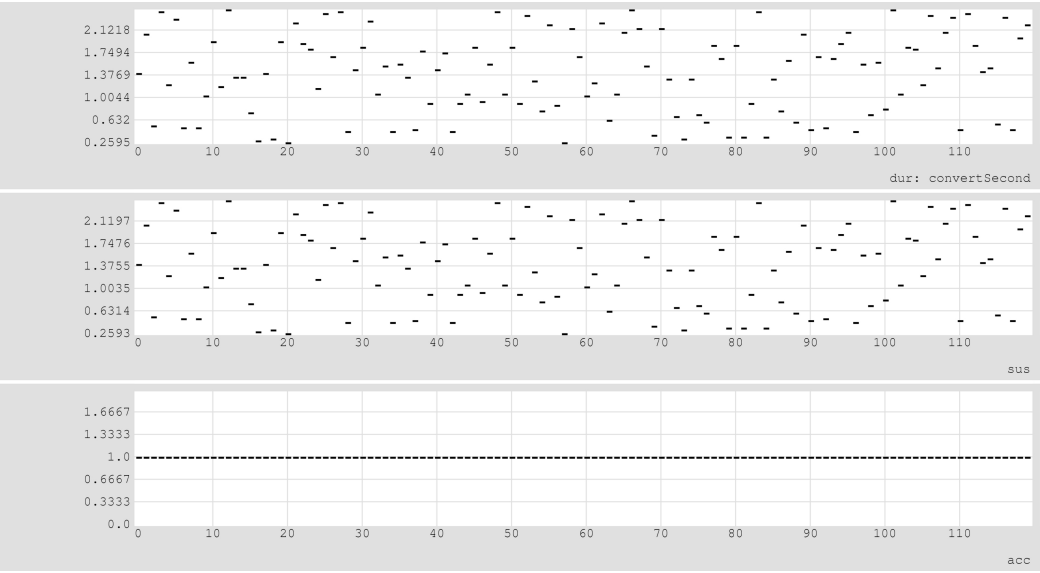
convertSecond, parameterObject

Description: Allows the use of a Generator ParameterObject to create rhythm durations. Values from this ParameterObject are interpreted as equal Pulse duration and sustain values in seconds. Accent values are fixed at 1. Note: when using this Rhythm Generator, tempo information (bpm) has no effect on event timing.

Arguments: (1) name, (2) parameterObject {duration values in seconds}

Sample Arguments: `cs, (ru,0.25,2.5)`

Example C-177. convertSecond Demonstration 1




```
convertSecond, (randomUniform, (constant, 0.25), (constant, 2.5))
```

C.2.3. convertSecondTriple (cst)

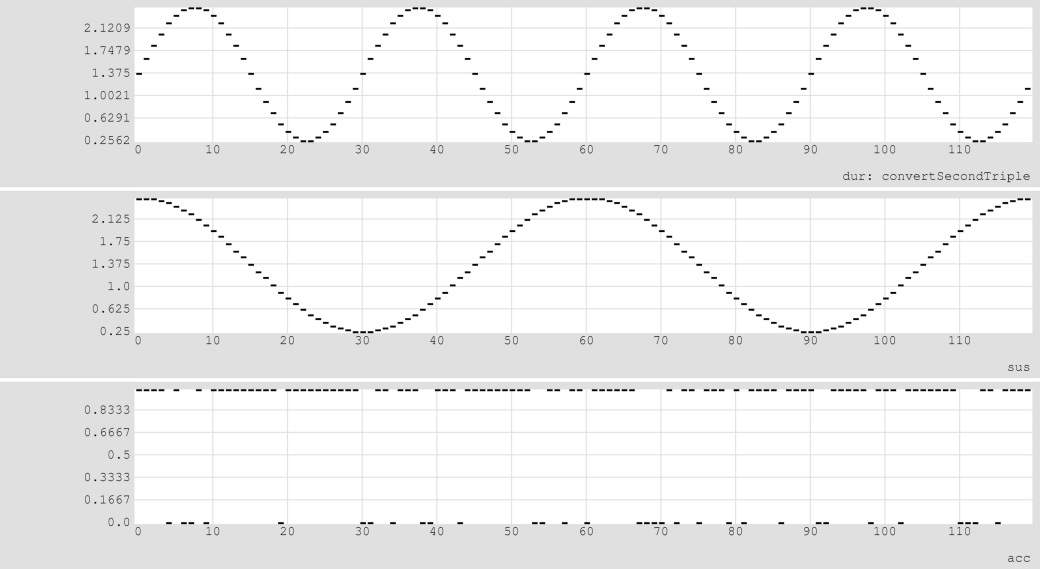
convertSecondTriple, parameterObject, parameterObject, parameterObject

Description: Allows the use of three Generator ParameterObjects to directly specify duration, sustain, and accent values. Values for duration and sustain are interpreted as values in seconds. Accent values must be between 0 and 1, where 0 is a measured silence and 1 is a fully sounding event. Note: when using this Rhythm Generator, tempo information (bpm) has no effect on event timing.

Arguments: (1) name, (2) parameterObject {duration values in seconds}, (3) parameterObject {sustain values in seconds}, (4) parameterObject {accent values between 0 and 1}

Sample Arguments: `cst, (ws,e,30,0,0.25,2.5), (ws,e,60,0.25,0.25,2.5), (bg,rc,(0,1,1,1))`

Example C-178. convertSecondTriple Demonstration 1




```
convertSecondTriple, (waveSine, event, (constant, 30), 0, (constant, 0.25),
(constant, 2.5)), (waveSine, event, (constant, 60), 0.25, (constant, 0.25),
(constant, 2.5)), (basketGen, randomChoice, (0,1,1,1))
```

C.2.4. gaRhythm (gr)

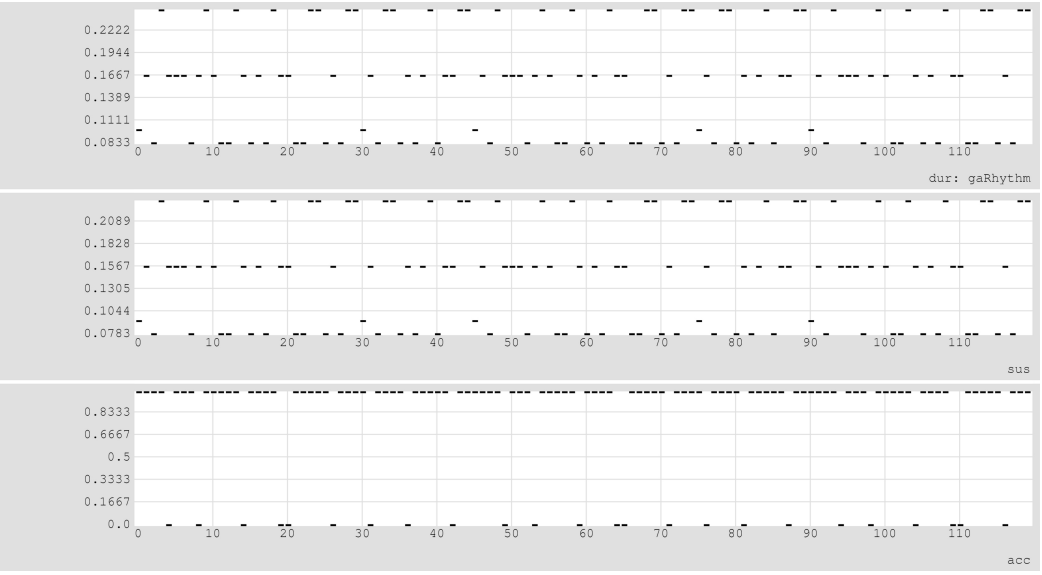
gaRhythm, pulseList, crossover, mutation, elitism, selectionString, populationSize

Description: Uses a genetic algorithm to create rhythmic variants of a source rhythm. Crossover rate is a percentage, expressed within the unit interval, of genetic crossings that undergo crossover. Mutation rate is a percentage, expressed within the unit interval, of genetic crossings that undergo mutation. Elitism rate is a percentage, expressed within the unit interval, of the entire population that passes into the next population unchanged. All rhythms in the final population are added to a list. Pulses are chosen from this list using the selector specified by the control argument.

Arguments: (1) name, (2) pulseList {a list of Pulse notations}, (3) crossover, (4) mutation, (5) elitism, (6) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}, (7) populationSize

Sample Arguments: `gr, ((3,1,1),(3,1,1),(6,1,1),(6,3,1),(3,1,0)), 0.7, 0.06, 0.01, oc, 20`

Example C-179. gaRhythm Demonstration 1




```
gaRhythm, ((3,1,+),(3,1,+),(6,1,+),(6,3,+),(3,1,o)), 0.7, 0.06, 0.01,
orderedCyclic, 20
```

C.2.5. iterateRhythmGroup (irg)

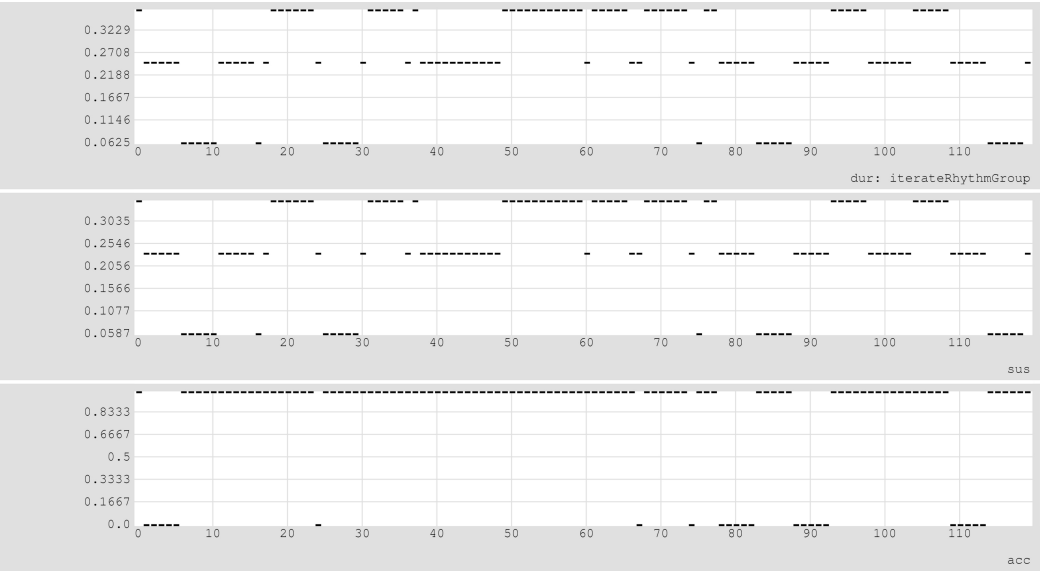
iterateRhythmGroup, parameterObject, parameterObject

Description: Allows the output of a source Rhythm ParameterObject to be grouped (a value is held and repeated a certain number of times), to be skipped (a number of values are generated and discarded), or to be bypassed. A numeric value from a control ParameterObject is used to determine the source ParameterObject behavior. A positive value (rounded to the nearest integer) will cause the value provided by the source ParameterObject to be repeated that many times. After output of these values, a new control value is generated. A negative value (rounded to the nearest integer) will cause that many number of values to be generated and discarded from the source ParameterObject, and force the selection of a new control value. A value of 0 is treated as a bypass, and forces the selection of a new control value. Note: if the control ParameterObject fails to produce positive values after many attempts, a value will be automatically generated from the selected ParameterObject.

Arguments: (1) name, (2) parameterObject {source Rhythm Generator}, (3) parameterObject {group or skip control Generator}

Sample Arguments: `irg, (1,((4,3,1),(4,3,1),(4,2,0),(8,1,1),(4,2,1),(4,2,1)),oc), (bg,rc,(-3,1,-1,5))`

Example C-180. iterateRhythmGroup Demonstration 1




```
iterateRhythmGroup, (loop, ((4,3,+),(4,3,+),(4,2,o),(8,1,+),(4,2,+),(4,2,+)),
orderedCyclic), (basketGen, randomChoice, (-3,1,-1,5))
```

C.2.6. iterateRhythmHold (irh)

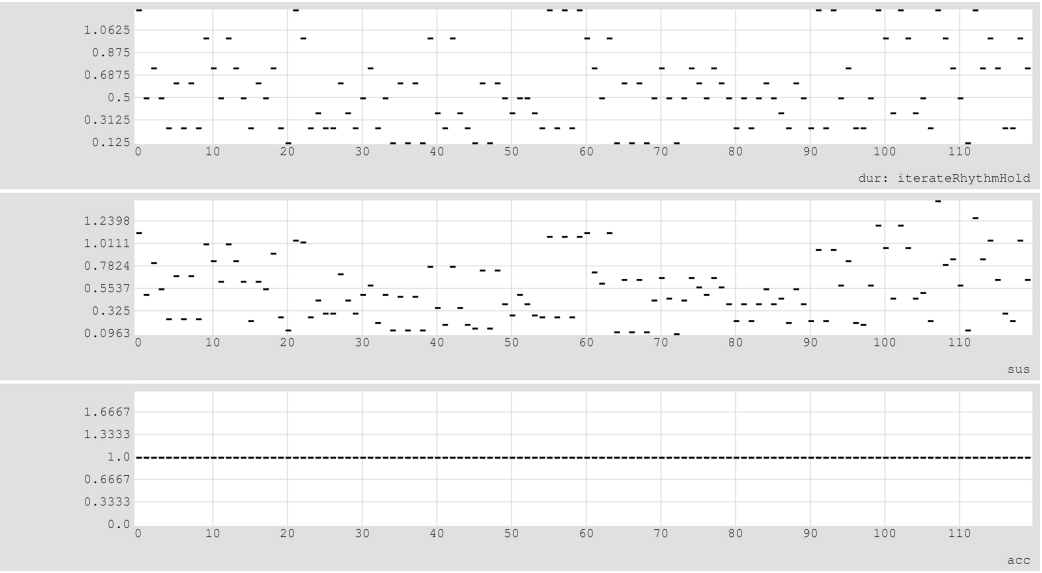
iterateRhythmHold, parameterObject, parameterObject, parameterObject, selectionString

Description: Allows a variable number of outputs from a source Rhythm ParameterObject, collected and stored in a list, to be held and selected. Values are chosen from this list using the selector specified by the selectionString argument. A numeric value from a size ParameterObject is used to determine how many values are drawn from the source ParameterObject. A numeric value from a refresh count ParameterObject is used to determine how many events must pass before a new size value is drawn and the source ParameterObject is used to refill the stored list. A refresh value of zero, once encountered, will prohibit any further changes to the stored list. Note: if the size ParameterObject fails to produce a non-zero value for the first event, an alternative count value will be assigned.

Arguments: (1) name, (2) parameterObject {source Rhythm Generator}, (3) parameterObject {size Generator}, (4) parameterObject {refresh count Generator}, (5) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `irh, (pt,(bg,rc,(4,2)),(bg,oc,(5,4,3,2,1)),(c,1),(ru,0.75,1.25)), (bg,rc,(2,3,4)), (bg,oc,(4,5,6)), oc`

Example C-181. iterateRhythmHold Demonstration 1




```
iterateRhythmHold, (pulseTriple, (basketGen, randomChoice, (4,2)), (basketGen,
orderedCyclic, (5,4,3,2,1)), (constant, 1), (randomUniform, (constant, 0.75),
(constant, 1.25))), (basketGen, randomChoice, (2,3,4)), (basketGen,
orderedCyclic, (4,5,6)), orderedCyclic
```

C.2.7. iterateRhythmWindow (irw)

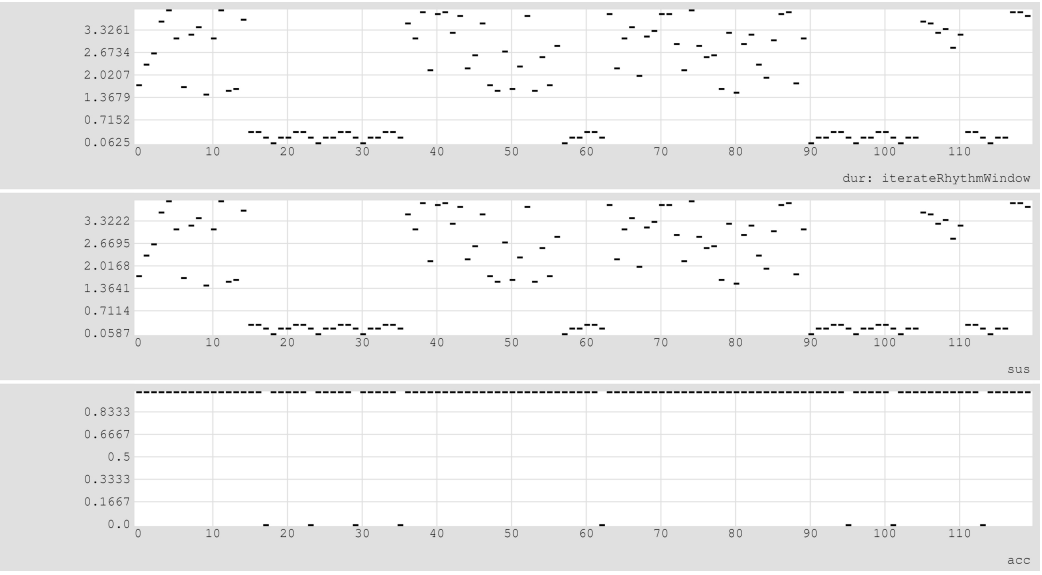
iterateRhythmWindow, parameterObjectList, parameterObject, selectionString

Description: Allows a Rhythm ParameterObject, selected from a list of Rhythm ParameterObjects, to generate values, to skip values (a number of values are generated and discarded), or to bypass value generation. A numeric value from a control ParameterObject is used to determine the selected ParameterObject behavior. A positive value (rounded to the nearest integer) will cause the selected ParameterObject to produce that many new values. After output of these values, a new ParameterObject is selected. A negative value (rounded to the nearest integer) will cause the selected ParameterObject to generate and discard that many values, and force the selection of a new ParameterObject. A value equal to 0 is treated as a bypass, and forces the selection of a new ParameterObject. ParameterObject selection is determined with a string argument for a selection method. Note: if the control ParameterObject fails to produce positive values after many attempts, a value will be automatically generated from the selected ParameterObject.

Arguments: (1) name, (2) parameterObjectList {a list of Rhythm Generators}, (3) parameterObject {generate or skip control Generator}, (4) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `irw,`
`((l,((4,3,1),(4,3,1),(4,2,0),(8,1,1),(4,2,1),(4,2,1)),oc),(cs,(ru,1.5,4))), (bg,rc,(-3,6,-1,15)), oc`

Example C-182. iterateRhythmWindow Demonstration 1




```
iterateRhythmWindow, ((loop,
((4,3,+),(4,3,+),(4,2,o),(8,1,+),(4,2,+),(4,2,+)), orderedCyclic),
(convertSecond, (randomUniform, (constant, 1.5), (constant, 4)))), (basketGen,
randomChoice, (-3,6,-1,15)), orderedCyclic
```

C.2.8. loop (l)

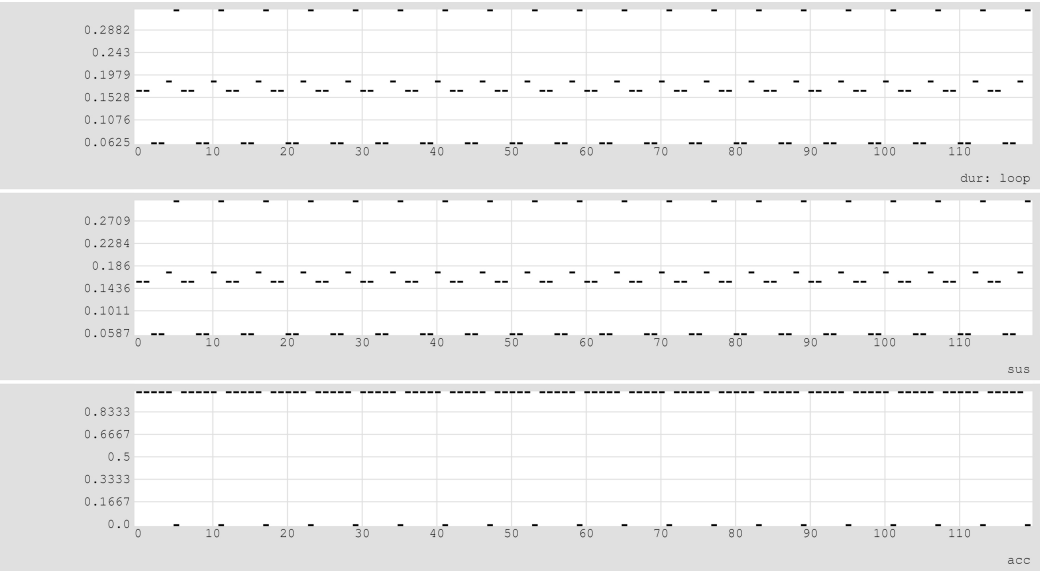
loop, pulseList, selectionString

Description: Deploys a fixed list of rhythms. Pulses are chosen from this list using the selector specified by the selectionString argument.

Arguments: (1) name, (2) pulseList {a list of Pulse notations}, (3) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `l, ((3,1,1),(3,1,1),(8,1,1),(8,1,1),(8,3,1),(3,2,0)), oc`

Example C-183. loop Demonstration 1




```
loop, ((3,1,+),(3,1,+),(8,1,+),(8,1,+),(8,3,+),(3,2,o)), orderedCyclic
```

C.2.9. markovPulse (mp)

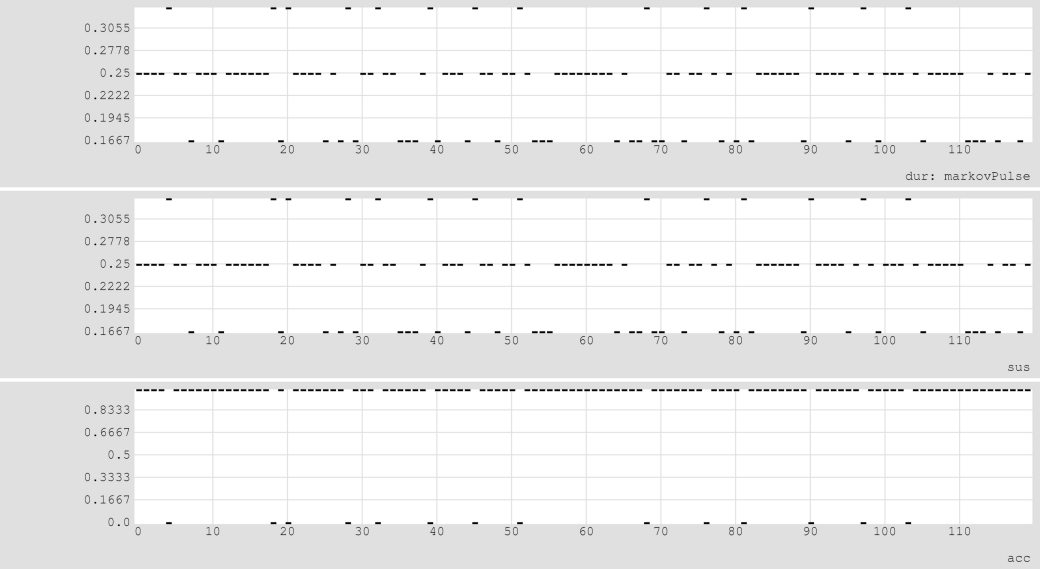
markovPulse, transitionString, parameterObject

Description: Produces Pulse sequences by means of a Markov transition string specification and a dynamic transition order generator. The Markov transition string must define symbols that specify valid Pulses. Markov transition order is specified by a ParameterObject that produces values between 0 and the maximum order available in the Markov transition string. If generated-orders are greater than those available, the largest available transition order will be used. Floating-point order values are treated as probabilistic weightings: for example, a transition of 1.5 offers equal probability of first or second order selection.

Arguments: (1) name, (2) transitionString, (3) parameterObject {order value}

Sample Arguments: `mp, a{3,1,1}b{2,1,1}c{3,2,0}:{a=3|b=4|c=1}, (c,0)`

Example C-184. markovPulse Demonstration 1




```
markovPulse, a{3,1,1}b{2,1,1}c{3,2,0}:{a=3|b=4|c=1}, (constant, 0)
```

C.2.10. markovRhythmAnalysis (mra)

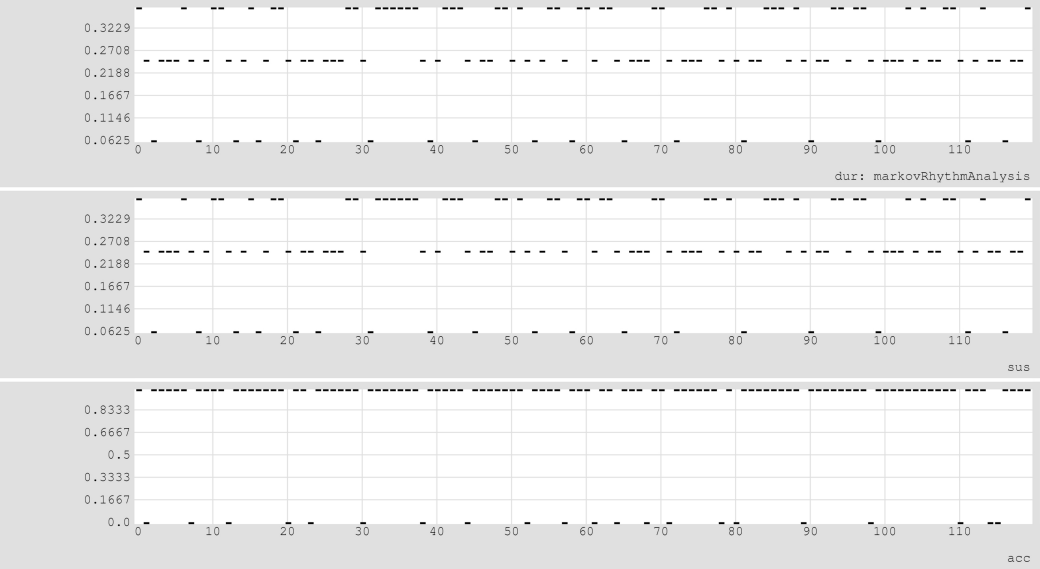
markovRhythmAnalysis, parameterObject, pulseCount, maxAnalysisOrder, parameterObject

Description: Produces Pulse sequences by means of a Markov analysis of a rhythm provided by a source Rhythm Generator ParameterObject; the analysis of these values is used with a dynamic transition order Generator to produce new values. The number of values drawn from the source Rhythm Generator is specified with the pulseCount argument. The maximum order of analysis is specified with the maxAnalysisOrder argument. Markov transition order is specified by a ParameterObject that produces values between 0 and the maximum order available in the Markov transition string. If generated-orders are greater than those available, the largest available transition order will be used. Floating-point order values are treated as probabilistic weightings: for example, a transition of 1.5 offers equal probability of first or second order selection.

Arguments: (1) name, (2) parameterObject {source Rhythm Generator}, (3) pulseCount, (4) maxAnalysisOrder, (5) parameterObject {output order value}

Sample Arguments: `mra, (1,((4,3,1),(4,3,1),(4,2,0),(8,1,1),(4,2,1),(4,2,1)),oc), 12, 2, (cg,u,0,2,0.25)`

Example C-185. markovRhythmAnalysis Demonstration 1




```
markovRhythmAnalysis, (loop,
((4,3,+),(4,3,+),(4,2,o),(8,1,+),(4,2,+),(4,2,+)), orderedCyclic), 12, 2,
(cyclicGen, up, 0, 2, 0.25)
```

C.2.11. pulseSieve (ps)

pulseSieve, logicalString, sieveLength, pulse, selectionString, articulationString

Description: Using the user-supplied logical string, this Generator produces a Xenakis sieve segment within the z range of zero to one less than the supplied length. This sieve, as a binary or width segment, is interpreted as a pulse list. The length of each pulse and the presence of rests are determined by the user-provided Pulse object and the articulationString argument. An articulationString of 'attack' creates durations equal to the provided Pulse for every non-zero binary sieve segment value; an articulationString of 'sustain' creates durations equal to the Pulse times the sieve segment width, or the duration of all following rests until the next Pulse. Values are chosen from this list using the selector specified by the selectionString argument.

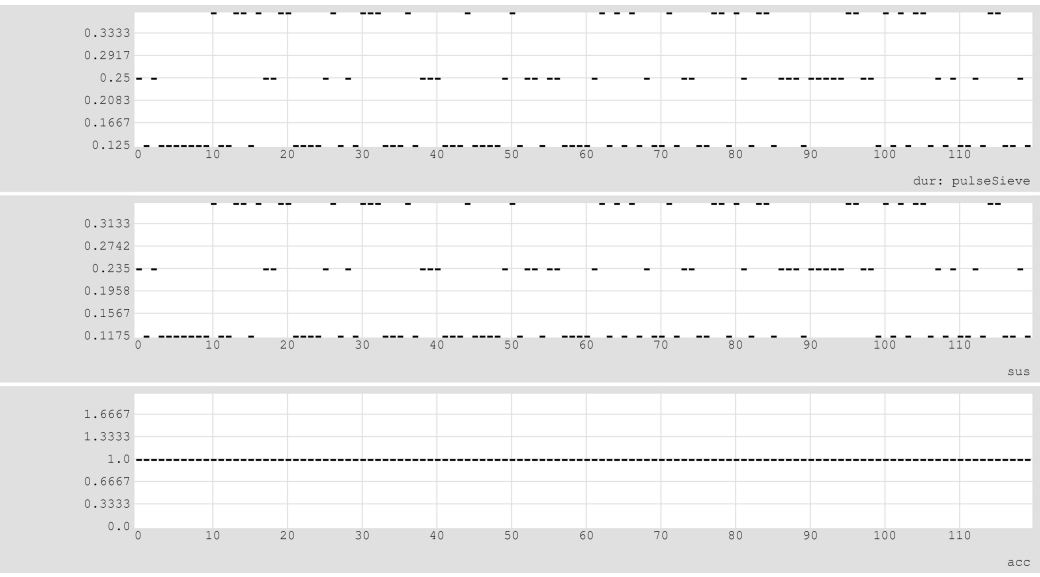
Arguments: (1) name, (2) logicalString, (3) sieveLength, (4) pulse {a single Pulse notation}, (5) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}, (6) articulationString {'attack', 'sustain'}

Sample Arguments: `ps, 3|4|5@2, 60, (3,1,1), oc, a`

Example C-186. pulseSieve Demonstration 1


```
pulseSieve, 3@0|4@0|5@2, 60, (3,1,+), orderedCyclic, attack
```

Example C-187. pulseSieve Demonstration 2




```
pulseSieve, 3@0|4@0|5@2, 60, (4,1,+), randomChoice, sustain
```

C.2.12. pulseTriple (pt)

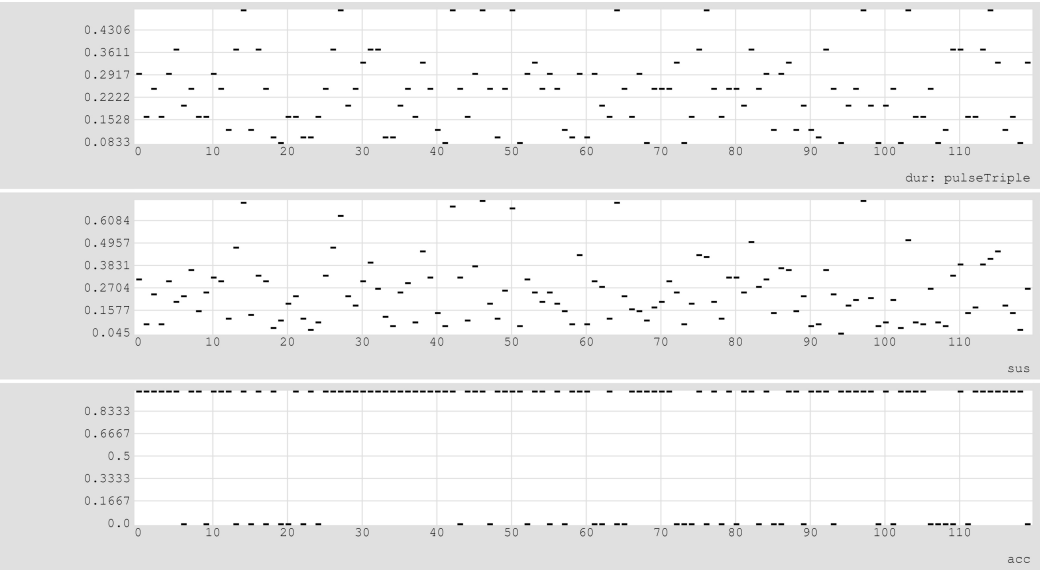
pulseTriple, parameterObject, parameterObject, parameterObject, parameterObject

Description: Produces Pulse sequences with four Generator ParameterObjects that directly specify Pulse triple values and a sustain scalar. The Generators specify Pulse divisor, multiplier, accent, and sustain scalar. Floating-point divisor and multiplier values are treated as probabilistic weightings. Note: divisor and multiplier values of 0 are not permitted and are replaced by 1; the absolute value is taken of all values.

Arguments: (1) name, (2) parameterObject {pulse divisor}, (3) parameterObject {pulse multiplier}, (4) parameterObject {accent value between 0 and 1}, (5) parameterObject {sustain scalar greater than 0}

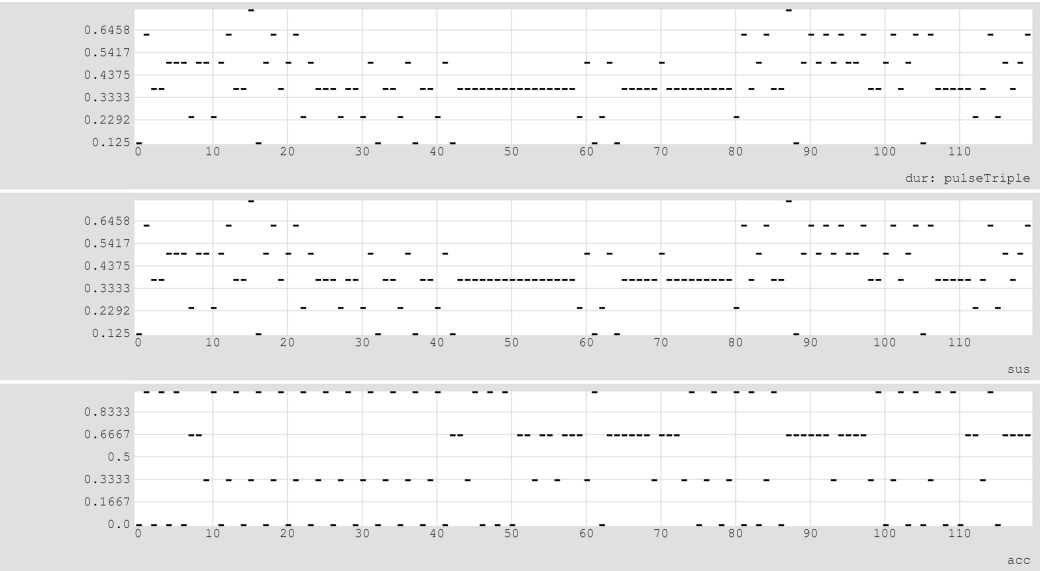
Sample Arguments: `pt, (bg,rc,(6,5,4,3)), (bg,rc,(1,2,3)), (bg,rc,(1,1,1,0)), (ru,0.5,1.5)`

Example C-188. pulseTriple Demonstration 1




```
pulseTriple, (basketGen, randomChoice, (6,5,4,3)), (basketGen, randomChoice,
(1,2,3)), (basketGen, randomChoice, (1,1,1,0)), (randomUniform, (constant,
0.5), (constant, 1.5))
```

Example C-189. pulseTriple Demonstration 2




```
pulseTriple, (constant, 4), (caList,
f{s}k{2}r{1}i{center}x{81}y{120}w{6}c{-2}s{0}, (constant, 109), (constant,
0.01), sumRow, orderedCyclic), (caValue,
f{s}k{2}r{1}i{center}x{81}y{120}w{3}c{8}s{0}, (constant, 109), (constant,
0.003), sumRow, (constant, 0), (constant, 1), orderedCyclic), (constant, 1)
```

C.2.13. rhythmSieve (rs)

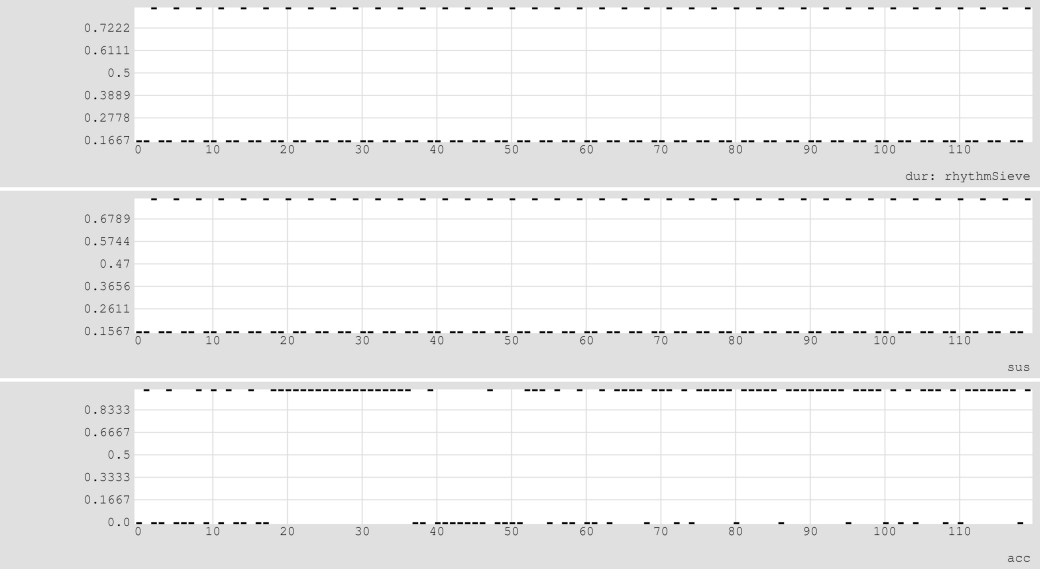
rhythmSieve, logicalString, sieveLength, selectionString, parameterObject

Description: Using the user-supplied logical string, this Generator produces a Xenakis sieve segment within the z range of zero to one less than the supplied length. The resulting binary sieve segment is used to filter any non-rest Pulse sequence generated by a Rhythm ParameterObject. The sieve is interpreted as a mask upon the ordered positions of the generated list of Pulses, where a sieve value retains the Pulse at the corresponding position, and all other Pulses are converted to rests. Note: any rests in the generated Pulse sequence will be converted to non-rests before sieve filtering.

Arguments: (1) name, (2) logicalString, (3) sieveLength, (4) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}, (5) parameterObject {Rhythm Generator}

Sample Arguments: `rs, 3|4|5, 60, rw, (l,((3,1,1),(3,1,1),(3,5,1)))`

Example C-190. rhythmSieve Demonstration 1




```
rhythmSieve, 3@0|4@0|5@0, 60, randomWalk, (loop, ((3,1,+),(3,1,+),(3,5,+)),
orderedCyclic)
```

C.3. Filter ParameterObjects

C.3.1. bypass (b)

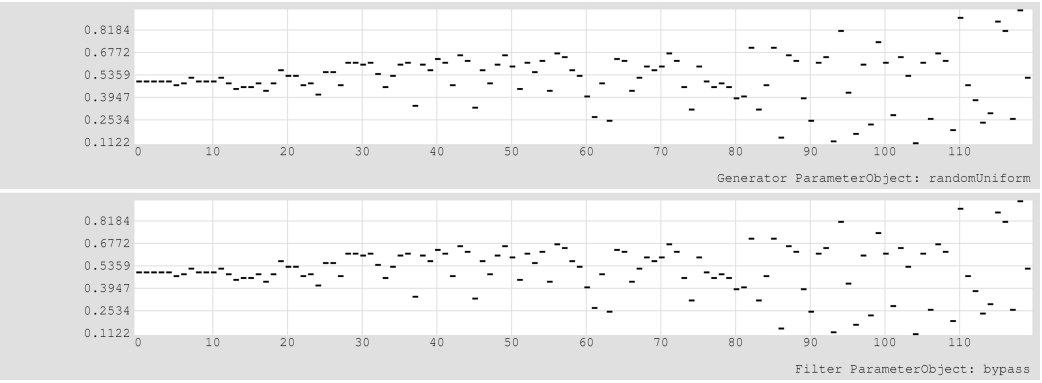
bypass

Description: Each input value is returned unaltered.

Arguments: (1) name

Sample Arguments: `b`

Example C-191. bypass Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
bypass
```

C.3.2. filterAdd (fa)

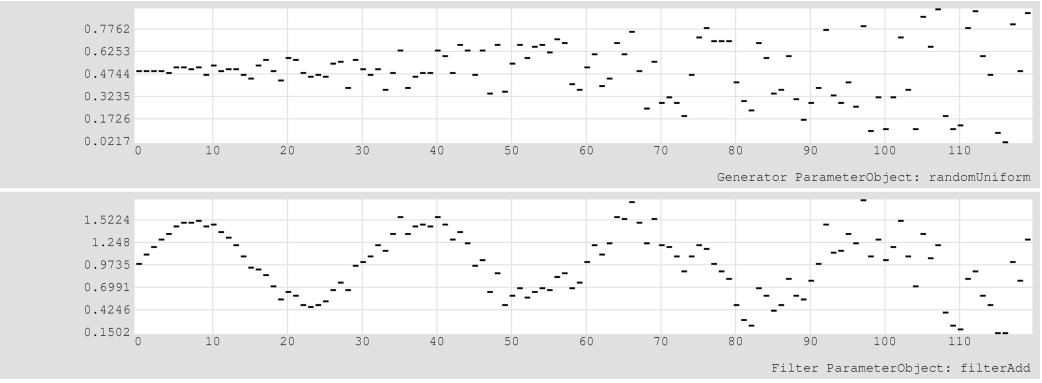
filterAdd, parameterObject

Description: Each input value is added to a value produced by a user-supplied ParameterObject.

Arguments: (1) name, (2) parameterObject {operator value generator}

Sample Arguments: `fa, (ws,e,30,0,0,1)`

Example C-192. filterAdd Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterAdd, (waveSine, event, (constant, 30), 0, (constant, 0), (constant, 1))
```

C.3.3. filterDivide (fd)

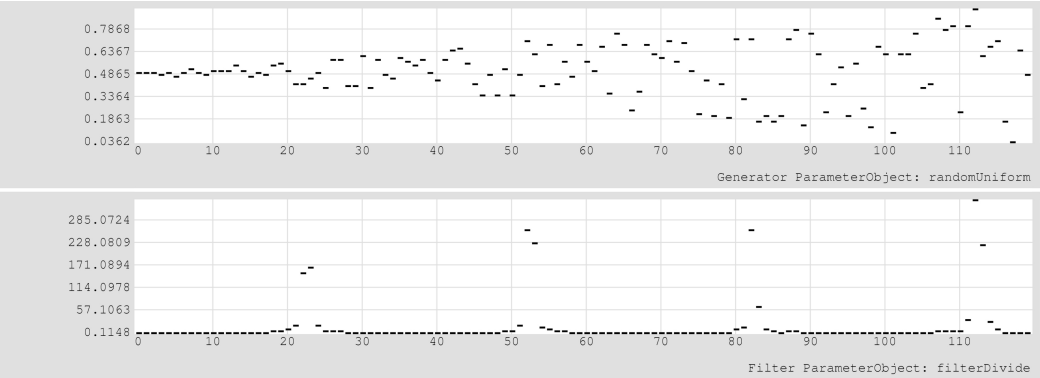
filterDivide, parameterObject

Description: Each input value is divided by a value produced by a user-supplied ParameterObject. Division by zero, if encountered, returns the value of the input value unaltered.

Arguments: (1) name, (2) parameterObject {operator value generator}

Sample Arguments: `fd, (ws,e,30,0,0,1)`

Example C-193. filterDivide Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterDivide, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1))
```

C.3.4. filterDivideAnchor (fda)

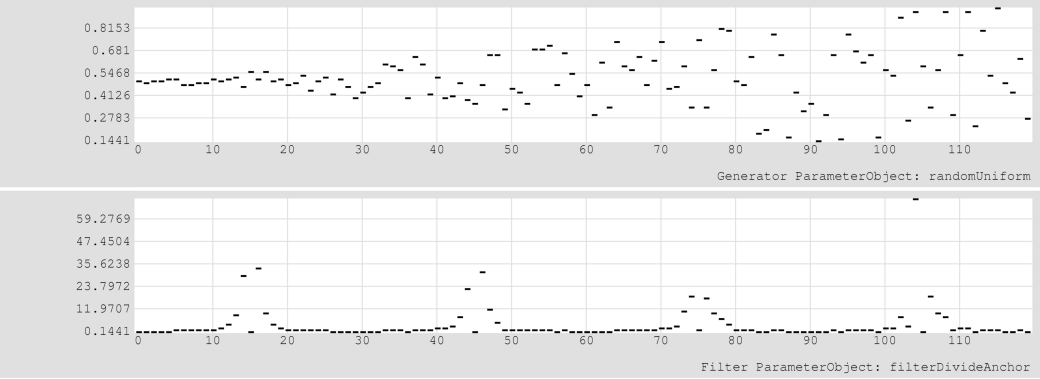
filterDivideAnchor, anchorString, parameterObject

Description: All input values are first shifted so that the position specified by anchor is zero; then each value is divided by the value produced by the parameterObject. All values are then re-shifted so that zero returns to its former position. Division by zero, if encountered, returns the value of the input value unaltered.

Arguments: (1) name, (2) anchorString {'lower', 'upper', 'average', 'median'}, (3) parameterObject {operator value generator}

Sample Arguments: `fda, lower, (wc,e,30,0,0,1)`

Example C-194. filterDivideAnchor Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterDivideAnchor, lower, (waveCosine, event, (constant, 30), 0, (constant,
0), (constant, 1))
```

C.3.5. filterFunnelBinary (ffb)

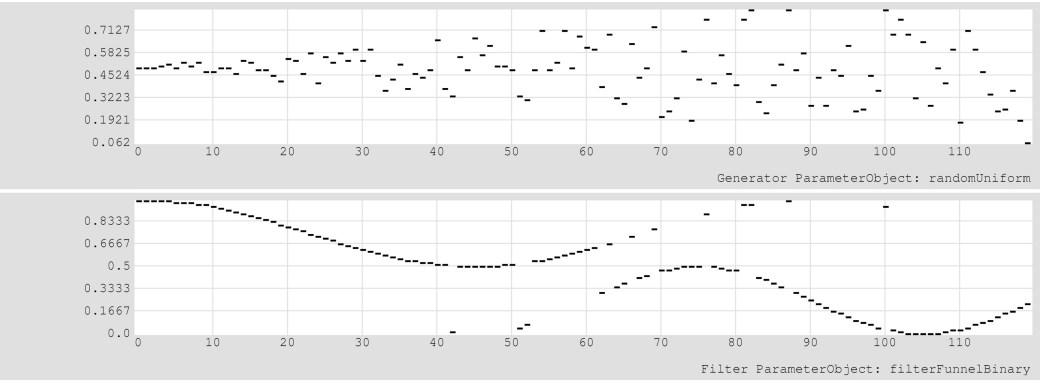
filterFunnelBinary, thresholdMatchString, parameterObject, parameterObject, parameterObject

Description: A dynamic, two-part variable funnel filter. Given values produced by two boundary parameterObjects and a threshold ParameterObject, the output of a Generator ParameterObject value is shifted to one of the boundaries (or the threshold) depending on the relationship of the generated value to the threshold. If the generated value is equal to the threshold, the value may be shifted to the upper or lower value, or retain the threshold value.

Arguments: (1) name, (2) thresholdMatchString {'upper', 'lower', 'match'}, (3) parameterObject {threshold}, (4) parameterObject {first boundary}, (5) parameterObject {second boundary}

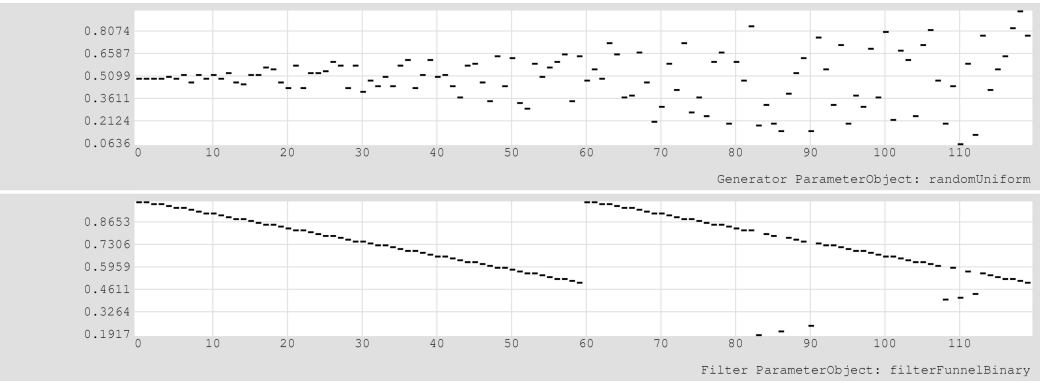
Sample Arguments: `ffb, u, (bpl,e,s,((0,0),(120,1))), (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1)`

Example C-195. filterFunnelBinary Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterFunnelBinary, upper, (breakPointLinear, event, single, ((0,0),(120,1))),
(waveSine, event, (constant, 60), 0, (constant, 0.5), (constant, 0)),
(waveCosine, event, (constant, 90), 0, (constant, 0.5), (constant, 1))
```

Example C-196. filterFunnelBinary Demonstration 2




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterFunnelBinary, match, (constant, 0.2), (breakPointLinear, event, loop,
((0,0),(60,0.5))), (breakPointLinear, event, loop, ((0,1),(60,0.5)))
```

C.3.6. filterMultiply (fm)

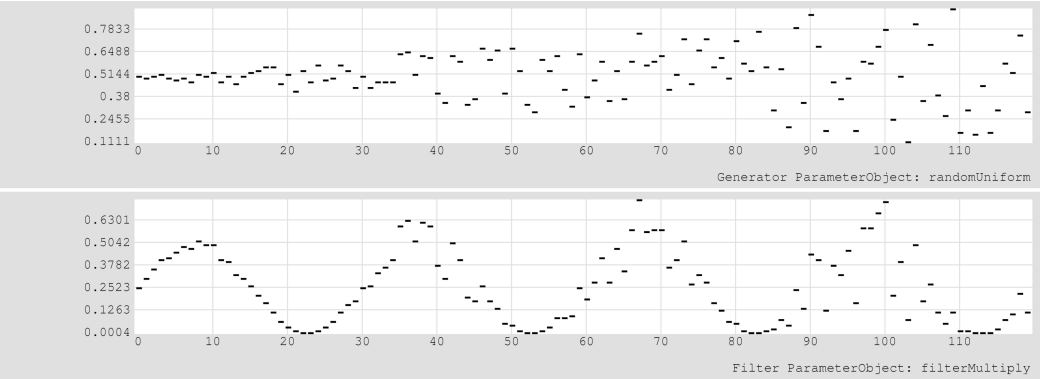
filterMultiply, parameterObject

Description: Each input value is multiplied by a value produced by a user-supplied ParameterObject.

Arguments: (1) name, (2) parameterObject {operator value generator}

Sample Arguments: `fm, (ws,e,30,0,0,1)`

Example C-197. filterMultiply Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterMultiply, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1))
```

C.3.7. filterMultiplyAnchor (fma)

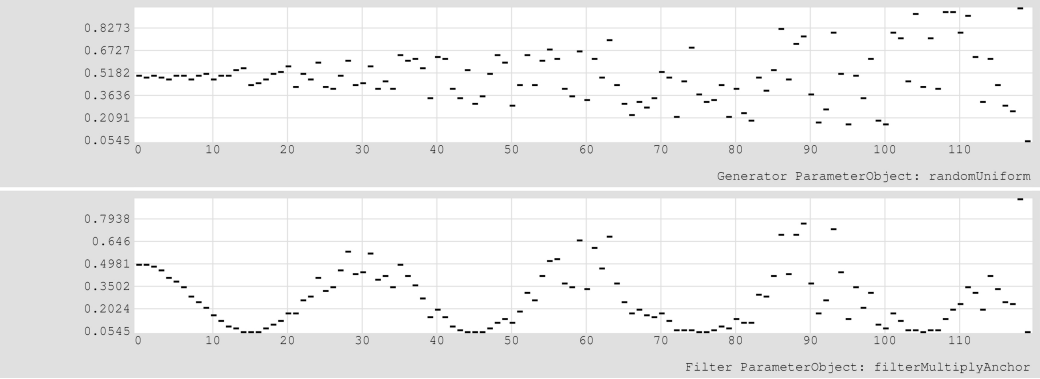
filterMultiplyAnchor, anchorString, parameterObject

Description: All input values are first shifted so that the position specified by anchor is zero; then each value is multiplied by the value produced by the parameterObject. All values are then re-shifted so that zero returns to its former position.

Arguments: (1) name, (2) anchorString {'lower', 'upper', 'average', 'median'}, (3) parameterObject {operator value generator}

Sample Arguments: `fma, lower, (wc,e,30,0,0,1)`

Example C-198. filterMultiplyAnchor Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterMultiplyAnchor, lower, (waveCosine, event, (constant, 30), 0, (constant,
0), (constant, 1))
```

C.3.8. filterPower (fp)

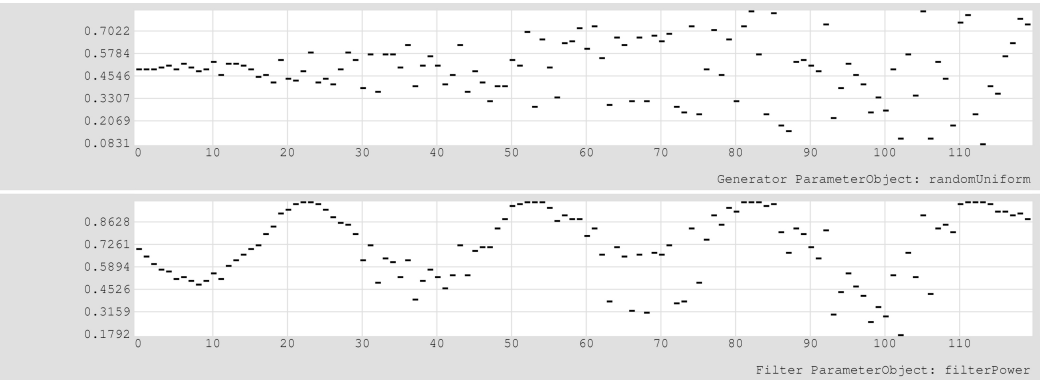
filterPower, parameterObject

Description: Each input value is taken to the power of the value produced by a user-supplied ParameterObject.

Arguments: (1) name, (2) parameterObject {operator value generator}

Sample Arguments: `fp, (ws,e,30,0,0,1)`

Example C-199. filterPower Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
```

```
filterPower, (waveSine, event, (constant, 30), 0, (constant, 0), (constant,
1))
```

C.3.9. filterQuantize (fq)

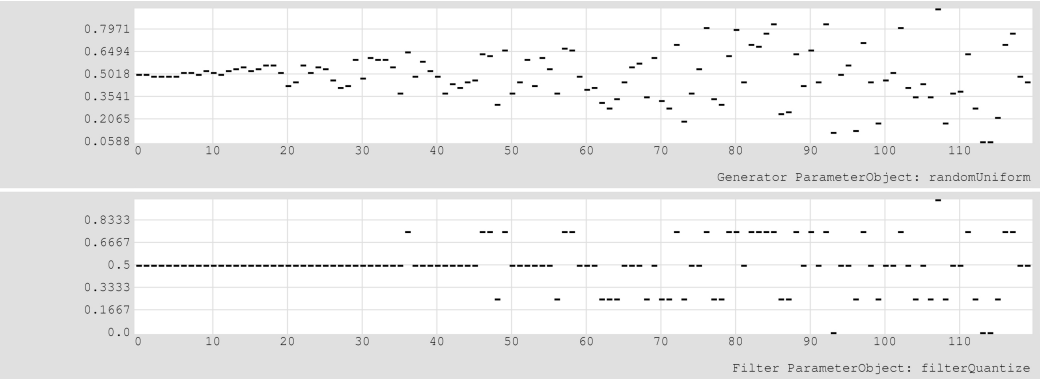
filterQuantize, parameterObject, parameterObject, stepCount, parameterObject

Description: Dynamic grid size and grid position quantization filter. For each value provided by the source ParameterObject, a grid is created. This grid is made by taking the number of steps specified by the stepCount integer from the step width Generator ParameterObject. The absolute value of these widths are used to create a grid above and below the reference value, with grid steps taken in order. The value provided by the source ParameterObject is found within this grid, and pulled to the nearest grid line. The degree of pull can be a dynamically allocated with a unit-interval quantize pull ParameterObject. A value of 1 forces all values to snap to the grid; a value of .5 will cause a weighted attraction.

Arguments: (1) name, (2) parameterObject {grid reference value Generator}, (3) parameterObject {step width Generator}, (4) stepCount, (5) parameterObject {unit interval measure of quantize pull}

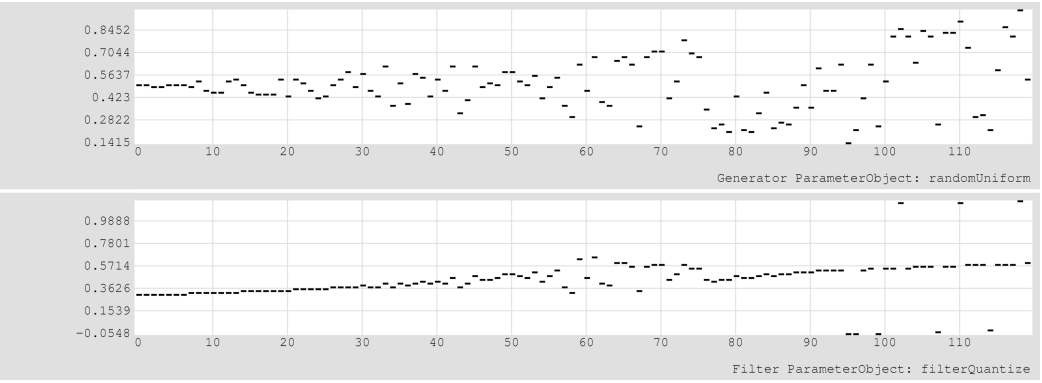
Sample Arguments: `fq, (c,0), (c,0.25), 1, (c,1)`

Example C-200. filterQuantize Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterQuantize, (constant, 0), (constant, 0.25), 1, (constant, 1)
```

Example C-201. filterQuantize Demonstration 2




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
filterQuantize, (cyclicGen, up, 0, 1, 0.003), (basketGen, orderedCyclic,
(0.4,0.6)), 2, (breakPointPower, event, loop, ((0,1),(59,0),(119,1)), -3)
```

C.3.10. maskFilter (mf)

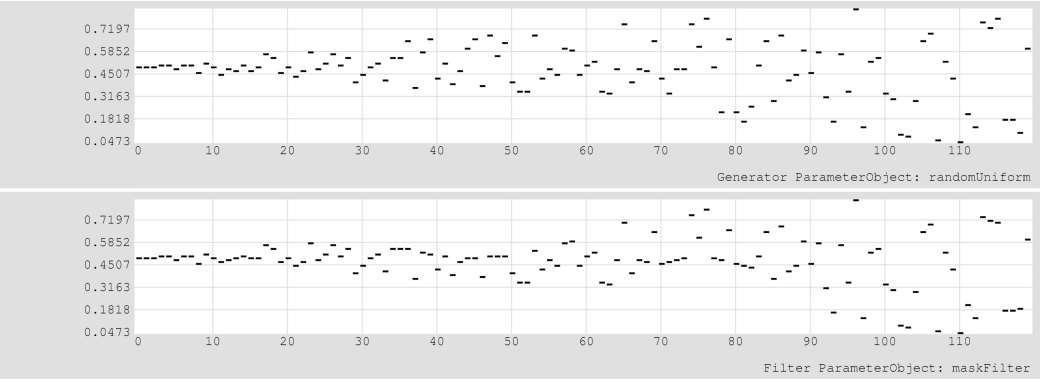
maskFilter, boundaryString, parameterObject, parameterObject

Description: Each input value is fit between values provided by two boundary Generator ParameterObjects. The fit is determined by the boundaryString: limit will fix the value at the nearest boundary; wrap will wrap the value through the range defined by the boundaries; reflect will bounce values in the opposite direction through the range defined by the boundaries.

Arguments: (1) name, (2) boundaryString {'limit', 'wrap', 'reflect'}, (3) parameterObject {first boundary}, (4) parameterObject {second boundary}

Sample Arguments: `mf, l, (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1)`

Example C-202. maskFilter Demonstration 1



```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
maskFilter, limit, (waveSine, event, (constant, 60), 0, (constant, 0.5),
(constant, 0)), (waveCosine, event, (constant, 90), 0, (constant, 0.5),
(constant, 1))
```

C.3.11. maskScaleFilter (msf)

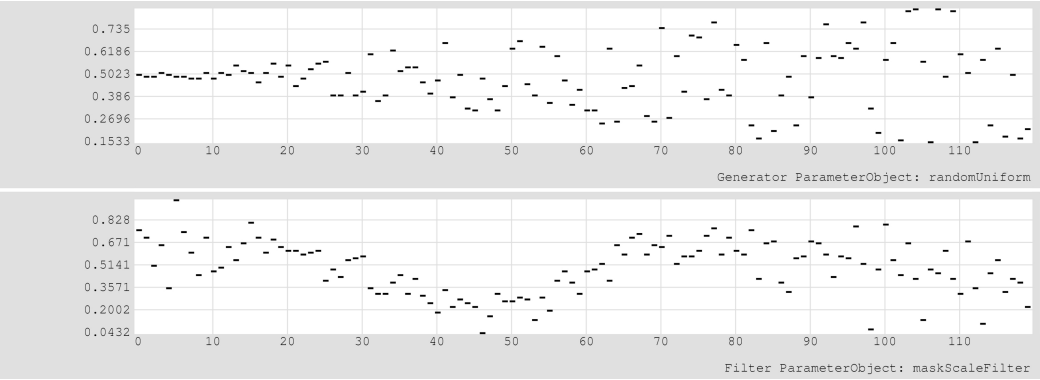
maskScaleFilter, min, max, selectionString

Description: Each input value is collected into a list. The resulting list of values is normalized within the unit interval. Values are chosen from this list using the selector specified by the selectionString argument. After selection, this value is scaled within the range designated by min and max; min and max may be specified with ParameterObjects.

Arguments: (1) name, (2) min, (3) max, (4) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `msf, (ws,e,60,0,0.5,0), (wc,e,90,0,0.5,1), rc`

Example C-203. maskScaleFilter Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
maskScaleFilter, (waveSine, event, (constant, 60), 0, (constant, 0.5),
(constant, 0)), (waveCosine, event, (constant, 90), 0, (constant, 0.5),
(constant, 1)), randomChoice
```

C.3.12. orderBackward (ob)

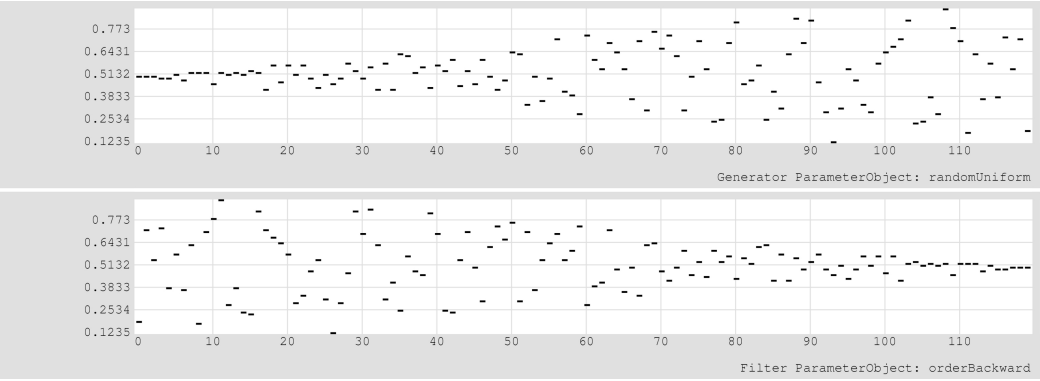
orderBackward

Description: All values input are returned in reversed order.

Arguments: (1) name

Sample Arguments: `ob`

Example C-204. orderBackward Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
orderBackward
```

C.3.13. orderRotate (or)

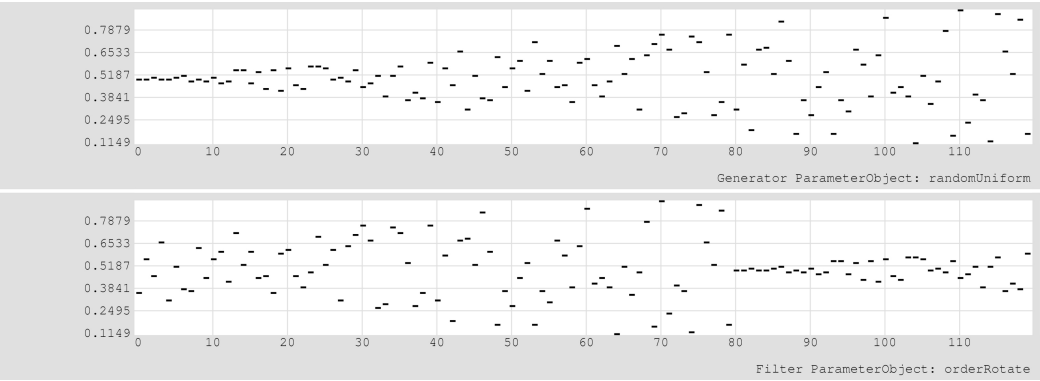
orderRotate, rotationSize

Description: Rotates all input values as many steps as specified; if the number of steps is greater than the number of input values, the modulus of the input length is used.

Arguments: (1) name, (2) rotationSize

Sample Arguments: `or, 40`

Example C-205. orderRotate Demonstration 1



```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
orderRotate, 40
```

C.3.14. pipeLine (pl)

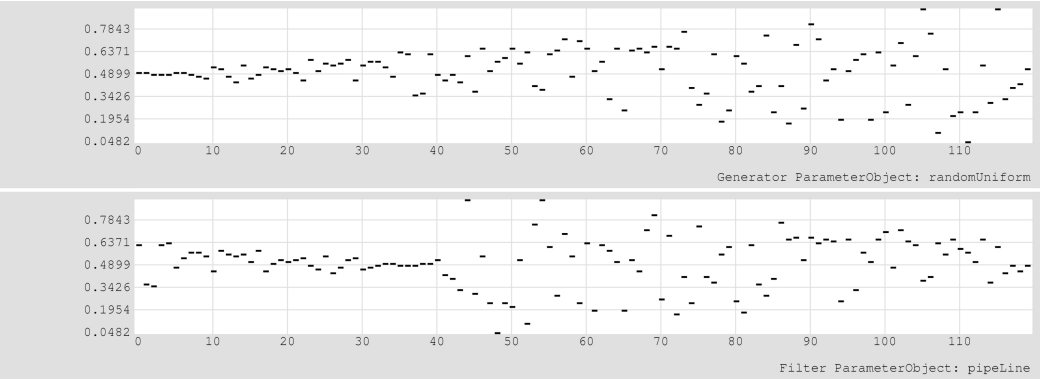
pipeLine, filterParameterObjectList

Description: Provide a list of Filter ParameterObjects; input values are passed through each filter in the user-supplied order from left to right.

Arguments: (1) name, (2) filterParameterObjectList {a list of sequential Filter ParameterObjects}

Sample Arguments: `pl, ((or,40),(ob))`

Example C-206. pipeLine Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
pipeLine, ((orderRotate, 40), (orderBackward))
```

C.3.15. replace (r)

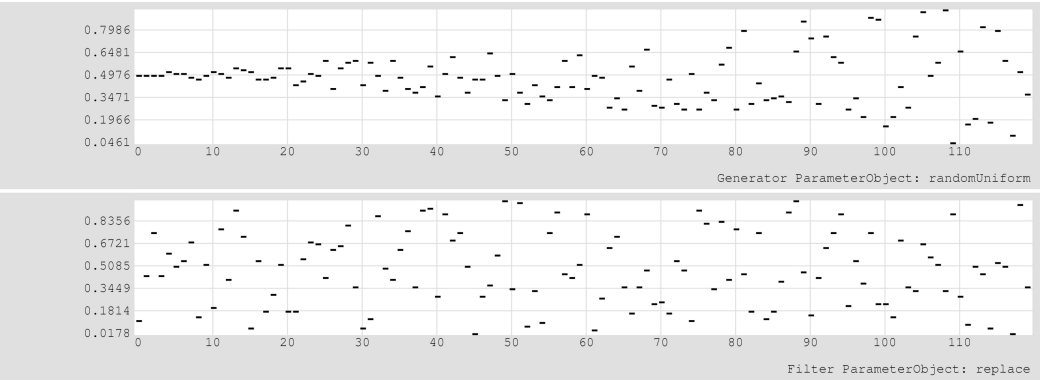
replace, parameterObject

Description: Replace input values with values produced by a Generator ParameterObject.

Arguments: (1) name, (2) parameterObject {generator to replace original values}

Sample Arguments: `r, (ru,0,1)`

Example C-207. replace Demonstration 1




```
randomUniform, (breakPointLinear, event, loop, ((0,0.5),(120,0))),
(breakPointLinear, event, loop, ((0,0.5),(120,1)))
replace, (randomUniform, (constant, 0), (constant, 1))
```

C.4. TextureStatic ParameterObjects

C.4.1. eventDensityPartition (edp)

eventDensityPartition, level

Description: Define how event count is distributed within a Texture, either proportional to path duration or equal proportion per path set.

Arguments: (1) name, (2) level {'duration', 'set'}

Sample Arguments: `edp, duration`

C.4.2. interpolationMethodControl (imc)

interpolationMethodControl, method

Description: Selects the type of interpolation used for all parameters.

Arguments: (1) name, (2) method {'linear', 'halfCosine', 'power'}

Sample Arguments: `imc, linear`

C.4.3. levelEventCount (lec)

levelEventCount, level

Description: Define at what level event count values are generated: once per Texture for the total event count (with a distribution per segment proportional to segment duration), or once per segment for each segment event count.

Arguments: (1) name, (2) level {'segment', 'texture'}

Sample Arguments: `lec, segment`

C.4.4. levelEventPartition (lep)

levelEventPartition, level

Description: Toggle between selection of event start time per set of the Texture Path, or per Path. This control will determine if the event generator is mapped within the Texture time range, or within the set time range. When set to path, this control will over-ride event density partitioning.

Arguments: (1) name, (2) level {'set', 'path'}

Sample Arguments: `lep, path`

C.4.5. levelFrameDuration (lfd)

levelFrameDuration, level

Description: Toggle between selection of frame duration values per frame or per event.

Arguments: (1) name, (2) level {'event', 'frame'}

Sample Arguments: `lfd, event`

C.4.6. levelFieldMonophonic (lfm)

levelFieldMonophonic, level

Description: Toggle between selection of local field (transposition) values per set of the Texture Path, or per event.

Arguments: (1) name, (2) level {'set', 'event'}

Sample Arguments: `lfm, event`

C.4.7. levelFieldPolyphonic (lfp)

levelFieldPolyphonic, level

Description: Toggle between selection of local field (transposition) values per set of the Texture Path, per event, or per polyphonic voice event.

Arguments: (1) name, (2) level {'set', 'event', 'voice'}

Sample Arguments: `lfp, event`

C.4.8. levelOctaveMonophonic (lom)

levelOctaveMonophonic, level

Description: Toggle between selection of local octave (transposition) values per set of the Texture Path, or per event.

Arguments: (1) name, (2) level {'set', 'event'}

Sample Arguments: `lom, event`

C.4.9. levelOctavePolyphonic (lop)

levelOctavePolyphonic, level

Description: Toggle between selection of local octave (transposition) values per set of the Texture Path, per event, or per polyphonic voice event.

Arguments: (1) name, (2) level {'set', 'event', 'voice'}

Sample Arguments: `lop, event`

C.4.10. loopWithinSet (lws)

loopWithinSet, onOff

Description: Controls if pitches in a set are repeated by a Texture within the set's duration fraction.

Arguments: (1) name, (2) onOff {'on', 'off'}

Sample Arguments: `lws, on`

C.4.11. multisetSelectorControl (msc)

multisetSelectorControl, selectionString

Description: Define the selector method of Multiset selection within a Path used by a Texture.

Arguments: (1) name, (2) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `msc, randomPermutate`

C.4.12. maxTimeOffset (mto)

maxTimeOffset, time

Description: Used to select an offset time in seconds. Offset is applied with the absolute value of a gaussian distribution after the Texture-generated event start time.

Arguments: (1) name, (2) time

Sample Arguments: `mto, 0.03`

C.4.13. ornamentLibrarySelect (ols)

ornamentLibrarySelect, libraryName

Description: Selects a library of ornaments to use with a Texture.

Arguments: (1) name, (2) libraryName {'chromaticGroupC', 'diatonicGroupA', 'diatonicGroupB', 'microGroupA', 'microGroupB', 'microGroupC', 'trillGroupA', 'off'}

Sample Arguments: `ols, diatonicGroupA`

C.4.14. ornamentMaxDensity (omd)

ornamentMaxDensity, percent

Description: Controls maximum percent of events that are ornamented. Density value should be specified within the unit interval.

Arguments: (1) name, (2) percent

Sample Arguments: `omd, 1`

C.4.15. pathDurationFraction (pdf)

pathDurationFraction, onOff

Description: Toggle Path duration fraction; if off, Path duration fractions are not used to partition Path deployment over the duration of the Texture. Instead, each Path set is used to create a single event.

Arguments: (1) name, (2) onOff {'on', 'off'}

Sample Arguments: `pdf, on`

C.4.16. parameterInterpolationControl (pic)

parameterInterpolationControl, onOff

Description: Controls if all non-duration parameter values are interpolated between events.

Arguments: (1) name, (2) onOff {'on', 'off'}

Sample Arguments: `pic, on`

C.4.17. parallelMotionList (pml)

parallelMotionList, transpositionList, timeDelay

Description: List is a collection of transpositions created above every Texture-generated base note. The timeDelay value determines the amount of time in seconds between each successive transposition in the transpositionList.

Arguments: (1) name, (2) transpositionList, (3) timeDelay

Sample Arguments: `pml, (), 0.0`

C.4.18. pitchSelectorControl (psc)

pitchSelectorControl, selectionString

Description: Define the selector method of Path pitch selection used by a Texture.

Arguments: (1) name, (2) selectionString {'randomChoice', 'randomWalk', 'randomPermutate', 'orderedCyclic', 'orderedCyclicRetrograde', 'orderedOscillate'}

Sample Arguments: `psc, randomPermutate`

C.4.19. snapEventTime (set)

snapEventTime, onOff

Description: Controls if all event start times are shifted to align with frame divisions.

Arguments: (1) name, (2) onOff {'on', 'off'}

Sample Arguments: `set, on`

C.4.20. snapSustainTime (sst)

snapSustainTime, onOff

Description: Controls if all event sustain values are scaled to the frame width.

Arguments: (1) name, (2) onOff {'on', 'off'}

Sample Arguments: `sst, on`

C.4.21. totalEventCount (tec)

totalEventCount, count

Description: Selects the total number of events generated within the Texture time range.

Arguments: (1) name, (2) count

Sample Arguments: `tec, 20`

C.4.22. totalSegmentCount (tsc)

totalSegmentCount, count

Description: Set the number of segments with which to divide the Texture's duration.

Arguments: (1) name, (2) count

Sample Arguments: `tsc, 10`

C.5. CloneStatic ParameterObjects

C.5.1. retrogradeMethodToggle (rmt)

retrogradeMethodToggle, name

Description: Selects type of retrograde transformation applied to Texture events.

Arguments: (1) name, (2) name {'timeInverse', 'eventInverse', 'off'}

Sample Arguments: `rmt, off`

C.5.2. timeReferenceSource (trs)

timeReferenceSource, name

Description: Selects time reference source used in calculating ParameterObjects.

Arguments: (1) name, (2) name {'cloneTime', 'textureTime'}

Sample Arguments: `trs, textureTime`

Appendix D. Temperament and TextureModule Reference

D.1. Temperaments

D.1.1. Temperament Interleave24Even

Even steps of a 24 tone equal tempered scale

D.1.2. Temperament Interleave24Odd

Odd steps of a 24 tone equal tempered scale

D.1.3. Temperament Just

Static Just tuning

D.1.4. Temperament MeanTone

Static Mean Tone tuning

D.1.5. Temperament NoiseHeavy

Provide uniform random +/- 15 cent noise on each pitch

D.1.6. Temperament NoiseLight

Provide uniform random +/- 5 cent noise on each pitch

D.1.7. Temperament NoiseMedium

Provide uniform random +/- 10 cent noise on each pitch

D.1.8. Temperament Pythagorean

Static Pythagorean tuning

D.1.9. Temperament Split24Lower

Lower half of a 24 tone equal tempered scale

D.1.10. Temperament Split24Upper

Upper half of a 24 tone equal tempered scale

D.1.11. Temperament TwelveEqual

Twelve tone equal temperament

D.2. TextureModules

D.2.1. TextureModule DroneArticulate

This non-linear TextureModule treats each pitch in each set of a Path as an independent voice; each voice is written one at time over the complete time range of each set in the Texture.

D.2.2. TextureModule DroneSustain

This TextureModule performs a simple vertical presentation of the Path, each set sustained over the complete duration proportion of the set within the Texture. Note: rhythm and bpm values have no effect on event durations.

D.2.3. TextureModule HarmonicAssembly

This TextureModule provides free access to Path pitch collections in an order, rate, simultaneity size, and simultaneity composition determined by generator ParameterObjects. Path Multisets are directly selected by index values generated by a ParameterObject; all values are probabilistically rounded to the nearest integer and are resolved by the modulus of the Path length. The number of simultaneities created from a selected Multiset is controlled by a generator ParameterObject; all values are probabilistically rounded to the nearest integer. Pitches within Multisets are directly chosen by index values generated by a ParameterObject; all values are probabilistically rounded to the nearest integer and are resolved by the modulus of the Multiset size. The number of pitches extracted from a Multiset is controlled by a generator ParameterObject; a size of zero takes all pitches from the selected Multiset; sizes greater than the number of pitches are resolved to the maximum number of pitches. Remaining event parameters are determined by their respective ParameterObjects.

D.2.4. TextureModule HarmonicShuffle

This TextureModule provides limited access to Path pitch collections in an order, rate, simultaneity size, and simultaneity composition determined by generator ParameterObjects. Path Multisets and pitches within Multisets are chosen by selectors. The number of simultaneities that are created from a Multiset, and the number of pitches in each simultaneity, are controlled by generator ParameterObjects; all values are probabilistically rounded to the nearest integer. When extracting

pitches, a size of zero takes all pitches from the selected Multiset; sizes greater than the number of available pitches are resolved to the maximum number of pitches. Remaining event parameters are determined by their respective ParameterObjects.

D.2.5. TextureModule InterpolateFill

This TextureModule interpolates parameters between events generated under a non-linear monophonic context. All standard and auxiliary parameters, or just time parameters, can be interpolated. Interpolation method may be linear, power, or half-cosine. Frames are generated between each event at a rate controlled by a ParameterObject. Frame rates can be updated once per event or once per frame, as set by the level frame duration texture parameter. Power segment interpolation may use dynamic exponent values from a ParameterObject; exponent values are updated once per event. Note: independent of silenceMode, silent events are always created.

D.2.6. TextureModule InterpolateLine

This TextureModule interpolates parameters between events generated under a linear monophonic context. All standard and auxiliary parameters, or just time parameters, can be interpolated. Interpolation method may be linear, power, or half-cosine. Frames are generated between each event at a rate controlled by a ParameterObject. Frame rates can be updated once per event or once per frame, as set by the level frame duration texture parameter. Power segment interpolation may use dynamic exponent values from a ParameterObject; exponent values are updated once per event. Note: independent of silenceMode, silent events are always created.

D.2.7. TextureModule IntervalExpansion

This TextureModule performs each set of a Path as a literal line; pitches are chosen from sets in order, and are optionally repeated within a single set's duration. Algorithmic ornamentation is added to a line based on two factors: the selection of an ornament repertory, and the specification of ornament density. Ornament pitch values, where integers are half steps, are additionally shifted by a value produced by a generator ParameterObject.

D.2.8. TextureModule LineCluster

This TextureModule performs each set of a Path as a chord cluster, randomly choosing different voicings.

D.2.9. TextureModule LineGroove

This TextureModule performs each set of a Path as a simple monophonic line; pitches are chosen from sets in the Path based on the pitch selector control.

D.2.10. TextureModule LiteralHorizontal

This TextureModule performs each set of a Path as a literal horizontal line; pitches are chosen from sets in fixed order, and are optionally repeated within a single set's proportional duration.

D.2.11. TextureModule LiteralVertical

This TextureModule performs each set of a Path as a literal verticality; pitches are chosen from sets in fixed order, and are optionally repeated within a single set's proportional duration.

D.2.12. TextureModule MonophonicOrnament

This TextureModule performs each set of a Path as a literal line; pitches are chosen from sets in order, and are optionally repeated within a single set's duration. Algorithmic ornamentation is added to a line based on two factors: the selection of an ornament repertory, and the specification of ornament density.

D.2.13. TextureModule TimeFill

This non-linear TextureModule fills a Texture time range with events; event start times are determined by mapping values produced by a generator ParameterObject (set to output values between 0 and 1) to the Texture time range. Remaining event parameters are determined by their respective ParameterObjects.

D.2.14. TextureModule TimeSegment

This non-linear TextureModule fills a Texture time range with events; event start times are determined by mapping values produced by a generator ParameterObject (set to output values between 0 and 1) to segments of the Texture time range, where each segment width is determined by both a generator ParameterObject for segment weight and a the total segment count. Segment weights are treated as proportional weightings of the Texture's duration. Remaining event parameters are determined by their respective ParameterObjects.

Appendix E. OutputFormat and OutputEngine Reference

E.1. OutputFormats

E.1.1. acToolbox

acToolbox: AC Toolbox Environment file. (.act)

E.1.2. audioFile

audioFile: Pulse Code Modulation (PCM) file. (.synth.aif)

E.1.3. csoundBatch

csoundBatch: Platform specific script or batch file. (.bat)

E.1.4. csoundData

csoundData: Csound XML unified file format. (.csd)

E.1.5. csoundOrchestra

csoundOrchestra: Csound orchestra file. (.orc)

E.1.6. csoundScore

csoundScore: Csound score file. (.sco)

E.1.7. midiFile

midiFile: Standard MIDI file. (.mid)

E.1.8. pureDataArray

pureDataArray: PureData (PD) patch with defined arrays. (.pd)

E.1.9. scScd

scScd: SuperCollider task data format. (.scd)

E.1.10. textSpace

textSpace: Space delimited event list. (.space.txt)

E.1.11. textTab

textTab: Tab delimited event list. (.tab.txt)

E.1.12. xmlAthenaObject

xmlAthenaObject: athenaCL native XML format. (.xml)

E.2. OutputEngines

E.2.1. EngineAcToolbox

Translates each Texture and each Clone into a Section and writes an Environment file for loading within Paul Berg's AC Toolbox. A Parallel Section, containing references to each of these Sections, is also provided. Compatible with all Orchestras; GeneralMidi Orchestra will be used for event postMap conversions.

E.2.2. EngineAudioFile

Translates events to audio samples, and writes an audio file. Each event's amplitude is scaled between -1 and 1. Event timing and other event parameter data are stripped. Compatible with all Orchestras.

E.2.3. EngineCsoundExternal

Translates events to a Csound score for use with an external orchestra. Event parameters instrument number, start time, and duration are always the first three parameters. Additional event parameters taken from auxiliary parameters. Compatible with all Orchestras.

E.2.4. EngineCsoundNative

Translates events to a Csound score for use with the native Csound orchestra. All event parameters are retained. Compatible only with the CsoundNative Orchestra.

E.2.5. EngineCsoundSilence

Translates Texture and Clone events to a Csound score for use with the Csound Silence system by Michael Goggins. Event parameters follow a standard number and order. Standard panning control applied to x pan event parameter. Compatible only with the CsoundSilence Orchestra.

E.2.6. EngineMidiFile

Translates events to a standard (type 1) MIDI file. Compatible with all Orchestras; in all cases events are translated with the GeneralMidi Orchestra.

E.2.7. EnginePureDataArray

Translates all event parameter streams to individual Pure Data (PD) arrays. Such arrays can be read from at the control or audio rate from within PD using tabread and related objects. Compatible with all Orchestras.

E.2.8. EngineSuperColliderTask

Translates events to a SuperCollider task process file for use with the native SuperCollider orchestra. All event parameters are retained. Compatible only with the SuperColliderNative Orchestra.

E.2.9. EngineText

Translate events to a plain text file. All event parameter values are separated by a delimiter (tab or space) and ended with a return carriage. Compatible with all Orchestras; EventMode Orchestra will be used for event postMap conversions.

Appendix F. Demonstration Command Scripts in Python

The following scripts provide both brief examples of how to programmatically send commands to the athenaCL Interpreter and also demonstrate numerous fundamental concepts and techniques. All .py files shown here are included in the athenaCL demo directory. Pre-rendered MIDI and Csound files are also included.

F.1. MIDI-based Output

F.1.1. script01a.py: Configuring Rhythms

```
# Configuring Rhythms
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 45',
'tie a rb,.3,.3,.4,.8',
'tie r pt,(c,4),(bg,oc,(3,3,2)),(c,1)',
'tin b 65',
'tie a re,15,.3,1',
'tie r pt,(bg,rp,(2,1,1,1)),(c,1),(c,1)',
'tin c 67',
'tie a rb,.1,.1,.4,.6',
'tie r cs,(rb,.2,.2,.01,1.5)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.2. script01b.py: Configuring Time Range

```
# Configuring Time Range
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo mp',
'tin a 45',
'tie t 0,20',
'tie a rb,.3,.3,.4,.8',
'tie r pt,(c,4),(bg,oc,(3,3,2)),(c,1)',
'tin b 65',
'tie t 10,20',
'tie a re,15,.3,1',
'tie r pt,(bg,rp,(2,1,1,1)),(c,1),(c,1)',
```

```
'tin c 67',
'tie t 15,25',
'tie a rb,.1,.1,.4,.6',
'tie r cs,(rb,.2,.2,.01,1.5)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.3. script02a.py: Building a Basic Beat

```
# Building a Basic Beat
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 36',
'tie r pt,(c,2),(bg,oc,(7,5,2,1,1)),(c,1)',
'tin b 37',
'tie r pt,(c,2),(bg,oc,(3,5)),(bg,oc,(0,1)) ',
'tin c 42',
'tie r pt,(c,2),(c,1),(bg,oc,(0,1))',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.4. script02b.py: Building a Basic Beat with a Complex Snare Part

```
# Building a Basic Beat with a Complex Snare Part
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 36',
'tie r pt,(c,2),(bg,oc,(7,5,2,1,1)),(c,1)',
'tin b 37',
'tie r pt,(c,4),(bg,rp,(3,3,5,4,1)),(bg,oc,(0,1,1))',
'tin c 42',
'tie r pt,(c,2),(c,1),(bg,oc,(0,1))',
]
def main(cmdList=[], fp=None, hear=True):
```

```
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.5. script02c.py: Building a Basic Beat with Canonic Snare Imitation

```
# Building a Basic Beat with Canonic Snare Imitation
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 36',
'tie r pt,(c,2),(bg,oc,(7,5,2,1,1)),(c,1)',
'tin b 37',
'tie r pt,(c,4),(bg,rp,(3,3,5,4,1)),(bg,oc,(0,1,1))',
'tin c 42',
'tie r pt,(c,2),(c,1),(bg,oc,(0,1))',
'tio b',
'ticp b b1',
'tie t .25, 20.25',
'tie i 76',
'ticp b b2',
'tie t .5, 20.5',
'tie i 77',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.6. script02d.py: Building an Extended Rhythmic Line with Canonic Imitation

```
# Building an Extended Rhythmic Line with Canonic Imitation
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 77',
'tie r pt,(c,1),(c,1),(c,1)',
'tin b 67',
'tie r pt,(bg,oc,(2,4,1)),(bg,oc,(3,5,1,7,1,3)),(c,1) ',
'ticp b b1',
'tie t 0.125,20.125',
'tie i 60',
```

```
'ticp b b2',
'tie t 0.25,20.25',
'tie i 68',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.7. script02e.py: Creating Mensural Canons

```
# Creating Mensural Canons
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 77',
'tie r pt,(c,1),(c,1),(c,1)',
'tin b 67',
'tie r pt,(bg,oc,(2,4,1)),(bg,oc,(3,5,1,7,1,3)),(c,1) ',
'ticp b b1',
'tie t 0.125,20.125',
'tie i 60',
'ticp b b2',
'tie t 0.25,20.25',
'tie i 68',
'tio b1',
'tie b c,90',
'tio b2',
'tie b c,180',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.8. script02f.py: Building an Extended Rhythmic Line with Fixed Tempo Phasing

```
# Building an Extended Rhythmic Line with Fixed Tempo Phasing
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
```

```
'tin a 70',
'tie r pt,(bg,oc,(2,4,4)),(bg,oc,(4,1,1,2,1)),(c,1) ',
'tie t 0,60',
'ticp a a1',
'tie b c,124',
'ticp a a2',
'tie b c,128',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.9. script02g.py: Building an Extended Rhythmic Line with Dynamic Tempo Phasing

```
# Building an Extended Rhythmic Line with Dynamic Tempo Phasing
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 64',
'tie r pt,(bg,oc,(2,4,4)),(bg,oc,(4,1,1,2,1)),(c,1) ',
'tie t 0,60',
'ticp a a1',
'tie i 60',
'tie b ws,t,20,0,115,125',
'ticp a a2',
'tie i 69',
'tie b ws,t,30,0,100,140',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.10. script03a.py: Grouping Selection

```
# Grouping Selection
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo m',
```

```
'tin a 6',
'tie r cs,(rb,.2,.2,.02,.25)',
'tie f ig,(bg,rc,(2,4,7,9,11)),(bg,rp,(2,3,5,8,13))',
'tie o ig,(bg,oc,(-2,-1,0,1)),(ru,20,30)',
'ticp a b c d',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.11. script03b.py: Tendency Mask: Random Values between Breakpoint Functions

```
# Tendency Mask: Random Values between Breakpoint Functions
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'tin a 15',
'tie r cs,(ig,(ru,.01,.25),(ru,4,12))',
'tie a ru,.2,(cg,u,.3,.9,.005)',
'tie f rb,.2,.2,(bpl,t,l,((0,-12),(30,12))),(ws,t,29,0,0,24)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.12. script03c.py: Tendency Mask: Random Values between Triangle Generators

```
# Tendency Mask: Random Values between Triangle Generators
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'pin a d,e,g,a,b',
'tin a 107  ',
'tie r pt,(c,16),(ig,(bg,rc,(1,2,3,5,7)),(bg,rc,(3,6,9,12))),(c,1)',
'tie o ru,(wt,t,25,0,-2,4),(wt,t,20,0,-3,1)',
]
def main(cmdList=[], fp=None, hear=True):
```

```
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.13. script04a.py: Large Scale Amplitude Behavior with Operators

```
# Large Scale Amplitude Behavior with Operators
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo mp',
'tin a 64',
'tie r pt,(bg,rp,(16,16,8)),(bg,rp,(2,2,1,4)),(c,1)',
'tie a om,(ls,e,9,(ru,.2,1),(ru,.2,1)),(wp,e,23,0,0,1)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.14. script04b.py: 1/f Noise in Melodic Generation: LineGroove

```
# 1/f Noise in Melodic Generation: LineGroove
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo m',
'tmo lg',
'tin a 108',
'tie r cs,(ls,e,10,(ru,.01,.2),(ru,.01,.2))',
'tie f bs,(2,4,7,9,11,14,16,19,21,23),(n,100,1,0,1)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
```

```
if __name__ == '__main__':
    main(cmd)
```

F.1.15. script04c.py: 1/f Noise in Melodic Generation: HarmonicAssembly

```
# 1/f Noise in Melodic Generation: HarmonicAssembly
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo m',
'pin a d3,e3,g3,a3,b3,d4,e4,g4,a4,b4,d5,e5,g5,a5,b5',
'tmo ha',
'tin a 27',
'tie r pt,(c,16),(ig,(bg,rc,(1,2,3,5,7)),(bg,rc,(3,6,9,12))),(c,1)',
'tie a om,(ls,e,9,(ru,.2,1),(ru,.2,1)),(wp,e,23,0,0,1)',
'tie d0 c,0',
'tie d1 n,100,2,0,14',
'tie d2 c,1',
#'tie d3 c,1',
'tie d3 ru,1,4',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.16. script05a.py: Self Similar Markovian Melody Generation and Transposition

```
# Self Similar Markovian Melody Generation and Transposition
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'tin a 24',
'tie r cs,(n,100,1.5,.100,.180)',
'tie r cs,(om,(n,100,1.5,.100,.180),(ws,t,8,0,.5,1))',
# Markov weighted pitch transposition
'tie f mv,a{2}b{4}c{7}d{9}e{11}:{a=1|b=6|c=1|d=9|e=1}',
# self-similar pitch transposition combing a grouped version of the same Markov
generator with OperatorAdd
'tie f oa,(mv,a{2}b{4}c{7}d{9}e{11}:{a=1|b=3|c=1|d=3|e=1}),
(ig,(mv,a{2}b{4}c{7}d{9}e{11}:{a=1|b=3|c=1|d=3|e=1}),(ru,10,20))',
# Markov based octave shifting
'tie o mv,a{-2}b{0}c{-2}d{0}e{-1}:{a=1|b=3|c=1|d=3|e=1}',
# A widening beta distribution
'tie a rb,.2,.5,(ls,e,(ru,3,20),.5,1)',
# Modulated with a pulse wave (and random frequency modulation on the PulseWave)
'tie a om,(rb,.2,.5,(ls,e,(ru,3,20),.5,1)),(wp,e,(ru,25,30),0,0,1)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
```

```
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.17. script05b.py: Markov-Based Proportional Rhythm Generation

```
# Markov-Based Proportional Rhythm Generation
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 64',
# simple zero-order selection
'tie r mp,a{4,1}b{4,3}c{4,5}d{4,7}:{a=4|b=3|c=2|d=1}',
# first order generation that encourages movement toward the shortest duration
'tie r mp,a{8,1}b{4,3}c{4,7}d{4,13}a:{a=9|d=1}b:{a=5|c=1}c:{b=1}d:{c=1},(c,1)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.18. script05c.py: Markov-Based Value Generation

```
# Markov-Based Value Generation
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'tin a 26',
# rhythm generated with absolute values via ConvertSecond and a dynamic
WaveHalfPeriodSine generator
'tie r cs,(whps,e,(bg,rp,(5,10,15,20)),0,.200,.050)',
# first-order selection
#'tie f
mv,a{2}b{4}c{7}d{9}e{11}:{a=1|b=3|c=1|d=3|e=1}a:{a=9|e=1}b:{a=3|c=1}c:{b=3|d=1}d:{c=3|
e=1}e:{d=1},(c,1)',
# dynamic first and zero order selection
'tie f
mv,a{2}b{4}c{7}d{9}e{11}:{a=1|b=3|c=1|d=3|e=1}a:{a=9|e=1}b:{a=3|c=1}c:{b=3|d=1}d:{c=3|
e=1}e:{d=1},(wp,e,100,0,1,0)',
# zero-order Markov amplitude values
#'tie a mv,a{.4}b{.6}c{.8}d{1}:{a=6|b=4|c=3|d=1}',
# amplitude values scaled by a dynamic WaveHalfPeriodPulse
'tie a om,(mv,a{.4}b{.6}c{.8}d{1}:{a=6|b=4|c=3|d=1}),(whpp,e,(bg,rp,(5,15,10)))',
```

```
# octave values are provided by a first-order Markov chain
'tie o mv,a{0}b{-1}c{-2}d{-3}a:{a=9|d=1}b:{a=3|b=1}c:{b=3|c=1}d:{c=1},(c,1)',
'tie t 0,60',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.19. script06a.py: Deploying Pitch Sieves with HarmonicAssembly

```
# Deploying Pitch Sieves with HarmonicAssembly
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'pin a 11@1|13@2|23@5|25@6,c1,c7',
'tmo ha',
'tin a 0',
'tie t 0,30',
'tie a rb,.2,.2,.6,1',
'tie b c,120',
#zero-order Markov chains building pulse triples
'tie r pt,(c,4),(mv,a{1}b{3}:{a=12|b=1}),(mv,a{1}b{0}:{a=9|b=1}),(c,.8)',
#index position of multiset: there is only one at zero
'tie d0 c,0',
#selecting pitches from the multiset (indices 0-15) with a tendency mask
'tie d1 ru,(bpl,t,l,[(0,0),(30,12)]),(bpl,t,l,[(0,3),(30,15)])',
#repetitions of each chord
'tie d2 c,1',
#chord size
'tie d3 bg,rc,(2,3)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.20. script07a.py: The CA as a Generator of Melodies

```
# The CA as a Generator of Melodies
from athenaCL.libATH import athenaObj
cmd = [
```

```
'emo m',
# create a single, large Multiset using a sieve
'pin a 5@0|7@2,c2,c7',
'tmo ha',
'tin a 27',
'tie r pt,(c,8),(ig,(bg,rc,(2,3)),(bg,rc,(3,6,9))),(c,1)',
'tie a ls,e,9,(ru,.2,1),(ru,.2,1)',
# select only Multiset 0
'tie d0 c,0',
# select pitches from Multiset using CaList
'tie d1 cl,f{s}x{20},90,0,fria,oc',
# create only 1 simultaneity from each multiset
'tie d2 c,1',
# create only 1-element simultaneities
'tie d3 c,1',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.21. script07b.py: The CA as a Generator of Rhythms

```
# The CA as a Generator of Rhythms
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tin a 47',
# set the multiplier to the integer output of CaList
'tie r
pt,(c,4),(cl,f{s}k{2}r{1}x{81}y{120}w{6}c{0}s{0},109,.05,sumRowActive,oc),(c,1)',
# set the amplitude to the floating point output of CaValue
'tie a cv,f{s}k{2}r{1}x{81}y{120}w{6}c{8}s{0},109,.05,sumRowActive,.2,1',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.22. script08a.py: Evolving African Drum Patterns with a GA: Two Durations

```
# Evolving African Drum Patterns with a GA: Two Durations
```

```
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tmo lg',
'tin a 61',
# bell line, set to loop
'tie r l,[(4,4,1),(4,4,1),(4,2,1),(4,4,1),(4,4,1),(4,4,1),(4,2,1)]',
# accent the first of each articulation
'tie a bg,oc,(1,.5,.5,.5,.5,.5)',
'tin b 68',
# create genetic variations using a high mutation rate
'tie r gr,[(4,4,1),(4,4,1),(4,2,1),(4,4,1),(4,4,1),(4,4,1),(4,2,1)],.7,.25,0',
'tie a bg,oc,(1,.5,.5,.5,.5,.5)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.23. script08b.py: Evolving African Drum Patterns with a GA: Combinations of Rests and Silences

```
# Evolving African Drum Patterns with a GA: Combinations of Rests and Silences
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo mp',
'tmo lg',
'tin a 61',
# kagan line, set to loop
'tie r l,[(4,2,0),(4,2,1),(4,2,1),(4,2,0),(4,2,1),(4,2,1),(4,2,0),
(4,2,1),(4,2,1),(4,2,0),(4,2,1),(4,2,1)]',
# accent the first of each articulation
'tie a bg,oc,(.5,1,.5, .5,.5,.5, .5,.5,.5, .5,.5,.5)',
# turning on silence mode will use parameters even for rests
'timode s on',
'tin b 68',
# create genetic variations using a high crossover, no mutation
'tie r gr,[(4,2,0),(4,2,1),(4,2,1),(4,2,0),(4,2,1),(4,2,1),(4,2,0),
(4,2,1),(4,2,1),(4,2,0),(4,2,1),(4,2,1)],1,0,0',
'tie a bg,oc,(.5,1,.5, .5,.5,.5, .5,.5,.5, .5,.5,.5)',
# turning on silence mode will use parameters even for rests
'timode s on',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
```

```
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.24. script08c.py: Evolving African Drum Patterns with a GA: Multiple Rhythmic Values

```
# Evolving African Drum Patterns with a GA: Multiple Rhythmic Values
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo mp',
'tmo lg',
'tin a 61',
# kroboto line, set to loop
'tie r l,[(4,3,1),(4,1,1),(4,2,1),(4,2,1),(4,1,1),(4,1,1),(4,2,1),
(4,3,1),(4,1,1),(4,2,1),(4,2,1),(4,1,1),(4,1,1),(4,2,1)]',
# accent the first of each articulation
'tie a bg,oc,(1,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5)',
'tin b 68',
# create genetic variations using a high crossover and mutation rate and some elitism
'tie r gr,[(4,3,1),(4,1,1),(4,2,1),(4,2,1),(4,1,1),(4,1,1),(4,2,1),
(4,3,1),(4,1,1),(4,2,1),(4,2,1),(4,1,1),(4,1,1),(4,2,1)],.9,.25,0.1',
'tie a bg,oc,(1,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.25. script08d.py: Evolving African Drum Patterns with a GA: Multiple Rhythmic Values

```
# Evolving African Drum Patterns with a GA: Multiple Rhythmic Values
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tmo lg',
'tin a 45',
'tie r gr,[(4,4,1),(4,4,1),(4,2,1),(4,4,1),(4,4,1),(4,4,1),(4,2,1)],.7,.15,0',
'tie a bg,oc,(1,.5,.5,.5,.5,.5)',
'tin b 60',
# create genetic variations using a high crossover, no mutation
'tie r gr,[(4,2,0),(4,2,1),(4,2,1),(4,2,0),(4,2,1),(4,2,1),(4,2,0),
(4,2,1),(4,2,1),(4,2,0),(4,2,1),(4,2,1)],1,0,0',
'tie a bg,oc,(.5,1,.5, .5,.5,.5, .5,.5,.5, .5,.5,.5)',
# turning on silence mode will use parameters even for rests
'timode s on',
'tin c 68',
```

```
# create genetic variations using a high crossover and mutation rate and some elitism
'tie r gr,[(4,3,1),(4,1,1),(4,2,1),(4,2,1),(4,1,1),(4,1,1),(4,2,1),
(4,3,1),(4,1,1),(4,2,1),(4,2,1),(4,1,1),(4,1,1),(4,2,1)],.9,.25,0.1',
'tie a bg,oc,(1,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5,.5)',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.26. script09a.py: Grammar States as Accent Patterns

```
# Grammar States as Accent Patterns
from athenaCL.libATH import athenaObj
cmd = [
'emo mp',
'tmo lg',
'tin a 60',
# non deterministic binary algae generator applied to accent
'tie r pt,(c,8),(c,1),(gt,a{0}b{1}@a{ab}b{a|aaa}@b,10,oc)',
'tie a c,1',
# four state deterministic applied to pulse multiplier
'tie r pt,(c,8), (gt,a{1}b{2}c{4}d{8}@a{ab}b{cd}c{aadd}d{bc}@ac,10,oc),(c,1)',
# four state deterministic applied to amplitude with different start string
'tie a gt,a{.25}b{.5}c{.75}d{1}@a{ab}b{cd}c{aadd}d{bc}@bbc,6,oc',
# four state deterministic applied to transposition with different start string
'tie f gt,a{0}b{1}c{2}d{3}@a{ab}b{cd}c{aadd}d{bc}@dc,6,oc',
# four state non-deterministic applied to transposition with different start string
'tie f gt,a{0}b{1}c{2}d{3}@a{ab}b{cd|aa}c{aadd|cb}d{bc|a}@dc,6,oc',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.27. script09b.py: Grammar States as Pitch Values

```
# Grammar States as Pitch Values
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'tmo lg',
```

```
'tin a 32',
# four state deterministic applied to pulse multiplier
'tie r pt,(c,8), (gt,a{1}b{2}c{4}d{8}@a{ab}b{cd}c{aadd}d{bc}@ac,8,oc),(c,1)',
'tie o c,-2',
# four state deterministic applied to transposition with different start string
'tie f gt,a{0}b{7}c{8}d{2}@a{ab}b{cd}c{aadd}d{bc}@ad,6,oc',
# four state deterministic applied to amplitude with different start string
'tie a gt,a{.25}b{.5}c{.75}d{1}@a{ab}b{cd}c{aadd}d{bc}@bbc,6,oc',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.28. script09c.py: Grammar States as Pitch Transpositions

```
# Grammar States as Pitch Transpositions
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
'tmo lg',
'tin a 15',
#four state deterministic applied to pulse multiplier
'tie r pt,(c,8), (gt,a{1}b{2}c{4}d{8}@a{ab}b{cd}c{aadd}d{bc}@ac,8,oc),(c,1)',
#four state deterministic applied to accumulated transposition with different start
string
'tie f a,0,(gt,a{1}b{-1}c{7}d{-7}@a{ab}b{cd}c{ad}d{bc}@ac,10,oc)',
# four state deterministic applied to amplitude with different start string
'tie a gt,a{.25}b{.5}c{.75}d{1}@a{ab}b{cd}c{aadd}d{bc}@bbc,6,oc',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.29. script09d.py: Grammar States as Path Index Values

```
# Grammar States as Path Index Values
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
# create a single, large Multiset using a sieve
```

```
'pin a 5@0|7@2,c2,c7',
'tmo ha',
'tin a 6',
# constant rhythm
'tie r pt,(c,4),(c,1),(c,1)',
# select only Multiset 0
'tie d0 c,0',
# select pitches from Multiset using accumulated deterministic grammar starting at 12
'tie d1 a,12,(gt,a{1}b{-1}c{2}d{-2}@a{ab}b{cd}c{ad}d{bc}@ac,10,oc)',
# create only 1 simultaneity from each multiset; create only 1-element simultaneities
'tie d2 c,1',
'tie d3 c,1',
# four state deterministic applied to amplitude with different start string
'tie a gt,a{.25}b{.5}c{.75}d{1}@a{ab}b{cd}c{aadd}d{bc}@bbc,6,oc',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.30. script10a.py: Feedback System as Dynamic Contour

```
# Feedback System as Dynamic Contour
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo mp',
'tmo lg',
'tin a 66',
# constant pulse
'tie r pt,(c,8),(c,1),(c,1)',
# amplitude controlled by Thermostat feedback
'tie a fml,t,(bg,rc,(1,1.5,2))',
# using convert second to set durations
'tie r cs,(fml,t,(c,1),(c,.7),.001,.400)',
# amplitude controlled by Climate Control feedback
'tie a fml,cc,(bg,rc,(.5,1,1.5)),(c,.7),0,1',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.1.31. script10b.py: Feedback System as Path Index Values

```
# Feedback System as Path Index Values
from athenaCL.libATH import athenaObj
cmd = [
'emo m',
# create a single, large Multiset using a sieve
'pin a 5@1|7@4,c2,c7',
'tmo ha',
'tin a 107',
# constant rhythm
'tie r pt,(c,4),(c,1),(c,1)',
# select only Multiset 0
'tie d0 c,0',
# create only 1 simultaneity from each multiset; create only 1-element simultaneities
'tie d2 c,1',
'tie d3 c,1',
# select pitches from Multiset using Thermostat
'tie d1 fml,t,(bg,rc,(1,1.5,2)),(c,.7),0,18',
# select pitches from Multiset using Climate Control
'tie d1 fml,cc,(bg,rc,(.5,1,1.5)),(c,.7),0,18',
]
def main(cmdList=[], fp=None, hear=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2. Csound-based Output

F.2.1. script01a.py: Testing Csound

```
# Testing Csound
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tin a 82',
'tie x6 ws,e,14,0,200,16000',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
```

```
if __name__ == '__main__':
    main(cmd)
```

F.2.2. script01b.py: A Noise Instrument

```
# A Noise Instrument
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tin a 13',
'tie r cs,(whps,e,(bg,rp,(5,10,15,20)),0,.200,.050)',
# set initial low-pass filter cutoff frequency
'tie x2 whps,e,(bg,rp,(5,10,20,2,10)),0,400,18000',
# set final low-pass filter cutoff frequency
'tie x3 whps,e,(bg,rp,(5,10,20,2,10)),0,400,18000',
# panning controlled by fractional noise with infrequent zero-order Markov controlled
jumps out of 1/f2 to 1/f0
'tie n n,100,(mv,a{2}b{0}:{a=12|b=1}),0,1',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.3. script01c.py: A Sample Playback Instrument

```
# A Sample Playback Instrument
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tin a 32',
# set a file path to an audio file
'tie x6 cf,/Volumes/xdisc/_sync/_x/src/martingale/martingale/audio/29561.aif',
# line segment absolute rhythm durations
'tie r cs,(ls,e,(ru,5,30),(ru,.03,.15),(ru,.03,.15))',
# start position within audio file in seconds
'tie x5 ru,0,40',
'tie a ls,e,(bg,rc,(3,5,20)),.1,1',
'tie x2 whps,e,(bg,rp,(5,10,20,2,10)),0,100,10000',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
```

```
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.4. script01d.py: A Sample Playback Instrument with Variable Playback Rate

```
# A Sample Playback Instrument with Variable Playback Rate
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tin a 230',
'tie x6 cf,/Volumes/xdisc/_sync/_x/src/martingale/martingale/audio/32673.aif ',
# line segment absolute rhythm durations
'tie r cs,(ls,e,(ru,10,30),(ru,.05,.25),(ru,.05,.25))',
'tie x5 ru,0,10',
# initial and final audio playback rate
'tie x7 mv,a{1}b{.75}c{.5}d{.2}e{2}:{a=6|b=3|c=2|d=1|e=1}',
'tie x8 mv,a{1}b{.75}c{.5}d{.2}e{2}:{a=6|b=3|c=2|d=1|e=1}',
# panning controlled by fractional noise with infrequent zero-order Markov controlled
jumps out of 1/f2 to 1/f0
'tie n n,100,(mv,a{2}b{0}:{a=12|b=1}),0,1',
'ticp a b',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.5. script02a.py: Composing with Densities using TM TimeFill and a Noise Instrument

```
# Composing with Densities using TM TimeFill and a Noise Instrument
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tmo tf',
'tin a 80',
'tie t 0,30',
# total event count is defined as static texture parameter, not a ParameterObject
'tie s3 600',
# start position within texture normalized within unit interval
'tie d0 rb,.3,.3,0,1',
# durations are independent of start time
'tie r cs,(mv,a{.01}b{1.5}c{3}:{a=20|b=1|c=1})',
'tie a ru,.5,.9',
```

```
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.6. script02b.py: Composing with Densities using TM TimeFill and a Single Sample

```
# Composing with Densities using TM TimeFill and a Single Sample
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tmo tf',
'tin a 32',
# set a file path to an audio file
'tie x6 cf,/Volumes/xdisc/_sync/_x/src/martingale/martingale/audio/27980-high-
slow.aif',
# start position within audio file in seconds
'tie x5 ru,0,1',
# vary a low pass filter start and end frequencies
'tie x2 mv,a{200}b{1000}c{10000}:{a=6|b=2|c=1}',
'tie x3 mv,a{200}b{1000}c{10000}:{a=6|b=2|c=1}',
# total event count is defined as static texture parameter, not a ParameterObject
'tie s3 500',
# start position within texture normalized within unit interval
'tie d0 ic,(rg,.2,.1,0,1),(rg,.7,.1,0,1),(bg,rc,(0,1))',
# durations are independent of start time
'tie r cs,(whps,e,(bg,rp,(5,10,15)),0,.010,.100)',
# must reduce amplitudes
'tie a ru,.1,.3',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.7. script03a.py: Polyphonic Sine Grains LineGroove

```
# Polyphonic Sine Grains LineGroove
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo cn',
'tmo LineGroove',
'tin a 4',
# set a event time between 60 and 120 ms
'tie r cs,(ru,.060,.120)',
# smooth envelope shapes
'tie x0 c,.1',
'tie x1 c,.5',
# set field with a tendency mask converging on a single pitch after 15 seconds
'tie f ru,(ls,t,15,-24,0),(ls,t,15,24,0)',
# set random panning
'tie n ru,0,1',
# create a few more instances
'ticp a b c d e f',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.8. script03b.py: Polyphonic Sine Grains: DroneArticulate

```
# Polyphonic Sine Grains: DroneArticulate
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'apr off',
'tmo DroneArticulate',
# a very large pitch collection made from a Xenakis sieve
'pin a 5@2|7@6,c1,c9',
'tin a 4',
# set a event time between 60 and 120 ms
'tie r cs,(ru,.060,.120)',
# smooth envelope shapes
'tie x0 c,.1',
'tie x1 c,.5',
# set random panning
'tie n ru,0,1',
# reduce amplitudes
'tie a ru,.6,.8',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
```

```
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.9. script03c.py: Polyphonic Sample Grains from a Single Audio File: LineGroove

```
# Polyphonic Sample Grains from a Single Audio File: LineGroove
from athenaCL.libATH import athenaObj
cmd = [
'emo cn',
'tmo LineGroove',
# instrument 32 is a fixed playback rate sample player
'tin a 32',
#set a file path to an audio file
'tie x6 cf,/Volumes/xdisc/_sync/_x/src/martingale/martingale/audio/32673.aif',
# set a event time between 60 and 120 ms
'tie r cs,(ru,.060,.120)',
#smooth envelope shapes
'tie x0 c,.01',
'tie x1 c,.5',
# start position within audio file in seconds
'tie x5 ru,0,10',
# set random panning
'tie n ru,0,1',
# create a few more instances
'ticp a b c d e f',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.10. script03d.py: Polyphonic Sample Grains from a Multiple Audio Files: LineGroove

```
# Polyphonic Sample Grains from a Multiple Audio Files: LineGroove
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
```

```
cmd = [
'emo cn',
'apr off',
'tmo LineGroove',
# instrument 32 is a fixed playback rate sample player
'tin a 32',
# set a file path to an directory, a file extension, and a selection method
'tie x6 ds,/Volumes/xdisc/_sync/_x/src/martingale/martingale/audio,.aif,rp',
# set a event time between 60 and 120 ms
'tie r cs,(ru,.060,.120)',
# smooth envelope shapes
'tie x0 c,.01',
'tie x1 c,.5',
# start position within audio file in seconds
'tie x5 ru,0,10',
# set random panning
'tie n ru,0,1',
# control a variety of amplitudes
'tie a ru,.2,.4',
# create a few more instances
'ticp a b c',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

F.2.11. script03e.py: Polyphonic Sample Grains from a Multiple Audio Files: LineGroove

```
# Polyphonic Sample Grains from a Multiple Audio Files: LineGroove
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()
cmd = [
'emo cn',
'apr off',
'tmo TimeFill',
# instrument 32 is a fixed playback rate sample player
'tin a 32',
# set a file path to an directory, a file extension, and a selection method
'tie x6 ds,/Volumes/xdisc/_sync/_x/src/martingale/martingale/audio,.aif,rp',
# set a event time between 60 and 120 ms
'tie r cs,(ru,.030,.090)',
# smooth envelope shapes
'tie x0 c,.01',
'tie x1 c,.5',
# start position within audio file in seconds
'tie x5 ru,0,10',
# set random panning
'tie n ru,0,1',
# control a variety of amplitudes
```

```
'tie a ru,.1,.2',
# set number of events
'tie s3 1000',
# start position within texture normalized within unit interval
'tie d0 rb,.3,.3,0,1',
]
def main(cmdList=[], fp=None, hear=True, render=True):
    ath = athenaObj.Interpreter()
    for line in cmdList:
        ath.cmd(line)
    if fp == None:
        ath.cmd('eln')
    else:
        ath.cmd('eln %s' % fp)
    if render:
        ath.cmd('elr')
    if hear:
        ath.cmd('elh')
if __name__ == '__main__':
    main(cmd)
```

Appendix G. Frequently Asked Questions

General Information

Q: Can users add Csound instruments to athenaCL?

A: Users can create athenaCL-generated eventLists (scores) with any number of parameter values, allowing the use of external Csound instrument definitions of any complexity.

Q: How can I contribute to this project?

A: If you are a developer and wish to contribute code, add new features, or fix bugs in athenaCL, contact Christopher Ariza via email.

Q: How much does athenaCL cost?

A: athenaCL is a free and open source software project. There is no cost or licensing fee associated with this software.

Q: I have found a bug; what do i do?

A: Report it: the athenaCL interpreter features an integrated bug-reporting system. When quitting athenaCL while an internet connection is available, users may anonymously submit the bug-report.

Q: What does athenaCL do?

A: athenaCL is a tool for computer-aided algorithmic composition, producing outputs for Csound, MIDI, and various other formats.

Q: What is Python?

A: Python is a programming language. Python is a high-level, object-oriented language that is cross-platform, free, and open source.

Q: What is an interactive command-line program?

A: athenaCL is an interactive command line program, which means that instead of using windows, buttons, and the mouse to get things done, the user enters commands and sees text displays. athenaCL is interactive in that, rather than having to give commands with complicated arguments and flags, users are prompted for each entry needed. Unix programs such as Pine and FTP are also interactive command-line programs. Users of UNIX-like operating systems will be familiar with this interface, whereas users of GUI-based operating systems such as Macintosh and Windows may find this interface challenging at first. The athenaCL system is designed to be as intuitive and user friendly as possible; knowledge of programming, UNIX, or other command-line programs, although helpful, is in no way required.

Q: Where can I ask questions about athenaCL?

A: The athenacl Google Group is for users and developers of athenaCL, and can be used to ask questions, get help, or discuss issues related to athenaCL. Users can view and/or subscribe to this list here: http://groups.google.com/group/athenacl. All questions are welcome. Alternatively, users may contact Christopher Ariza directly.

Q: Where is the source code?

A: Every distribution download of athenaCL comes with a complete copy of the source code. Since Python is an interpreted language, the source code can be run "live": there is no executable or binary of athenaCL, the source-code simply runs in the Python interpreter. Developers can get (with SVN) the most recent source at Google Code (http://code.google.com/p/athenacl/). An athenaCL.exe installer is available; this installs athenaCL as a Python package.

Q: Who is athenaCL designed for?

A: athenaCL is designed for use by musicians, composers, sound designers, and programmers. Basic familiarity with stochastics, computer music concepts, and output formats (MIDI, Csound) is helpful.

Installing, Starting, and Uninstalling athenaCL

Q: How do I uninstall athenaCL?

A: To uninstall an athenaCL distribution, in most cases all that is necessary is to delete the athenaCL folder. Windows users who have used an athenaCL installer (athenaCL.exe) will be able to remove athenaCL with the Windows "Add or Remove Programs" Control Panel. On POSIX (UNIX-like) operating systems such as Linux, BSD, and Mac OSX, athenaCL may write a standard configuration file in the user's home directory: ~/.athenaclrc; if an error has occurred during an athenaCL session, a log may be stored in the user's home directory: ~/.athenacl-log; and if the user selected to install the optional athenaCL launcher tool, a script will be found in /usr/local/bin: /usr/local/bin/athenacl. All of these can be removed py using the setup.py script with the argument "uninstall".

Q: Is Csound required to use athenaCL?

A: Csound is only required for rendering audio with athenaCL's built-in library of Csound instrument; MIDI files, as well as other output formats, can always be produced without Csound. Csound is a free, cross-platform tool that renders audio based on instrument definitions in a "orchestra" file and music definitions in a "score" file. As athenaCL provides an integrated library of Csound instruments, no knowledge of Csound is required to use athenaCL.

Q: Is Python required to use athenaCL?

A: Python is required for athenaCL to run, and is not distributed with athenaCL. Python is free, runs on every platform, and comes in easy-to-use installers. Many advanced operating systems (UNIX-

based operating systems including GNU/Linux and MacOS X) ship with Python installed. Visit www.python.org for more information and downloads.

Q: What platforms does athenaCL run on?

A: Because of the cross-platform foundations of the Python programming language, athenaCL runs on every modern platform that Python runs on. This includes Mac OSX, Windows, Linux, BSD and all UNIX-based systems.

Q: Where is the .exe? how do I start a program without a .exe?

A: There is no .exe file. Rather than having an executable binary, athenaCL runs in the Python interpreter. Python is a free programming language available at http://www.python.org. After installing Python, you can launch athenaCL simply by double clicking the file "athenaCL.py"; for more information see the file "README.txt" in the athenaCL directory.

References

Ariza, C. 2002. "Prokaryotic Groove: Rhythmic Cycles as Real-Value Encoded Genetic Algorithms." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 561-567.

———. 2003. "Ornament as Data Structure: An Algorithmic Model based on Micro-Rhythms of Csángó Laments and Funeral Music." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 187-193.

———. 2004. "An Object Oriented Model of the Xenakis Sieve for Algorithmic Pitch, Rhythm, and Parameter Generation." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 63-70.

———. 2005a. *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL*. Ph.D. Dissertation, New York University.

———. 2005b. "Navigating the Landscape of Computer-Aided Algorithmic Composition Systems: A Definition, Seven Descriptors, and a Lexicon of Systems and Research." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 765-772.

———. 2005c. "The Xenakis Sieve as Object: A New Model and a Complete Implementation." *Computer Music Journal* 29(2): 40-60.

———. 2006. "Beyond the Transition Matrix: A Language-Independent, String-Based Input Notation for Incomplete, Multiple-Order, Static Markov Transition Values." Internet: http://www.flexatone.net/docs/btmimosmtv.pdf.

———. 2007a. "Automata Bending: Applications of Dynamic Mutation and Dynamic Rules in Modular One-Dimensional Cellular Automata." *Computer Music Journal* 31(1): 29-49.

———. 2007b. "Serial RSS Sound Installation as Open Work: The babelcast." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association. 1: 275-278.

———. 2008. "Python at the Control Rate: athenaCL Generators as Csound Signals." *Csound Journal* 9.

———. 2009a. "The Interrogator as Critic: The Turing Test and the Evaluation of Generative Music Systems." *Computer Music Journal* 33(2): 48-70.

———. 2009b. "Sonifying Sieves: Synthesis and Signal Processing Applicatinos of the Xenakis Sieve with Python and Csound." In *Proceedings of the International Computer Music Conference*. San Francisco: International Computer Music Association.

Forte, A. 1973. *The Structure of Atonal Music*. New Haven: Yale University Press.

Straus, J. N. 1990. *Introduction to Post-Tonal Theory*. Englewood Cliffs, NJ: Prentice-Hall.

Xenakis, I. 1990. "Sieves." *Perspectives of New Music* 28(1): 58-78.

———. 1992. *Formalized Music: Thought and Mathematics in Music*. Indiana: Indiana University Press.